Democratizing Machine Learning Contributions in AutoML and Fairness

Florian Pfisterer

München 2022

Democratizing Machine Learning Contributions in AutoML and Fairness

Florian Pfisterer

Dissertation
an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig–Maximilians–Universität
München

Eingereicht von
Florian Pfisterer
München, den 30.05.2022

Erstgutachter: Prof. Dr. Bernd Bischl
Zweitgutachter: Prof. Dr. Marius Lindauer
Drittgutacher: Prof. Dr. Eyke Hüllermeier
Tag der mündlichen Prüfung: 19.10.2022

Zusammenfassung

Modelle des maschinellen Lernens sind zunehmend in der Gesellschaft verankert, oft in Form von automatisierten Entscheidungsprozessen. Ein wesentlicher Grund dafür ist die verbesserte Zugänglichkeit von Daten, aber auch von Toolkits für maschinelles Lernen, die den Zugang zu Methoden des maschinellen Lernens für Nicht-Experten ermöglichen. Diese Arbeit umfasst mehrere Beiträge zur Demokratisierung des Zugangs zum maschinellem Lernen, mit dem Ziel, einem breiterem Publikum Zugang zu diesen Technologien zu ermöglichen. Die Beiträge in diesem Manuskript stammen aus mehreren Bereichen innerhalb dieses weiten Gebiets. Ein großer Teil ist dem Bereich des automatisierten maschinellen Lernens (AutoML) und der Hyperparameter-Optimierung gewidmet, mit dem Ziel, die oft mühsame Aufgabe, ein optimales Vorhersagemodell für einen gegebenen Datensatz zu finden, zu vereinfachen. Dieser Prozess besteht meist darin ein für vom Benutzer vorgegebene Leistungsmetrik(en) optimales Modell zu finden. Oft kann dieser Prozess durch Lernen aus vorhergehenden Experimenten verbessert oder beschleunigt werden. In dieser Arbeit werden drei solcher Methoden vorgestellt, die entweder darauf abzielen, eine feste Menge möglicher Hyperparameterkonfigurationen zu erhalten, die wahrscheinlich gute Lösungen für jeden neuen Datensatz enthalten, oder Eigenschaften der Datensätze zu nutzen, um neue Konfigurationen vorzuschlagen. Darüber hinaus wird eine Sammlung solcher erforderlichen Metadaten zu den Experimenten vorgestellt, und es wird gezeigt, wie solche Metadaten für die Entwicklung und als Testumgebung für neue Hyperparameter-Optimierungsmethoden verwendet werden können.
Die weite Verbreitung von ML-Modellen in vielen Bereichen der Gesellschaft erfordert gleichzeitig eine genauere Untersuchung der Art und Weise, wie aus Modellen abgeleitete automatisierte Entscheidungen die Gesellschaft formen, und ob sie möglicherweise Individuen oder einzelne Bevölkerungsgruppen benachteiligen. In dieser Arbeit wird daher ein AutoML-Tool vorgestellt, das es ermöglicht, solche Überlegungen in die Suche nach einem optimalen Modell miteinzubeziehen. Diese Forderung nach Fairness wirft gleichzeitig die Frage auf, ob die Fairness eines Modells zuverlässig geschätzt werden kann, was in einem weiteren Beitrag in dieser Arbeit untersucht wird.
Da der Zugang zu Methoden des maschinellen Lernens auch stark vom Zugang zu Software und Toolboxen abhängt, sind mehrere Beiträge in Form von Software Teil dieser Arbeit. Das R-Paket mlr3pipelines ermöglicht die Einbettung von Modellen in sogenannte Machine Learning Pipelines, die Vor- und Nachverarbeitungsschritte enthalten, die im maschinellen Lernen und AutoML häufig benötigt werden. Das mlr3fairness R-Paket hingegen ermöglicht es dem Benutzer, Modelle auf potentielle Benachteiligung hin zu überprüfen und diese durch verschiedene Techniken zu reduzieren. Eine dieser Techniken, *multi-calibration* wurde darüberhinaus als seperate Software veröffentlicht.

Summary

Machine learning artifacts are increasingly embedded in society, often in the form of automated decision-making processes. One major reason for this, along with methodological improvements, is the increasing accessibility of data but also machine learning toolkits that enable access to machine learning methodology for non-experts. The core focus of this thesis is exactly this – democratizing access to machine learning in order to enable a wider audience to benefit from its potential. Contributions in this manuscript stem from several different areas within this broader area. A major section is dedicated to the field of automated machine learning (AutoML) with the goal to abstract away the tedious task of obtaining an optimal predictive model for a given dataset. This process mostly consists of finding said optimal model, often through hyperparameter optimization, while the user in turn only selects the appropriate performance metric(s) and validates the resulting models. This process can be improved or sped up by learning from previous experiments. Three such methods one with the goal to obtain a fixed set of possible hyperparameter configurations that likely contain good solutions for any new dataset and two using dataset characteristics to propose new configurations are presented in this thesis. It furthermore presents a collection of required experiment metadata and how such meta-data can be used for the development and as a test bed for new hyperparameter optimization methods.

The pervasion of models derived from ML in many aspects of society simultaneously calls for increased scrutiny with respect to how such models shape society and the eventual biases they exhibit. Therefore, this thesis presents an AutoML tool that allows incorporating fairness considerations into the search for an optimal model. This requirement for fairness simultaneously poses the question of whether we can reliably estimate a model's fairness, which is studied in a further contribution in this thesis.

Since access to machine learning methods also heavily depends on access to software and toolboxes, several contributions in the form of software are part of this thesis. The mlr3pipelines R package allows for embedding models in so-called machine learning pipelines that include pre- and postprocessing steps often required in machine learning and AutoML. The mlr3fairness R package on the other hand enables users to audit models for potential biases as well as reduce those biases through different debiasing techniques. One such technique, multi-calibration is published as a separate software package, mcboost.

CONTENTS

Contributions

This cumulative PhD thesis consists of the following contributions:

1. J. N. van Rijn, F. Pfisterer, J. Thomas, B. Bischl, and J. Vanschoren. Meta learning for defaults–symbolic defaults. In *NeurIPS 2018 Workshop on Meta Learning*, 2018

2. P. Gijsbers, F. Pfisterer, J. N. van Rijn, B. Bischl, and J. Vanschoren. Meta-learning for symbolic hyperparameter defaults. In *2021 Genetic and Evolutionary Computation Conference Companion (GECCO '21 Companion)*, page 151–152, Lile, France, 2021. ACM

3. F. Pfisterer, J. N. van Rijn, P. Probst, A. Müller, and B. Bischl. Learning multiple defaults for machine learning algorithms. In *2021 Genetic and Evolutionary Computation Conference Companion (GECCO '21 Companion)*, page 241–242, Lile, France, 2021. ACM

4. M. Binder, F. Pfisterer, and B. Bischl. Collecting empirical data about hyperparameters for data driven AutoML. In *AutoML Workshop at ICML*, 2020

5. F. Pfisterer, S. Coors, J. Thomas, and B. Bischl. Multi-objective automatic machine learning with AutoxgboostMC. In *Automating Data Science Workshop at ECML*, 2019, arXiv:1908.10796

6. J. Moosbauer, M. Binder, L. Schneider, F. Pfisterer, M. Becker, M. Lang, L. Kotthoff, and B. Bischl. Automated benchmark-driven design and explanation of hyperparameter optimizers. *To appear in IEEE Transactions on Evolutionary Computation*, 2022

7. F. Pfisterer, L. Schneider, J. Moosbauer, M. Binder, and B. Bischl. Yahpo gym - an efficient multi-objective multi-fidelity benchmark for hyperparameter optimization. In I. Guyon, M. Lindauer, M. van der Schaar, F. Hutter, and R. Garnett, editors, *Proceedings of the First International Conference on Automated Machine Learning*, volume 188 of *Proceedings of Machine Learning Research*, pages 3/1–39. PMLR, 25–27 Jul 2022

8. L. Schneider, F. Pfisterer, M. Binder, and B. Bischl. Mutation is all you need. In *AutoML Workshop at ICML*, 2021, arXiv:2107.07343

9. L. Schneider, F. Pfisterer, P. Kent, J. Branke, B. Bischl, and J. Thomas. Tackling neural architecture search with quality diversity optimization. In I. Guyon, M. Lindauer, M. van der Schaar, F. Hutter, and R. Garnett, editors, *Proceedings of the First International Conference on Automated Machine Learning*, volume 188 of *Proceedings of Machine Learning Research*, pages 9/1–30. PMLR, 25–27 Jul 2022

10. X. Sun, A. Bommert, F. Pfisterer, J. Rahnenführer, M. Lang, and B. Bischl. High dimensional restrictive federated model selection with multi-objective Bayesian Optimization over shifted distributions. In Y. Bi, R. Bhatia, and S. Kapoor, editors, *Intelligent Systems and Applications*, pages 629–647, Cham, 2020. Springer International Publishing

11. A. Agrawal, F. Pfisterer, B. Bischl, J. Chen, S. Sood, S. Shah, F. Buet-Golfouse, B. A. Mateen, and S. Vollmer. Debiasing classifiers: is reality at variance with expectation?, 2020, arXiv:2011.02407

12. S. Dandl, F. Pfisterer, and B. Bischl. Multi-objective counterfactual fairness. In *GECCO '22: Proceedings of the Genetic and Evolutionary Computation Conference Companion*, page 328–331, Boston, United States of America, 2022. ACM

13. F. Pfisterer, L. Beggel, X. Sun, F. Scheipl, and B. Bischl. Benchmarking time series classification – functional data vs machine learning approaches, 2019, arXiv:1911.07511

14. F. Pargent, F. Pfisterer, J. Thomas, and B. Bischl. Regularized target encoding outperforms traditional methods in supervised machine learning with high cardinality features. *Computational Statistics*, pages 1–22, 2022

15. F. Pfisterer, C. Harbron, G. Jansen, and T. Xu. Evaluating domain generalization for survival analysis in clinical studies. In G. Flores, G. H. Chen, T. Pollard, J. C. Ho, and T. Naumann, editors, *Proceedings of the Conference on Health, Inference, and Learning*, volume 174 of *Proceedings of Machine Learning Research*, pages 32–47. PMLR, 07–08 Apr 2022

16. M. Binder, F. Pfisterer, M. Lang, L. Schneider, L. Kotthoff, and B. Bischl. mlr3pipelines - flexible machine learning pipelines in R. *Journal of Machine Learning Research*, 22(184):1–7, 2021

17. F. Pfisterer, S. Wei, S. Vollmer, M. Lang, and B. Bischl. *Fairness Audits And Debiasing Using mlr3fairness*, Manuscript submitted for publication

18. F. Pfisterer, C. Kern, S. Dandl, M. Sun, M. P. Kim, and B. Bischl. mcboost: Multi-calibration boosting for R. *Journal of Open Source Software*, 6(64):3453, 2021

Acknowledgements

I want to express my gratitude to a number of people without whom this dissertation would not have been possible.
My sincerest thanks to

- my supervisor Prof. Dr. Bernd Bischl for providing me with guidance and support. I am most grateful for the trust and freedom to conduct research in a number of areas I wanted to pursue.
- Prof. Dr. Marius Lindauer and Prof. Dr. Eyke Hüllermeier for acting as referees of this thesis.
- all my current and former colleagues at the Institute for Statistics for their support, feedback, friendship and input. I would especially like to thank Christoph, Xudong, Janek, Martin and countless others for hours of conversations and discussions. Moreover, I would also like to thank Martin, Lennart, Susanne, David, Xudong and Janek for reviewing (parts of) this dissertation.
- all co-authors who sparked my interest in many different areas and provided me with countless new perspectives.
- the mentoring program at the LMU as well as the MCML for financial and academic support.
- my friends, family and my partner for supporting me on the way to and during my dissertation.

CHAPTER 1 INTRODUCTION

The personality of science is neither that of a chivalrous knight nor that of a pitiless juggernaut. What is then science?
Science is a golem. A golem is a creature from Jewish mythology. It is a humanoid made by man from clay and water, with incantations and spells. It is powerful every day. It will follow orders, do your work, and protect you from the ever threatening enemy. But it is clumsy and dangerous. Without control, a golem may destroy its masters with its flailing vigour.

The Golem – what everybody should know about science
Harry Collins & Trevor Pinch

This dissertation comes into existence during a time of unprecedented growth in the field of machine learning and artificial intelligence. This makes for an exciting time to conduct research in this field due to an explosion of new research ideas and possibilities to apply new methods, but simultaneously allows for observing the increasing impact of such systems on individuals or society as a whole from the perspective of an insider. From this point of view, ML models are strikingly similar to the golem of science described by Collins & Pinch in [61] in the quotation above. They can enable humans and extend their capabilities to answer questions that could not be answered before, but they also harbor dangers if we do not carefully consider consequences of questions we ask of them, e.g. if we blindly trust answers derived from miss-specified or imprecise questions.

Machine learning-based systems are embedded in almost all aspects of people's day-to-day life [13] e.g. by personalizing search results for individual users [258] or assisting in medical decision making, e.g. to aid with cancer diagnosis [84] along with a wide variety of other applications [22, 235, 239, 205]. This fusion of society and technology leads to a perspective of technology embedded into society as *socio-technological systems* [64, 7, 219]. It is characterized by interdependency between humans and technological artifacts, in this case ML systems. Humans generate the data necessary for machine learning systems to function, which in turn shapes models learned by ML systems. ML systems, likewise exert control over or influence an individual's life through automated decision making, e.g. by automatically adjusting prices on online platforms or screening applicants for eligibility for a loan [52]. This interplay often results in implicit or explicit feedback loops [164, 58] that need to be carefully assessed and investigated in order to prevent ethically undesirable outcomes such as perpetuating existing injustices into the future or introducing new injustices based on biased models.

On the other hand, the increasing accessibility of machine learning has enabled domain experts to improve processes and policies e.g. in medical decision making [235], public policy [4], social good [221], environmental sustainability [247] or generally natural sciences [48, 207]. There are several reasons for this increased accessibility. The accessibility of data, especially the availability of benchmark datasets and repositories such as Imagenet [72], the *UCI* repository [73] or *OpenML* [245] have played a large role. So have advancements in methodology and computational resources [151]. Another major role that is often overlooked is the availability of data, as well as software i.e. readily available open source implementations of new methodological advances in machine learning toolkits [184, 181, 148]. Especially the latter increasingly enables domain experts without extensive programming expertise or expertise in machine learning to benefit from said advancements. This is further enhanced by the development of automated machine learning (AutoML) tools [122] that automate the often tedious task of obtaining an optimal predictive model for a given dataset. As a result, the amount of time and expertise required for practitioners to develop and deploy models is heavily decreased.

To employ the metaphor of the golem again – ML models are *golems* in a similar sense Collins & Pinch [61] describe science as a golem. They are powerful tools that can help improve their master's and many other people's lives, but can also simultaneously result in fatal consequences if their *commands* are unclear or their consequences are not entirely forethought by the master. This ambiguity therefore requires that the commands given to those *golems* and their potential consequences are thoroughly scrutinized before we unleash these creatures into the world. Similarly advances in ML give us more powerful (and hopefully ultimately more useful) golems. *Democratization* in this context now serves a double purpose – Availability of technology can significantly improve people's lives, e.g. through enhancing decision making, recommendation, or simply by improving access to services or productivity. On the other hand, we would like to prevent those golems from only serving the purposes of a selected few, in this case large companies with a budget to employ machine learning specialists, but instead ultimately lead to an improvement for everybody. Similarly, it is important that we prevent those ever more powerful golems from wreaking havoc on society by ensuring their objectives are *aligned* with societal norms.

It is now important to understand what *democratization* means in the context of this thesis. A definition for *democratization* could therefore be *the process of increasing accessibility to data, compute, methodology and knowledge to a broader audience ranging from citizen data scientists to machine learning researchers.*. It is now vital that democratization is done responsibly – it also needs to consider detrimental consequences to society and the environment. This entails several aspects that constitute prerequisites for developing ML models.

- **Access to data** In order to gain insight into a subject of interest, investigators need access to relevant, up-to-date, and high quality data. This is often problematic, as access to such data is often not available or comes at a steep cost. Solutions for this problem are for example freely available and appropriately licensed data sets and artifacts [173].

- **Access to compute** Similarly developing models requires access to sufficient hardware (e.g. GPUs). This is partially addressed by the availability of cloud compute instances, but especially bigger models are often expensive to train[1] and have considerable environmental impact.

- **Access to methods** ML methods increasingly rely on heavily optimized implementations that e.g. allow to efficiently make use of multiple compute resources. This requires careful and highly optimized implementations (see e.g. [54]). Such implementations are increasingly available as part of ML toolboxes or standalone software.

- **Access to knowledge** Developing ML models furthermore requires expert knowledge regarding problem formulation, the ML method and hyperparameter optimization as well as evaluation protocols. This also holds for methods regarding (fairness) model auditing and interpretability.

Contributions in this thesis tackle the latter two aspects of this democratization process: *AutoML* reduces entry barriers for the development of ML models, *benchmarks* provide a better understanding of which methods to use while *software* provides access to important methods. On the other hand, *fairness* and *robustness* are required to prevent these models from producing unfavorable outcomes either for minorities or disadvantaged populations or in situations where the data distribution changes between training and model deployment.

The typical workflow when developing machine learning models consists of several *processing steps*: It often begins by collecting and verifying data, *prep-processing* it, subsequently fitting a machine learning model and optionally by *post-processing* resulting models or predictions. Those steps involve a multitude of decisions that require careful examination by a machine learning expert in order to arrive at optimal models [150]. Several other steps in the context of model deployment and maintenance exist, they are discussed in

[1]To provide an example, the recent *PaLM* [59] language model uses over 6144 TPU Chips for 1200 hours

more detail in the outlook. To provide an example, during preprocessing we might want to decide on how to impute missing data, how to represent or transform other variables such as e.g. dates or whether to include or exclude certain variables from the analysis. All of those steps include control parameters (subsequently called hyperparameters, denoted by $\boldsymbol{\lambda}_{(\cdot)}$), such as the data imputation technique to use as well as further hyperparameters depending on the technique. During the *Model* phase, a practitioner has to choose between an ever-growing list of potential ML algorithms (or neural network architectures) and simultaneously select the respective algorithm's hyperparameters. After choosing a model, predictions or models can optionally be post-processed (e.g. calibrated). The entire procedure is schematically depicted in Figure 1.1. Several *options*, indicated by blue nodes exist for each step in the pipeline, indicated by orange nodes. Each valid traversal of the graph constitutes a valid configuration of the pipeline that can be evaluated. The evaluation of a configured pipeline is then usually embedded in a *resampling* strategy such as cross-validation [225, 31] in order to obtain unbiased estimates of a desired performance metric(s). The requirement for combining preprocessing steps as well as models and hyperparameters adds considerable complexity, since choosing optimal hyperparameters can have a strong influence on model performance [150].

AutoML in the context of democratization intends to abstract away aspects of this process that require input from ML experts, such as the choice of model (*model selection*) and hyperparameters (*hyperparameter selection*). This problem is introduced as the *Combined Algorithm and Hyperparameter Selection Problem* (CASH) [233] or *Machine Learning Pipeline Configuration* [230]. For the purposes of AutoML, the configuration space is usually restated as a *directed acyclic graph* (DAG). To achieve this, preprocessing is usually rewritten as a fixed number of steps in order to avoid cycles in the graph. Hyperparameters for a node only take effect if the node is selected, result-

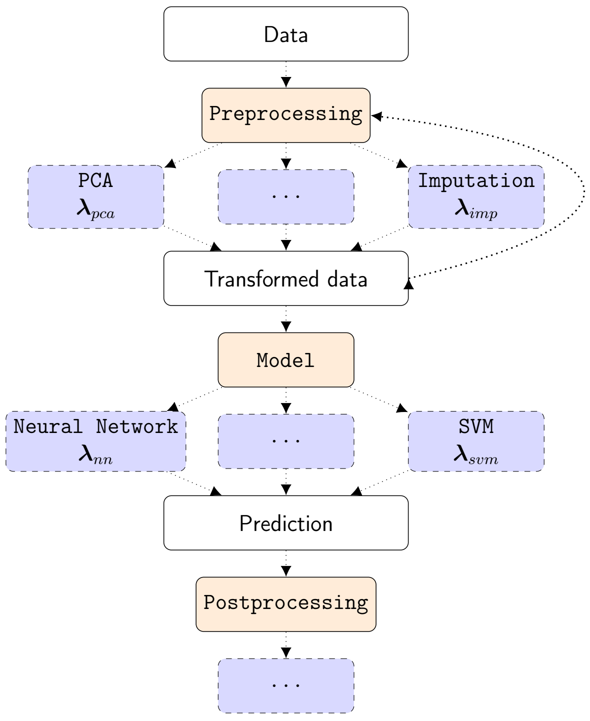

Data
Preprocessing
PCA
$\boldsymbol{\lambda}_{pca}$
...
Imputation
$\boldsymbol{\lambda}_{imp}$
Transformed data
Model
Neural Network
$\boldsymbol{\lambda}_{nn}$
...
SVM
$\boldsymbol{\lambda}_{svm}$
Prediction
Postprocessing
...

Figure 1.1: Schematic of a machine learning pipeline for tabular data. Data is preprocessed using a selected preprocessing step with respective hyperparameters (e.g. PCA and $\boldsymbol{\lambda}_{pca}$) before fitting a model, again with respective hyperparameters before optionally being post-processed. Several preprocessing steps can be combined or selected, depending on the variable.

ing in dependencies. This combination of discrete choices, numerical hyperparameters and dependencies results in a *mixed type, hierarchical* hyperparameter space. Our goal now is to find an optimal (or good enough) configuration within this search space. Instead of manual trial and error or relying on expert knowledge, AutoML systems usually solve this problem using black-box optimization techniques. The goal of an AutoML system usually is to yield a model with optimal generalization error within a given allotted time window, often with an additional constraint on computational resources. This is exemplified in the AutoML benchmark [100], where AutoML systems compete under time- and resource constraints on a set of pre-defined datasets in order to determine which systems actually work and to track progress in the field. Several AutoML tools have been proposed in recent years. They mainly differ in their approach to solving the CASH problem induced by the available processing steps but also in many other details such as whether the resulting models are stacked or ensembled [254, 51], whether the search process is initialized via meta-learning [253, 90, 244], or whether the tools make use of *low-fidelity* approximations [266]. Tools use different approaches towards solving the CASH problem, such as *Bayesian Optimization*[92, 91], Evolutionary Algorithms [178, 103] or planning [170].
An important aspect of AutoML studied in more detail in this manuscript is *Meta-Learning* [246, 40, 244], often also called *learning to learn* [211, 117, 234]. The goal of meta-learning is to use meta-data about machine learning experiments to better understand learning algorithms and subsequently use this *meta-knowledge* to robustify, speed up, or improve learning or optimization. In the context of AutoML, this often means using collected data about the performance of fitted models and hyperparameters across a broad set of learning tasks in order to *warm-start* optimization [253, 90] by e.g. focusing on configurations that have performed well in previous experiments in the hope to find promising solutions early and focus subsequent optimization on relevant reasons.
The aforementioned machine learning pipelines-based approach is mostly applicable for data modalities that admit an efficient tabular representation. Deep learning models on the other hand are popular for other data modalities such as images or text. Neural Architecture Search (NAS) tries to find optimal neural network architectures for deep learning models. The distinguishing factor between NAS and traditional ML pipeline optimization is that *a*) preprocessing is largely absorbed by the neural network, and *b*) all models are part of the same model class and *c*) it allows for switching between *first order* and *second order* optimization since access to gradients is available. The NAS problem thus often considers the optimization over a DAG defined by neural network modules and layers [157, 81], where nodes can potentially share weights between different instantiations of a neural architecture. Similarly to the CASH problem, optimization is done via e.g. *Bayesian Optimization*[250], evolutionary algorithms [203] or reinforcement learning [157, 269]. In order to facilitate exposition, if not explicitly stated, NAS will be treated as a subcategory of the more general AutoML framework throughout the remainder of this manuscript.

The only user input to an AutoML system – other than the data and computational budget – is the evaluation protocol and the performance metric to optimize for. This is an often critically overlooked input, since it allows the user to encode problem specifics as well as to specify the constraints required to make a solution acceptable [79]. Especially human subjective evaluations of model quality often widely disagree with classical performance metrics (cf. [106]). In other scenarios such as automated decision making, where humans are subject to model predictions, metrics again pose a central role when we ask how to evaluate whether a model is *biased* against certain subpopulations [232]. From this perspective, AutoML could be further extended to contexts where the solution quality is not easily quantifiable or where optimization with respect to multiple metrics is required [124, 187]. This aspect will be more thoroughly discussed in Section 3.1.

While advances in AutoML often come from improved optimizers or better engineering, a better understanding of the individual components and how they interact can similarly lead to performance improvements. For this reason, benchmarks of individual components constitute an important aspect of advancing AutoML research. This, on the one hand, allows for reducing the search space to components that demonstrably work on typical workloads, thereby drastically increasing efficiency. On the other hand, benchmarking helps advance understanding of the underlying functionality by allowing to track whether novel, often increasingly complex methods actually improve over simple baselines. Third, an improved understanding of *what* actually works could possibly lead to even better components in the future. For example, [212] analyze the performance of stochastic gradient descent based optimizers for deep learning, finding that none of the newly published optimizers consistently beat established baselines. This allows practitioners to select from a small subset of optimizers instead of having to include a large set of (possibly ineffective) optimizers. Simultaneously, benchmarks serve as a pulse for advancement in a scientific discipline and therefore constitute an important part of the scientific process. Replication of published results helps to ensure validity and reproducibility in science [35, 177]. This calls in particular for benchmark studies conducted by researchers who are impartial in their conclusions, so-called *neutral* benchmark studies [34] in order to avoid *cherry picking* or *rigging the lottery* [218, 71].
In order to diffuse new scientific results in the realm of machine learning to a more general audience, an important step is the availability of user-friendly software, essentially allowing a broader audience to make use of implemented functionality. As a result, large open source software libraries such as *WEKA* [118, 110] *scikit-learn* [184], *mlr* [30], *MLJ* [32] and neural network libraries such as *pytorch* [181] and *tensorflow* [2] that often in turn build upon existing libraries and programming languages such as *R* [229], *Python* [243], *Java* [10] or *Julia* [23] have been used in a considerable amount of scientific publications. Rigorously validated implementations of algorithms and methods in such frameworks furthermore improve reproducibility due to clearly defined software versions (and algorithm details) and less chance of implementation errors in comparison to custom implementations. Implementations of algorithms are especially important in areas such as algorithmic fairness [13] where several software implementations required to audit machine learning models for potential biases are available [210, 18, 27]. Along with raising awareness for said problems, readily available and easy to use tools are vital for detecting and alleviating potential fairness issues in ML systems, which could prevent ethically questionable prac-

tices as well as address (legal) compliance problems.

The concept of algorithmic fairness constitutes the last element of this manuscript. The goal of algorithmic fairness is to *detect* and possibly alleviate ethical issues related to discrimination in ML systems [13, 227, 168]. Fairness eludes a single and precise definition because what constitutes fairness depends on ethical perspective and cultural context [138, 93] – in other words, fairness is highly situational.
Trying to find a single definition, Mehrabi et al. [165] define fairness as *"the absence of any prejudice or favoritism towards an individual or a group based on their inherent or acquired characteristics. Thus, an unfair algorithm is one whose decisions are skewed toward a particular group of people"*.

Several types of biases can lead to unfairness in a fitted model. Societal biases can occur due to data reflecting historical biases, e.g. judiciary decisions that historically favor specific ethnic groups [60]. Measurement biases on the other hand, can stem from the under or over representation of specific groups in the collected data, for example, due to increased policing in specific areas [248, 104] or differential access to services [149]. *Statistical bias* can creep in when the model does not accurately reflect the physical data-generating process, e.g. due to model under- or overspecification. Other biases include differences in collected data between groups, e.g. due to differing standards of documentation. A third bias, here called learning bias, is introduced during the modeling stage, where the choice and specification of ML model can add biases that later manifest in disparate predictions (e.g. due to model under- or overspecification). The full process is detailed in Figure 1.2. At each step, the difference between the *world as it should be* and *world as it is* can introduce biases into the final trained model. It is important to note that how the *world as it should be* looks like is a highly normative question that depends on the ethical perspective. As a result, judging whether societal bias – a systematic mismatch between the modeled and an optimal world exists, requires ethical, legal and political considerations. Other sources of bias, in turn, are more technical in nature. Machine learning

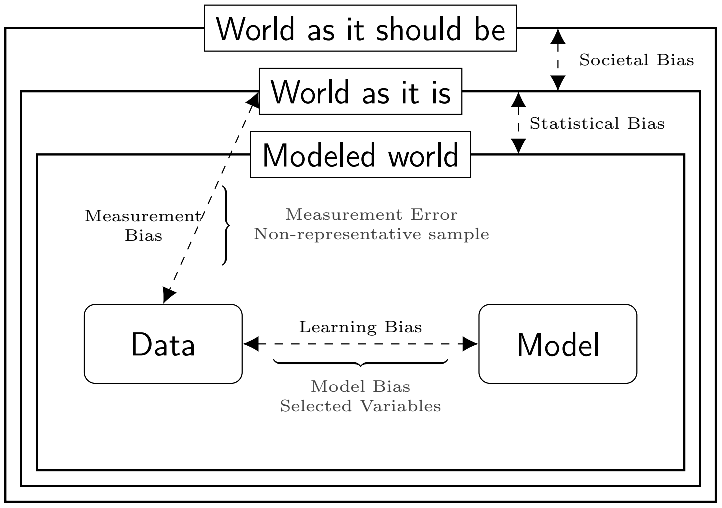


Figure 1.2: Types and sources of bias in ML models adapted from [17, 168, 13]. Two main sources of bias between the *ideal world* and the *modeled world* exist: societal and statistical bias, the latter of which encompasses other sources of bias (measurement bias and learning biases).

often focuses on this technical aspect – reducing the systematic mismatch between modeled and observed world [17].
Therefore, an important first task is to detect potential *biases* in ML models. As a result, a large body of work has proposed different fairness metrics measuring deviations from fairness that can be used in order to *audit* predictive models. The choice of metric depends on the problem context (see, e.g. [165] for a survey and [210] for a flow chart). One important chasm in this context is the differentiation between *causal* notions of fairness, e.g. notions that rely on modeling the mechanistic data generating process [147, 134, 263, 262, 55] and notions that rely on *observational* data [49, 65, 22, 112, 57]. While only a causal perspective can realistically assume fairness, those notions require access to the causal *directed acyclic graph* (DAG, c.f. [182]) underlying the data generating process. The exact specification of DAGs, especially in the context of high dimensional data is often highly ambiguous (see e.g. [183] for a discussion), leading to a gap in applicability. For this reason, observational notions, such as measures focusing on differences in performances between groups, are often prefered in practice.
Should such problems be detected, several so-called *debiasing techniques* [128, 260, 259, 112, 113] promise to alleviate biases in ML models. Similar to the schema for ML pipelines in Figure 1.1, this can happen in the different stages of the ML pipeline. preprocessing methods [128, 129] change or augment the data that is fed to the subsequent models, e.g. by balancing the number of data points in the protected groups or by yielding fair representations of data [161]. *In-processing* techniques are ML models that include fairness constraints as part of the model fit [260, 259] while *post-processing* techniques adjust predictions of fitted models in order to improve or account for different fairness criteria [112, 113].
It is important to note that fairness metrics as well as bias mitigation techniques have several problems that need to be considered when being used in order to audit predictive models for bias: Metrics assume that the relevant ethical perspective can be condensed into numeric values [63]. Furthermore, there are often better techniques that can be used to decrease bias in ML models like collecting additional data [47, 63]. Differences between the actual world and its representation in the data can act as sources of bias that is impossible to spot given only observational data, strengthening calls for improved documentation of data artifacts [98]. Furthermore, bias mitigation often requires trading off notions of utility (e.g. accuracy) and fairness [267, 249] (see e.g. [206] for a contrasting observation).
In the context of democratization, it is now important that not only the capability to build powerful models, but simultaneously the capability to audit existing models for biases and mitigating biases where required, is made available to a broad audience. While AutoML can only reasonably hope to solve parts of the multiplicity of possible problems and solutions, it is nonetheless an interesting area of application that can hopefully lead to fair(er) models.

Outline

The remainder of this thesis is structured as follows: Chapter 2 introduces the core concepts and the required notation for this thesis. Section 2.2 introduces *single- and multi-objective* *optimization* as well as Neural Architecture Search (NAS) and automated machine learning (AutoML) before briefly presenting the contributions made as part of this thesis. In Section 2.3, different notions of *algorithmic fairness* are introduced along with *bias mitigation techniques* that can help to improve ML models with respect to fairness considerations. It again concludes by presenting contributions in this thesis. Section 2.4 then introduces several contributions in the area of *benchmarking* as well as software developed as part of this thesis. Finally, Chapter 3 attempts to cast a wider net by looking at the field of AutoML, considering also its role in democratizing access to machine learning methods and its intersection with research conducted in the area of algorithmic fairness. We conclude by presenting individual contributions in Sections 4.1 to 6.6.

CHAPTER 2

BACKGROUND

The goal of this section is to set the scene for the contributions detailed throughout this thesis and to formally introduce some of the required notation. It furthermore introduces the reader to the core topics and perspectives taken throughout the contributions part of this thesis: Hyperparameter Optimization, AutoML, Fairness, Benchmarking and Software.

2.1 Setting & Notation

Contributions in this thesis are usually set in a supervised learning setting. That is, we are interested in estimating the functional relationship between a set of features $\mathbf{x}$ and a target variable of interest $\mathbf{y}$. Together they comprise a dataset $\mathcal{D} = \big((\mathbf{x}^{(i)}, y^{(i)}); i \in \{1, ..., n\}\big) \in (\mathcal{X} \times \mathcal{Y})^n$, consisting of n observations often assumed to be drawn independently and identically distributed (i.i.d.) from a data-generating distribution $\mathbb{P}_{xy}$. With slight abuse of notation, we will also denote with $\mathbf{x}$ and y random variables instead of realizations of the random variable. What is meant is made clear from context. Features $\mathbf{x}$ and the target variable(s) y can take different forms throughout this manuscript, such as images and text, but are generally assumed to be categorical or numeric values in tabular format if not indicated differently.

ML models $\hat{f}$ are obtained by training an *inducer* algorithm or learner $\mathcal{I}$ that constructs $\hat{f}$ based on a training data set $\mathcal{D}$ often controlled by *hyperparameters* $\boldsymbol{\lambda}$. This process is often refered to as fitting the model. The resulting model $\hat{f} : \mathcal{X} \rightarrow \mathcal{Y}'$ assigns a prediction $\hat{f}(\mathbf{x})$ to each feature vector $\mathbf{x}$. We now want this prediction to be as close to y as possible, as indicated by a performance metric or loss function L that measures the differences between the predictions and the ground truth: $L : \mathcal{Y} \times \mathcal{Y}' \rightarrow \mathbb{R}$. In this work, generally no differentiation between different *inducer* algorithms or *model classes* is made expecting that the process of obtaining $\hat{f}$ is a black-box process that can only be controlled by adjusting $\boldsymbol{\lambda}$. Throughout this work, we will refer to the *inducer* as a learner or ML algorithm, while

we will call $\hat{f}$ the (ML-) model produced or *fitted* learner.
Since we require that our model generalizes to future data unseen during training, our goal is to minimize the expected error on new samples from the data distribution $\mathbb{P}_{xy}$, the *generalization error*:

$$GE = \mathbb{E}_{(\mathbf{x},y)\sim\mathbb{P}_{xy}}\left[L(y, \hat{f}(\mathbf{x}))\right] \tag{2.1}$$

Since this quantity is not estimable directly without access to the data distribution $\mathbb{P}_{xy}$, we instead obtain an empirical estimate of GE by leaving out portions of the data during fitting either in the form of *train-test* splits, or in the form of more involved resampling techniques like *cross-validation* (cf. [225, 31]) that allow for obtaining estimates $\widehat{GE}$ by (iteratively) fitting on parts of the data and evaluating on the held-out part of the dataset. The estimate $\widehat{GE}$ is therefore a function of the inducing algorithm $\mathcal{I}$, hyperparameters $\boldsymbol{\lambda}$, the dataset $\mathcal{D}$, the loss function L and the estimation procedure. During optimization, we assume that hyperparameters other than $\boldsymbol{\lambda}$ are constant and therefore omit them in the notation defining the estimated generalization error $\widehat{GE(\boldsymbol{\lambda})} : \Lambda \rightarrow \mathbb{R}$ as a function of $\boldsymbol{\lambda}$ only.
We consider loss functions $L : \mathcal{Y} \times \mathcal{Y}' \rightarrow \mathbb{R}$ that take as input the true labels y and prediction scores $\hat{y} \in \mathcal{Y}'$. For ease of exposition, we focus on *classification* and *regression* scenarios where we assume $\mathcal{Y}'$ to either be $\mathbb{R}^g$ (a g dimensional vector of prediction scores for g classes; $\mathbb{R}$ for binary classification) that can be translated to predicted labels y' in a classification scenario or $\mathbb{R}$, the predicted response for regression [28]. Popular choices for such loss functions are *accuracy*, *true*, and *false positive rates* or *area under the ROC curve* (cf. [111]) for classification or *mean squared error* and *mean absolute error* for regression. Note, that the definition in Equation (2.1) does only hold for point-based losses, [28] provide a more general formulation for set-based losses like AUC.

First, second and third level optimization An important distinction between *first*, *second* and *third level* optimization has to be made at this point. First order optimization learns or optimizes model parameters via the inducer algorithm, e.g. coefficients in a linear model through maximum likelihood estimation, splits in a decision tree through recursive partitioning [43] or neural network weights through stochastic gradient descent. We will denote these *learned model parameters* with $\hat{\theta}$ in the remainder of the manuscript where required for differentiation. First-order optimization is often controlled by hyperparameters $\boldsymbol{\lambda}$ that influence the way the model is generated. While the model parameters $\hat{\theta}$ are an output of the inducer $\mathcal{I}$, HPs $\boldsymbol{\lambda}$ are an input. In contrast, the goal of second-order optimization, also often called *hyperparameter optimization* (HPO), is finding a number of hyperparameters $\boldsymbol{\lambda}^\star$ that lead to a model $\hat{f}(\mathbf{x})$ which is optimal with respect to one or multiple performance metrics. This work concerns itself primarily with *second order* optimization. Without loss of generality, we assume that the objective function $c(\boldsymbol{\lambda})$ should be minimized in the remainder of this manuscript if not indicated differently. As a third level, we consider *optimizing the optimizer*: In practice, we are interested in an HPO optimizer (or AutoML system) that works across a large variety of datasets $\mathcal{D}$ coming from a distribution of datasets $\mathbb{D}$. This can be achieved by configuring hyperparameters[1] $\boldsymbol{\gamma}$ of the optimization algorithm. This third optimization level is often solved using *algorithm configuration* [121, 119, 159].

Domain Generalization While the scenario described above covers a large portion of problems typically encountered in practice, several interesting problems that violate or extend assumptions made above exist. One such setting is *domain generalization*:
The field of *domain generalization* studies settings where data do not come from a joint distribution $\mathbb{P}_{xy}$ but instead from several different distributions. In this case, we denote by $\mathbb{P}_{xy}^{(j)}$ the distribution from which we have sampled a data set $\mathcal{D}^{(j)}$. This is frequently encountered in real-world applications. Data can e.g. be a collection of data sets $\{\mathcal{D}^{(1)}, ..., \mathcal{D}^{(J)}\}$ clinical records originating from J different hospitals that each provide care for different patient populations. In this case, shifts in $\mathsf{P}(X)$ also called *data drift* (e.g. due to differences in population) and $\mathsf{P}(Y|X)$ (e.g. due to differences in treatment) can occur. We are now interested in producing a model $\hat{f}$ that has a low generalization error $\widehat{GE}$ estimated on a newly sampled dataset $\mathcal{D}^{(J+1)}$ from available datasets $\{\mathcal{D}^{(1)}, ..., \mathcal{D}^{(J)}\}$. A large variety of methods to address this scenario exist, a survey providing an overview over methodology is provided in [265]. It is important to note, that optimization in this context can happen at the *first* level (by adapting the training procedure) and at the *second* level (by adapting the tuning procedure), in both cases yielding a model $\hat{f}$.

[1]γ denotes third-level hyperparameters, often also called hyper-hyperparameters in literature.

2.2 Hyperparameter Optimization

Hyperparameter configurations (HPC) $\boldsymbol{\lambda} \in \Lambda$ control the *training* of the inducer $\mathcal{I}$ and therefore affect the estimated performance of the resulting model $\hat{f}$. A hyperparameter configuration $\boldsymbol{\lambda}$ consists of settings for each individual hyperparameter $\lambda_i \in \Lambda_i, i \in \{1, ..., d\}$:

$$\boldsymbol{\lambda} := (\lambda_1, ..., \lambda_d)$$

Optimization is usually done over a fixed *search space* $\tilde{\Lambda} = \tilde{\Lambda}_1 \times ... \times \tilde{\Lambda}_d \subseteq \Lambda$, where $\tilde{\Lambda}_i$ is the (usually bounded) search space for hyperparameter λ_i [28]. $\tilde{\Lambda} \subseteq \Lambda = \Lambda_1 \times ... \times \Lambda_d$ is usually a subspace of the set of all possible HPC. Individual hyperparameters are often integer, real-valued, or categorical. If both types occur, a search space is said to be mixed, while hyperparameters can also depend on each other, in which case it is said to be *hierarchical* [233, 28]. This can occur, for example, if a hyperparameter λ_i is only active if a given condition holds for a different hyperparameter λ_j.

2.2.1 Single-Objective Optimization

We introduce the optimization problem of finding an optimal configuration $\boldsymbol{\lambda}^\star$ for a single criterion $c : \Lambda \to \mathbb{R}$ following Bischl et al. [28]:

$$\boldsymbol{\lambda}^\star \in \underset{\boldsymbol{\lambda} \in \tilde{\Lambda}}{\arg\min}\, c(\boldsymbol{\lambda}). \tag{2.2}$$

Here c is an optimization criterion or objective function that maps $\boldsymbol{\lambda}$ to the target metric (for example, $\widehat{GE}$ for a fixed loss function L) with respect to which $\hat{f}$ should be optimal. In the context of *Hyperparameter Optimization*, we usually consider the objective function $c(\boldsymbol{\lambda})$ to be *i*) expensive to evaluate and *ii*) a *black-box function*. That is, *i*) when evaluating a configuration $\boldsymbol{\lambda}$, we obtain no additional information (e.g. gradients) that might help with optimizing the objective function and *ii*) evaluating $c(\boldsymbol{\lambda})$ incurs considerable cost (e.g. computational cost) . Therefore, our goal is to find a good configuration $\boldsymbol{\lambda}^\star$ using as few evaluations of our objective function c as possible.
Several methods for solving the single objective optimization problem have been proposed. Main variations between proposed methods are how each new candidate point for evaluation $\boldsymbol{\lambda}$ is selected and how the optimization result is returned.
A generic template for iterative optimization methods inspired by [171] is given in Algorithm 1. They iteratively evaluate the objective function $c(\boldsymbol{\lambda})$ with the goal to determine the optimal configuration $\boldsymbol{\lambda}^\star$. A wide variety of existing optimization methods can be understood as instantiations of this schema. Methods usually start by iteratively proposing a new candidate configuration for evaluation, evaluating it, and writing results to an archive in order to keep track of previously evaluated configurations. This is repeated until a stopping criterion (e.g. number of function evaluations or a prespecified maximum runtime) is met. In the end, an optimal configuration $\boldsymbol{\lambda}^\star$ is proposed, often by taking the configuration that resulted in the best objective function value from the archive. The pseudocode

Algorithm 1: Generic pseudo code for a iterative single-objective optimizer.

```
Data: Objective $\mathbf{c}$; Search Space $\tilde{\Lambda}$; Budget $b$.
Result: Optimal configuration $\hat{\boldsymbol{\lambda}}^\star$
$A \leftarrow \{\}$
while not stopping_criterion(A, b) do
    // Propose a new candidate for evaluation:
    $\boldsymbol{\lambda} \leftarrow propose\_candidate(A, \tilde{\Lambda})$
    // Evaluate $\boldsymbol{\lambda}$:
    $c_\lambda \leftarrow \mathbf{c}(\boldsymbol{\lambda})$
    // Write results to archive:
    $A \leftarrow A \cup \{(\boldsymbol{\lambda}, c_\lambda)\}$
end
// Return results
$\boldsymbol{\lambda}^\star \leftarrow propose\_result(A)$
```

in Algorithm 1 omits two details for the sake of clarity: First, several methods require additional state variables (e.g. the current iteration) and second proposal and evaluation of candidates can happen in batches evaluated in parallel or asynchronously. It furthermore absorbs initialization required for some algorithms into the *propose_ candidate* function.

Grid Search and Random Search are both widely used in the context of HPO due to their simplicity and ease of implementation. *Grid search* evaluates points on a multidimensional grid of (usually equally spaced) points along each axis. *Random search*, in contrast generates configurations by sampling random points from each dimension independently. Both can be expressed within Algorithm 1 by adopting the *propose_candidate*() to generate points, either from a grid specified on $\tilde{\Lambda}$ or at random respectively. Bergstra et al. [21] show that random search should be preferred over grid search, because grid search experiments focus too much on the exploration of dimensions that do not matter and simultaneously suffer from poor coverage in important dimensions. We now consider two more involved optimizers, *Bayesian Optimization* and *Evolutionary Algorithms*.

Bayesian Optimization (BO) [169, 127, 15, 224] employs so-called *surrogate models* to propose next candidate points for evaluation. Surrogate models are regression models trained on a meta-dataset of previously evaluated configurations and corresponding performances $\mathcal{D} = \{(\boldsymbol{\lambda}^{(i)}, c(\boldsymbol{\lambda}^{(i)}))\}, i \in \{1, ..., N\}$ with the goal to approximate the global relationship between $\boldsymbol{\lambda}$ and the performance metric of interest $c(\boldsymbol{\lambda})$. Widely used surrogate models are Tree-structured Parzen Estimators [20], Gaussian Processes [169, 224] or Random Forests [127, 120], while the right choice of surrogate model is often problem- and search-space dependent. *Bayesian Optimization* methods often make use of the surrogate model's *uncertainty* (e.g. variance of the posterior prediction in a Gaussian Process) to trade off exploring regions of the search space with high uncertainty and exploiting regions

where good performing configurations are expected.
Future candidate configurations are now chosen based on the so-called *infill criteria*, often also called *acquisition function* [44]. A variety of *myopic* infill criteria, such as Probability of Improvement [146], Expected Improvement [169], Lower/Upper Confidence Bound (LCB) [66] have been proposed along with several *non-myopic* variants such as Entropy Search [114] and Predictive Entropy Search [116] that do not consider the optimal point in the next iteration but instead propose candidates that are optimal when looking ahead multiple iterations into the future.
Given a surrogate model fitted on a dataset of previously evaluated configurations, we can now obtain predictions of the posterior mean μ and variance σ^2 for each configuration $\boldsymbol{\lambda} \in \tilde{\Lambda}$. We can now present the Lower Confidence Bound Criterion (LCB) [66]

$$LCB(\boldsymbol{\lambda}) = \mu(\boldsymbol{\lambda}) - \kappa\sigma(\boldsymbol{\lambda}). \tag{2.3}$$

Intuitively, the LCB criterion trades off exploring in regions with a large posterior standard deviation σ and exploitation in regions with low predicted mean μ. This search can be further guided using a hyperparameter κ. Using the LCB criterion, we can now propose new evaluation candidates by solving the inner infill optimization problem

$$\arg\min_{\boldsymbol{\lambda} \in \tilde{\Lambda}} LCB(\boldsymbol{\lambda}). \tag{2.4}$$

Taking into account, that obtaining predictions from the surrogate model is *cheap*, it can be solved using derivative-free optimizers, e.g. the DIRECT algorithm (as done, e.g. in [127, 44] or a simple random search with a large budget).
We can mold BO into the pseudo-code provided in Algorithm 1 by specifying a *propose_candidate* method. This method is detailed in Algorithm 2, where the *optimize_infill* method could e.g. choose $\boldsymbol{\lambda}$ based on optimizing the LCB criterion in Equation (2.4). If no sufficient number of configurations for fitting the surrogate model exist in the archive ($|A| < k$), the *propose_initial_point* proposes initial data points. This initial data set of configurations required to fit surrogate models is usually generated using a fixed number of configurations drawn at random from $\tilde{\Lambda}$ or using experimental design techniques (e.g. *Latin Hypercube Sampling* [120]).
The basic principle of *Bayesian Optimization* has been extended in multiple directions: Several methods [224, 228, 152] consider settings where the evaluation of candidate configurations incurs varying costs, while other extensions focus on *(batch-)parallel* [105, 256, 251] and *asynchronous* evaluation [131]. [175, 163] in turn study early stopping of *Bayesian Optimization* methods. Further methods that extend BO for multi-fidelity and multi-objective will be discussed in separate chapters below.

Evolutionary Algorithms Evolutionary Algorithms (EAs) are iterative methods that maintain and update a population of candidate solutions [11, 69, 14]. They are inspired by the basic principle of *survival of the fittest* in natural evolution, maintaining a population of individuals that compete for survival and generate offspring in order to arrive at (fitter)

Algorithm 2: *propose_candidate* method for *Bayesian Optimization*

```
Data: Archive $A$, Search space $\tilde{\Lambda}$
Result: Candidate configuration $\boldsymbol{\lambda}$
if $|A| < k$ then
    $\boldsymbol{\lambda} \leftarrow propose\_initial\_point(A, \tilde{\Lambda}))$
else
    // Fit the surrogate model
    $\hat{f} \leftarrow fit\_surrogate(A)$
    // Solve infill optimization problem
    $\boldsymbol{\lambda} \leftarrow optimize\_infill(\hat{f}, \tilde{\Lambda})$ // e.g. Equation (2.4)
end
```

individuals. EA methods usually start by generating an initial population (often randomly generated) and assessing each individual's fitness - measured via the objective function to be optimized. Then, individuals with high fitness value are selected for survival and used to generate new evaluation candidates, so-called *offspring* via cross-over (combining population members) and mutation (random perturbations of a population member). This procedure is repeated iteratively until a stopping criterion is met. We can mold *Evolutionary Algorithms* into the pseudo-code provided in Algorithm 1 by adapting it in several ways: The archive A now has to keep track of which individuals are alive within each given generation and the *propose_candidate* method needs to generate new points from individuals alive in the current generation.

EAs are widely used because they do not make strong assumptions about the objective function's structure, are easy to implement, easy to parallelize, and generally robust [3]. In contrast to *Bayesian Optimization*, EAs can be applied to even richer and more complex search spaces, since they do not require surrogate models, but can instead be adapted using specialized initialization and mutation operators. To provide an example, [144] use *Genetic Programming*, an extension of EAs [14] to optimize formulas that describe mathematical identities from a dataset of inputs and outputs (e.g. the trigonometric identity $\cos(2x) = 1 - 2 \cdot \sin^2(x)$). Genetic Programming allows a variable-length representation of solutions, e.g. through representation of solutions as binary trees with differing node depths [179]. The space of available solutions can then be described by a grammar of *symbolic expressions* `EXPR(SYMB)` where each expression `EXPR` and symbol `SYMB` can be substituted with either another expression (e.g. sin, $+$, $-$ in the example above) or terminal nodes such as variables or constants (x and 1 in the example above). Expressions can be unary (e.g. *sin*), binary ($+$), or take a variable number of inputs (n-ary). Iteratively substituting expressions and symbols can then result in arbitrarily complex formulas. Optimization over the space of binary trees induced by a given grammar can now again be done using principles from EAs. Expressions and Symbols can be altered by mutation (e.g. replaced by a different symbol allowed by the grammar) or cross-over (e.g. by combining randomly chosen (sub-)trees from two or more parent programs).

Multi-Fidelity Methods

In many settings, ML algorithms allow for obtaining comparatively cheap approximations of $\widehat{GE}$. Such approximations can e.g. be obtained by training only a fraction of the available dataset [185], by training iterative algorithms (for example, neural networks or gradient-boosting machines) for only few iterations, or by evaluating only a subset of all cross-validation folds [233]. The parameters controlling the computational cost are called *budget parameters*, which are often low initially and then progressively increase until evaluations are made at *full-fidelity*. For ease of exposition, we assume that there only exists a single such budget parameter, denoted by $\boldsymbol{\lambda}_{budget} \in \Lambda$. The more general setting, where we can cheaply query a second, correlated source is called multi-information source optimization [199]. In contrast, problems are assumed to be multi-fidelity if there exists a strict hierarchy of budgets [199] [2].
So-called *low-fidelity* evaluations can often speed up optimization, as unpromising configurations can be stopped early; e.g. [133, 123] proposes iteratively stopping half of evaluated configurations while doubling the available budget for more promising configurations. Since then, evaluations at multiple fidelity levels have been incorporated into many existing algorithms [154, 213], extending, for example, BO [87] or as a basis for *asynchronous methods* [155].

Neural Architecture Search

Similarly to HPO, the goal of Neural Architecture Search (NAS) is to find model architectures that yield an optimal model $\hat{f}$ after training. The search space in this case defines possible architectures for *deep learning models*. NAS methods naturally lend themselves to *multi-fidelity* evaluation, as they can be trained for a variable number of iterations. The goal of NAS similarly is to finding an architecture that minimizes some cost metric c, e.g. the validation error [81]. Denoting with $\psi \in \Psi$ an architecture from the space of available architectures and with $\boldsymbol{\lambda} \in \tilde{\Lambda}$ the space of hyperparameters (e.g. learning rate), we can define the *joint hyperparameter optimization and neural architecture search problem*:

$$\boldsymbol{\lambda}^\star, \psi^* \in \operatorname*{arg\,min}_{\boldsymbol{\lambda} \in \tilde{\Lambda}, \psi \in \Psi} c(\boldsymbol{\lambda}, \phi) \tag{2.5}$$

If gradients with respect to ψ are required, the problem is often defined as a bilevel optimization problem where the architecture ψ and the model parameters θ are optimized simultaneously [81, 158]. The search space Ψ in NAS is often complex and cannot be easily represented in a tabular format. As a result, several approaches to optimizing over such search spaces have been proposed, e.g. based on *evolutionary algorithms* [202, 80], BO [139, 250], reinforcement learning [268, 270], or making use of available gradients [158, 197]. Since full-fidelity evaluations for classical NAS scenarios are often prohibitively expensive,

[2] To the author's knowledge there is no exhaustive formal definition of what constitutes a multi-fidelity evaluation.

methods often include *multi-fidelity* evaluations in order to speed up training. Extensions of NAS towards *multi-objective* evaluations (e.g. [80]) and AutoML systems for NAS (e.g. [266]) have similarly been proposed.

2.2.2 Multi-Objective Optimization

In practical applications, we are often concerned with multiple criteria, e.g. multiple performance criteria or other aspects of the model such as robustness, interpretability, or computational efficiency when predicting with $\hat{f}$ [124]. We now introduce the generalized problem for the m criteria following [132]:
Given a number of evaluation criteria $c_1, \dots, c_m$; each assigning a value to a configuration $\boldsymbol{\lambda}$: $\Lambda \to \mathbb{R}$ with $m \in \mathbb{N}$. Furthermore, $c : \Lambda \to \mathbb{R}^m$ assigns to each configuration $\boldsymbol{\lambda}$ a cost vector of dimensions m. The goal of multi-objective optimization then is[3]

$$\min_{\boldsymbol{\lambda} \in \Lambda} c(\boldsymbol{\lambda}) = \min_{\boldsymbol{\lambda} \in \Lambda} \left(c_1(\boldsymbol{\lambda}), c_2(\boldsymbol{\lambda}), \dots, c_m(\boldsymbol{\lambda}) \right) \ . \tag{2.6}$$

Without loss of generalization, we assume that all criteria are minimized. Note that some performance criteria (e.g. *model size*) do not depend on model predictions $\hat{f}(\mathbf{x})$ but instead require properties of the fitted model. Optimization of Equation (2.6) usually yields a set of solutions. Solutions in this set are Pareto optimal and usually incur different trade-offs, where improving with respect to one criterion implies trading off or deteriorating other objectives. Therefore, the goal of multi-objective optimization methods is to approximate the theoretical Pareto front along possible trade-offs of all optimization criteria c. Since, typically, there is no total order on $\mathbb{R}^M$, and hence there usually is no single best objective value, we now consider *Pareto-dominance* and *Pareto-optimality* instead. Given a metric $c : \Lambda \to \mathbb{R}^M$, define a binary relation ' $\prec$ ' on $\mathbb{R}^M \times \mathbb{R}^M$. Given two cost vectors $\boldsymbol{\zeta}^{(1)}, \boldsymbol{\zeta}^{(2)} \in \mathbb{R}^M$ we say $\boldsymbol{\zeta}^{(1)}$ dominates $\boldsymbol{\zeta}^{(2)}$, written as $\boldsymbol{\zeta}^{(1)} \prec \boldsymbol{\zeta}^{(2)}$, if and only if

$$\forall k \in \{1, ..., M\} : \boldsymbol{\zeta}_k^{(1)} \leq \boldsymbol{\zeta}_k^{(2)} \ \wedge \ \exists k \in \{1, ..., M\} : \boldsymbol{\zeta}_k^{(1)} < \boldsymbol{\zeta}_k^{(2)}. \tag{2.7}$$

We similarly define a dominance relationship for configurations $\boldsymbol{\lambda}$: A configuration $\boldsymbol{\lambda}^{(1)}$ dominates another configuration $\boldsymbol{\lambda}^{(2)}$, so $\boldsymbol{\lambda}^{(1)} \prec \boldsymbol{\lambda}^{(2)}$. if and only if $c(\boldsymbol{\lambda}^{(1)}) \prec c(\boldsymbol{\lambda}^{(2)})$. The Pareto front $\tilde{\mathcal{P}}$ is then given by

$$\tilde{\mathcal{P}} = \{\boldsymbol{\zeta} \in c(\Lambda) \,|\, \nexists \boldsymbol{\zeta}' \in c(\Lambda) \text{ s.t. } \boldsymbol{\zeta}' \prec \boldsymbol{\zeta}\} \tag{2.8}$$

and conversely the Pareto set $\tilde{\mathcal{P}}_\Lambda$ as the pre-image of $\tilde{\mathcal{P}}$:

$$\tilde{\mathcal{P}}_\Lambda = c^{-1}(\tilde{\mathcal{P}}). \tag{2.9}$$

One important aspect of multi-objective problems is selecting an optimal model from the resulting set. Available trade-offs (i.e. the shape of the Pareto-front, cf. Figure 2.1) are

[3]min of a vector valued quantity here results in the pareto set defined in 2.9.

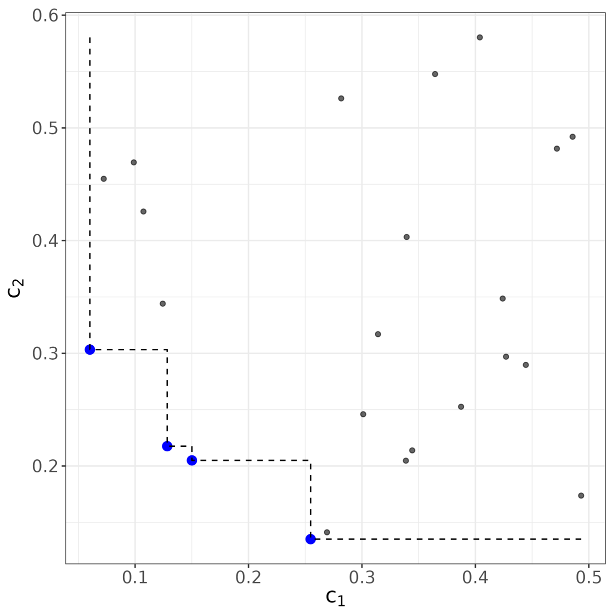


Figure 2.1: Solution set for a multi-objective optimization problem considering two objectives c_1 and c_2. Pareto-optimal points are highlighted in blue.

often unknown *a-priori* and methods should therefore yield the set of all possible trade-offs. At the same time, not all trade-offs are relevant, e.g. models below some (a priori unknown) accuracy threshold might not be interesting. This gives rise to strategies that suggest the involvement of humans *in the loop* or the imposing of additional *user preferences* [37] One assumption made in multi-objective strategies is that the objectives (or user preferences) are quantifiable. If this is not the case, user preferences can also be learned [95]. This can e.g. be done by involving the decision maker in the optimization process and asking to rank available solutions, allowing one to learn implicit user preferences from the collected rankings.
An introductory example for a Pareto-optimal set in a two-objective case is depicted in Figure 2.1. Each point in the figure denotes a configuration $\boldsymbol{\lambda}$ evaluated for the two objectives we aim to simultaneously minimize, c_1 and c_2. Within the Pareto-optimal set, improving with respect to one metric requires trading in the other metric. Depending on the nature of the involved optimization criteria, interpolation between points on the Pareto-front can yield further trade-offs, e.g. through ensembling or randomization, making the Pareto-front convex.

Several methods have been proposed to solve the optimization problem defined in Equation (2.6). These range from methods based on scalarization of the objectives [166, 125, 78] together with *Bayesian Optimization* e.g. *parEGO* [141] or *SMS-EGO* [200] to methods based on evolutionary algorithms [70, 82] along with many other approaches.
In this thesis, [187] proposes an AutoML system that allows incorporating *multiple objectives* based on *parEGO* [142]. Therefore, we briefly present its core components to facili-

tate understanding: *parEGO* is based on scalarization using the *augmented Tschebyscheff function* [142]. It weights objectives $c_i, i \in \{1, ..., m\}$ using a *weighting* parameter $\boldsymbol{\alpha} = \{\alpha_1, ..., \alpha_m\}$ sampled uniformly at random from the probability simplex, subject to $\sum_i \alpha_i = 1$:

$$c_\alpha(\boldsymbol{\lambda}) = \max_{i \in \{1,...,m\}} (\alpha_i c_i(\boldsymbol{\lambda})) + \rho \cdot \sum_{i=1}^{m} \alpha_i \cdot c_i(\boldsymbol{\lambda}). \tag{2.10}$$

In each iteration, new scalarization weights α are sampled and the next evaluation candidate is chosen using the expected improvement [141] on a surrogate model. The parameter ρ is a small constant that ensures that solutions are also found in non-convex regions of the Pareto front.

2.2.3 Hyperparameter Defaults

Although hyperparameter optimization can lead to better solutions [150], it is often expensive, since multiple evaluations of the objective function are required. *Default values* present a solution to this problem that require no evaluation of an algorithm during optimization. For this reason, hyperparameter defaults are also often called *zero-shot* HPO. In comparison to tuning, defaults have several benefits: They require fitting the objective function only once, are simple to implement and do not require any experimental design considerations or considerations with respect to which parameter should be tuned [201]. As a result, defaults are often used in practice. Considerable effort has been spent on obtaining good defaults and they are made available in different software implementations [184, 42, 50]. Although default configurations are not likely to surpass complex tuning methods across a large variety of datasets, they can nevertheless provide strong baselines to compare against.

We consider obtaining defaults as a *meta-learning* [38, 244] task in this thesis. The goal in this context is to infer configurations that are expected to perform well on future datasets. We formalize this, denoting with $\mathbb{D}$ the distribution of all datasets $\mathcal{D}$ and with $GE_\mathcal{D}(\lambda)$ the *generalization error* of a configuration $\boldsymbol{\lambda}$ on a dataset $\mathcal{D}$

$$\lambda^\star_{def} \in \arg\min_{\boldsymbol{\lambda} \in \Lambda} \; \mathbb{E}_{\mathcal{D} \sim \mathbb{D}} \left[GE_\mathcal{D}(\boldsymbol{\lambda}) \right]. \tag{2.11}$$

We can solve the optimization problem in Equation (2.11) by relying on the empirical formulation based on estimated generalization errors across T datasets $\{\mathcal{D}_1, ..., \mathcal{D}_T\}$ and an aggregation function h, such as the *mean*:

$$\hat{\lambda_{def}} \in \arg\min_{\boldsymbol{\lambda} \in \Lambda} \; h\left(\widehat{GE_{\mathcal{D}_1}(\boldsymbol{\lambda})}, ..., \widehat{GE_{\mathcal{D}_T}(\boldsymbol{\lambda})} \right). \tag{2.12}$$

Obtaining λ_{def} in practice requires evaluating a large number of configurations λ across a large quantity of datasets. This computational cost can be reduced if collections of metadata on the relationship between hyperparameters $\boldsymbol{\lambda}$ and a performance metric of interest (e.g. accuracy) collected across a large and diverse set of datasets are available. This requires that the same $\boldsymbol{\lambda} \in \tilde{\Lambda}$ are evaluated on each dataset. If such meta-data is not available, evaluations of $\widehat{GE(\boldsymbol{\lambda})}$ can be substituted with evaluations of surrogate model that approximates the relationship between λ and $\widehat{GE}$ on each dataset [195]. Different aggregation functions h, such as more robust estimators or ranking [39] have been studied in literature [194, 255].

Symbolic Defaults For many algorithms, the optimal configuration $\boldsymbol{\lambda}$ might depend on characteristics of the dataset, e.g. the number of features or the number of datapoints. We call such defaults *symbolic defaults* as they rely on symbolic expressions that, when evaluated using dataset criteria, yield a configuration. A popular example for such expressions is $mtry = \sqrt{p}$, that is, setting the number of variables considered in each split to the square root of the number of available features in the random forest algorithm [42]. To integrate formulas using dataset characteristics into Equation (2.11) we define a space of functions $\mathcal{F}$. Those functions take as input meta-features obtained from a dataset $\mathcal{D}$ as input and return a configuration λ. Equipped with such a function $g \in \mathcal{F} : \mathcal{D} \to \Lambda$, we can state the optimization problem

$$g_{def^\star} := \underset{g \in \mathcal{F}}{\arg\min} \;\; \mathbb{E}_{\mathcal{D} \sim \mathbb{D}} \left[GE_{\mathcal{D}}(g(\mathcal{D})) \right]. \tag{2.13}$$

Optimally, solutions g_{def} to this problem should be simple to make them easy to implement and understand, but also to prevent overfitting to the often relatively limited amount of data that is available for such scenarios. Solutions to this optimization problem can, for example, be obtained from Genetic Programming (cf. Section 2.2.1).

Multiple Defaults Given that a single default might not be optimal, the optimization problem can be extended to return a set of multiple defaults. Formally, we now want to obtain a set of defaults Λ_{def} consisting of K configurations that are jointly optimal. That is, for each dataset $\mathcal{D} \sim \mathbb{D}$ we would like that there exists one $\lambda \in \Lambda_{def}$ that yields good performance. This can be formalized as follows:

$$\Lambda_{def} := \underset{\{\boldsymbol{\lambda}_1, \ldots, \boldsymbol{\lambda}_K\};\, \boldsymbol{\lambda}_i \in \Lambda}{\arg\min} \;\; \mathbb{E}_{\mathcal{D} \sim \mathbb{D}} \left[\min_{k \in \{1, \ldots, K\}} GE_{\mathcal{D}}(\boldsymbol{\lambda}_k) \right]. \tag{2.14}$$

Using a set of pre-computed defaults now requires evaluating all defaults in the set as well as obtaining unbiased estimates of $\widehat{GE}(\boldsymbol{\lambda})$, e.g. through cross-validation. On the other hand, multiple defaults contain information about the performance of configurations across data sets and can therefore be competitive with more involved tuning methods that do not incorporate this knowledge [195]. The concept of a single (static) default for HPO was previously discussed e.g. in [201]. Several publications discuss (often symbolic) defaults along with new algorithms or software implementations, e.g. using the inverse median for the inverse kernel width γ of a radial basis function SVM [50]. The concept of obtaining multiple defaults was first introduced in [253] in the context of obtaining good initializations for *Bayesian Optimization*. It has subsequently been extended to work in an online-fashion making use of multi-fidelity evaluations [252] to significantly reduce computational cost of obtaining such sets. Static defaults are also used in practice; `AutoGluon` [83] uses a fixed set of hyperparameter configurations instead of employing any optimization.

2.2.4 Algorithm Configuration

With the introduction of a large variety of HPO methods and the multiplicity of available design choices when deciding which optimization method to use and how to select hyperparameters of the optimization method, a natural next step is *algorithm configuration*, e.g. used when the goal is to *optimize the optimizer*.
We now go on to formalize the goal of algorithm configuration for a general optimization algorithm $\mathbf{O}$ controlled by configuration parameters $\boldsymbol{\gamma} \in \Gamma$ following [171]: The algorithm $\mathbf{O} : \Omega \times \Gamma \rightarrow Z$ takes as input a problem instance $\omega \in \Omega$ drawn from a distribution over problem instances $\mathbb{P}_\Omega$ and algorithm control parameters $\boldsymbol{\gamma}$ to produce an algorithm z configured with $\boldsymbol{\gamma}$. Given a cost metric $\nu : Z \rightarrow \mathbb{R}$, that takes the configured algorithm, our goal is now to find a configuration $\boldsymbol{\gamma}^*$ that optimizes the expected cost across instances drawn from $\mathbb{P}_\Omega$.

$$\boldsymbol{\gamma}^* \in \underset{\boldsymbol{\gamma} \in \Gamma}{\arg\min} \;\; \mathbb{E}_{\omega \sim \mathbb{P}_\Omega} \left[\nu(\mathbf{O}(\omega, \boldsymbol{\gamma})) \right] \tag{2.15}$$

The result of running algorithm configuration is an optimizer $\mathbf{O}$. Selected control parameters γ for $\mathbf{O}$ can refer to the particular options for the choice in optimization strategy, but also individual hyperparameters of the given methods, e.g. the choice of κ when using the LCB criterion from Equation (2.4). Having obtained a set of representative instances, the algorithm configuration now proposes the configuration $\boldsymbol{\gamma}^*$ that minimizes the aggregate cost across all instances.

It is now interesting to view *hyperparameter defaults* introduced in Section 2.2.3 as a form of algorithm configuration (cf. e.g. [156]). We are interested in optimizing the algorithm $\mathbf{O}$ across a diverse set of instances, $\omega \in \Omega$, which in this case are datasets $\mathcal{D} \sim \mathbb{D}$. In the simplest case, our algorithm $\mathbf{O}$ is configured using a single hyperparameter configuration λ. Setting $\Gamma = \Lambda$, choosing an appropriate cost function ν, we can essentially arrive at the formulation of a default hyperparameter defined in Equation (2.11). After configuration, we arrive at an optimal configuration $\boldsymbol{\gamma}^* = \boldsymbol{\lambda}_{def}$. This naturally extends to the notions of *symbolic* and *multiple* hyperparameter defaults by adapting algorithm and search space.

2.2.5 AutoML

The field of AutoML now has the goal to optimize the process of obtaining optimal models in multiple directions by *i*) abstracting away experimental design considerations regarding algorithm and hyperparameter selection and solving the *Combined Algorithm and Hyperparameter Selection (CASH) Problem* [233] automatically and *ii*) increasing the efficiency of obtaining such solutions through the use of efficient HPO methods, meta-learning and parallelization. This results in AutoML systems that only expect a dataset, the performance metric and optionally the computational budget as input and return one or multiple trained models as output. Internally, the AutoML system should then determine the optimal pre- and post-processing steps, as well as the optimal ML algorithm and hyperparameters. This can be summarized as *Machine Learning Pipeline Configuration* [230].

2.2.6 Machine Learning Pipelines

Figure 1.1 in the Introduction depicts a typical machine learning pipeline for *tabular* data. While different AutoML systems employ different strategies towards optimizing over the search space arising from the directed acyclic graph (DAG), the core component of AutoML systems are machine learning pipelines. The following section mainly focuses on *typical* AutoML systems for tabular data [233, 92, 91, 122]. Depending on the application domain, differences between systems can exist, e.g. typical automatic NAS systems often implicitly include pre-processing in the model architecture and optimization routine (e.g. in the form of feature extraction through learned CNN filters). We therefore omit a formal definition of an AutoML system, as it would likely only cover a small portion of existing and future systems and instead go on to introduce relevant concepts and components.

The following section focuses mainly on machine learning pipelines typically used in AutoML systems for tabular data [233, 92, 91, 122]. We define a machine learning pipeline as a sequence of processing steps that sequentially transform its input and produce a prediction as its output. This process is illustrated in Figure 2.2.

Notation We start by defining a general processing step or pipeline operator $\mathfrak{P}$: Pipelines are generally employed in two distinct phases: A *train* phase with the goal of learning the processing steps required to transform incoming data and learn a model, and a *predict* phase, in which predictions for new data points are obtained, each requiring distinct processing steps. ML pipelines are composed of individual

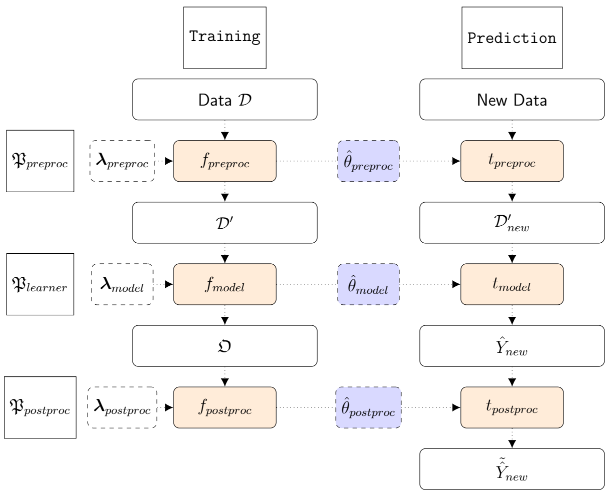


Figure 2.2: Schematic of a machine learning pipeline. We differentiate between *training* and *prediction* mode. During training, processing steps g before being transformed into predictions by a ML model.

processing steps $\mathfrak{P}$. In the following,
we will define processing steps and their composition into general ML pipelines.

A general processing step consists of a tuple $(\mathfrak{f}_{\mathfrak{P}}, \mathfrak{t}_{\mathfrak{P}})$ that defines how inputs are processed during the *train* and *predict* phase respectively. In general, we assume inputs during training ($v_{\mathfrak{P}} \in \mathcal{V}_{\mathfrak{P}}$) and prediction ($w_{\mathfrak{P}} \in \mathcal{W}_{\mathfrak{P}}$) differ. The corresponding outputs of the processing steps are primed to indicate that they might have changed structurally.

Train phase $\mathfrak{f}_{\mathfrak{P}}$: The *train* function $\mathfrak{f}_{\mathfrak{P}} : \mathcal{V}_{\mathfrak{P}} \times \Lambda_{\mathfrak{P}} \to \mathcal{V}'_{\mathfrak{P}} \times \Theta_{\mathfrak{P}}$ takes an input $v_{\mathfrak{P}}$ and hyperparameters $\lambda_{\mathfrak{P}}$ and returns transformed inputs $v'_{\mathfrak{P}}$ together with the trained model $\hat{\theta}_{\mathfrak{P}}$. In general, we differentiate between two components in the training phase: ***state*** and ***process***. In the ***state*** step, the operation generates the model $\hat{\theta}_{\mathfrak{P}} : \mathcal{W}_{\mathfrak{P}} \to \mathcal{W}'_{\mathfrak{P}}$ from $v_{\mathfrak{P}}$ controlled by hyperparameters $\boldsymbol{\lambda}_{\mathfrak{P}} \in \Lambda_{\mathfrak{P}}$ with the goal to transform new, future data points $w_{\mathfrak{P}}$ in the prediction phase. In the ***process*** step, the goal is to transform the input data that is returned during the train step. In some cases, this transformation is given, for example, by applying the learned model $\hat{\theta}_{\mathfrak{P}}$ to the inputs $v_{\mathfrak{P}}$.

Predict phase $\mathfrak{t}_{\mathfrak{P}}$: The *predict* function $\mathfrak{t}_{\mathfrak{P}} : \mathcal{W}_{\mathfrak{P}} \times \Theta_{\mathfrak{P}} \times \Lambda_{\mathfrak{P}} \to \mathcal{W}'_{\mathfrak{P}}$ employs the model $\hat{\theta}_{\mathfrak{P}}$ learned during the training phase to compute the output: $w'_{\mathfrak{P}} = \hat{\theta}_{\mathfrak{P}}(w_{\mathfrak{P}})$. The prediction step can similarly be controlled by hyperparameters $\boldsymbol{\lambda}_{\mathfrak{P}}$.

Operators Different operators are often composed together into ML pipelines. We can now define *pre-processing* steps as well as *learner* and *post-processing* steps that further specialize a general processing step $\mathfrak{P}$. We briefly introduce generic typical steps for a *supervised* setting. Adoption to *unsupervised* settings is straightforward. In this setting, we assume that the input features come from a dataset $D \in \mathcal{D}$, that is comprised of features from a feature space $\mathcal{X}$ and corresponding labels from a space of labels $\mathcal{Y}$.

- A *pre-processing* operator $\mathfrak{P}_{preproc}$ processes features $\mathcal{X}$ and labels $\mathcal{Y}$.

$$\begin{aligned} \mathfrak{f}_{preproc} &: \ (\mathcal{X} \times \mathcal{Y})^n \times \Lambda_{preproc} \to (\mathcal{X}' \times \mathcal{Y}')^{n'} \times \Theta_{preproc} \\ \mathfrak{t}_{preproc} &: \ (\mathcal{X})^{\tilde{n}} \times \Theta_{preproc} \times \Lambda_{preproc} \to (\mathcal{X}')^{\tilde{n}} \end{aligned}$$

  During training, $\mathfrak{f}_{preproc}$ learns a model $\hat{\theta}_{preproc}$ while transforming the input data and returns transformed features $\mathcal{X}'$ along with a learned transformation operation $\hat{\theta}_{preproc}$. The number of input samples n can differ between in- and output during training. We denote the number of outputs during training with n'. During prediction $\mathfrak{t}_{preproc}$, the operator transforms the input using the learned model and returns the transformed features. In general, we assume that the number of samples in in- and outputs $\tilde{n}$ in the prediction phase is constant.

  Preprocessing operators usually transform features $\mathcal{X}$ (feature transforms, e.g., missing data imputation), targets $\mathcal{Y}$ (target transforms, e.g. log-scaling), or both, for example, by over- or undersampling observations, or by introducing synthetic observations. The goal of *pre-processing* is not only to obtain different transformations of the data that allow for better modeling, but also to remedy potential problems that can arise from the data. As an example, ML algorithms often cannot naturally handle missing values in the data, and pre-processing therefore needs to impute such missing data while preserving all relevant information. Or, if data in a feature are categorical, a numeric representation might be required, e.g. in the form of embeddings [108] or other categorical encoding schemes [180] for different algorithms to work. As a result, some decisions regarding pre-processing depend on the learner selected in the subsequent modeling step. In some cases, pre-processing also makes use of or transforms the labels fed to the subsequent inducer, e.g. when encoding categorical variables [180] or when applying log-scaling to regression outcomes.

- A *learner* operator $\mathfrak{P}_{learner}$ learns a prediction model from features $\mathcal{X}$ and labels $\mathcal{Y}$.

$$\mathfrak{f}_{model}: \ (\mathcal{X} \times \mathcal{Y})^n \times \Lambda_{model} \rightarrow \mathfrak{O} \times \Theta_{model}$$
$$\mathfrak{t}_{model}: \ (\mathcal{X})^{\tilde{n}} \times \Theta_{model} \times \Lambda_{model} \rightarrow (\mathcal{Y})^{\tilde{n}}$$

It takes as input feature, label pairs $(\mathcal{X}, \mathcal{Y})^n$ during training and produces a machine learning model $\hat{\theta}_{model}$. The goal of $\hat{\theta}_{model}$ is to predict the label $\hat{\mathcal{Y}}$ for new data points $\mathcal{X}$ during prediction. We denote with $\mathfrak{O}$ any other outputs a model might yield during training. During prediction, the operator simply returns the learned model's prediction; $\mathfrak{t}_{model} : (\mathcal{X})^{\tilde{n}} \rightarrow (\hat{\mathcal{Y}})^{\tilde{n}}$. The learner operators can wrap different machine learning models controlled by a respective configuration $\boldsymbol{\lambda}_{learner}$. Typical choices for ML algorithms are decision trees [43], random forests [41] or xgboost models [54] or support vector machines (SVM) [217], but many more are used in typical AutoML systems.

- A *post-processing* operator $\mathfrak{P}_{postproc}$ takes as input a predicted label $(\hat{\mathcal{Y}})^n$ and learns a transformation model $\hat{\theta}_{postproc}$. During prediction, the predict function $\mathfrak{f}_{postproc}$: $(\hat{\mathcal{Y}})^n \rightarrow (\hat{\mathcal{Y}}')^n$ transforms predictions according to the learned model. Again, we denote with $\mathfrak{O}$ any other inputs and outputs that might be required for a postprocessing step.

$$\mathfrak{f}_{postproc}: \ \mathfrak{O} \times \Lambda_{postproc} \rightarrow \mathfrak{O} \times \Theta_{postproc}$$
$$\mathfrak{t}_{postproc}: \ (\mathcal{Y})^{\tilde{n}} \times \Theta_{postproc} \times \Lambda_{postproc} \rightarrow (\mathcal{Y}')^{\tilde{n}}$$

Typical post-processing steps include *thresholding*, i.e. applying cut-offs that transforms classification scores to class predictions, *prediction transforms*, e.g. exponentiation of predictions learned on a log-transformed target or *calibration* [176, 113] of predicted class probabilities. Post-processing is often done on a separate training set (or on cross-validated predictions [254]) to avoid further overfitting to the training data.

In some cases, operators do not infer any quantities from the data, and instead simply perform a static transformation during training and prediction. Examples for this, e.g. include log-transforming regression labels before prediction and exponentiating predictions.

Composing Operators We now define an operation $\mathfrak{P}^{(2)} \circ \mathfrak{P}^{(1)}$ that defines the (sequential) composition of two operators $\mathfrak{P}^{(1)}$ and $\mathfrak{P}^{(2)}$ into a new operator $\mathfrak{P}^{comp} = (\mathfrak{f}_{comp}, \mathfrak{t}_{comp})$, such that ${\mathcal{V}'_{\mathfrak{P}}}^{(1)} = {\mathcal{V}_{\mathfrak{P}}}^{(2)}$ and ${\mathcal{W}'_{\mathfrak{P}}}^{(1)} = {\mathcal{W}_{\mathfrak{P}}}^{(2)}$. During training, inputs are transformed sequentially and resulting models are returned together.

$$
\begin{aligned}
\mathfrak{f}_{comp} &: {\mathcal{V}_{\mathfrak{P}}}^{(1)} \times \overbrace{\Lambda_{\mathfrak{P}^{comp}}}^{\Lambda_{\mathfrak{P}^{(1)}} \times \Lambda_{\mathfrak{P}^{(2)}}} \to {\mathcal{V}'_{\mathfrak{P}}}^{(2)} \times \overbrace{\Theta_{\mathfrak{P}^{comp}}}^{\Theta_{\mathfrak{P}^{(1)}} \times \Theta_{\mathfrak{P}^{(2)}}} \\
\mathfrak{t}_{comp} &: {\mathcal{W}_{\mathfrak{P}}}^{(1)} \times \Theta_{\mathfrak{P}^{comp}} \times \Lambda_{\mathfrak{P}^{comp}} \to {\mathcal{W}'_{\mathfrak{P}}}^{(2)}
\end{aligned}
$$

Similarly, during prediction, inputs are also transformed sequentially given previously learned models.

$$
\begin{aligned}
\mathfrak{f}_{comp}\,({v_{\mathfrak{P}}}^{(1)}, \overbrace{\lambda_{\mathfrak{P}^{comp}}}^{\lambda_{\mathfrak{P}^{(1)}} \times \lambda_{\mathfrak{P}^{(2)}}}) &= {v'_{\mathfrak{P}}}^{(2)}, \overbrace{\hat{\theta}_{\mathfrak{P}^{comp}}}^{\hat{\theta}_{\mathfrak{P}^{(1)}} \times \hat{\theta}_{\mathfrak{P}^{(2)}}} \\
{v'_{\mathfrak{P}}}^{(1)}, \hat{\theta}_{\mathfrak{P}^{(1)}} &= \mathfrak{f}^{(1)}_{\mathfrak{P}}({v_{\mathfrak{P}}}^{(1)}, \lambda_{\mathfrak{P}^{(1)}}) \\
{v'_{\mathfrak{P}}}^{(2)}, \hat{\theta}_{\mathfrak{P}^{(2)}} &= \mathfrak{f}^{(2)}_{\mathfrak{P}}({v'_{\mathfrak{P}}}^{(1)}, \lambda_{\mathfrak{P}^{(1)}})
\end{aligned}
$$

$$
\begin{aligned}
\mathfrak{t}_{comp}\,({w_{\mathfrak{P}}}^{(1)}, \hat{\theta}_{\mathfrak{P}^{comp}}, \lambda_{\mathfrak{P}^{comp}}) &= {w'_{\mathfrak{P}}}^{(2)} \\
{w'_{\mathfrak{P}}}^{(1)} &= \mathfrak{t}^{(1)}_{\mathfrak{P}}({w_{\mathfrak{P}}}^{(1)}, \hat{\theta}_{\mathfrak{P}^{(1)}}, \lambda_{\mathfrak{P}^{(1)}}) \\
{w'_{\mathfrak{P}}}^{(2)} &= \mathfrak{t}^{(2)}_{\mathfrak{P}}({w'_{\mathfrak{P}}}^{(1)}, \hat{\theta}_{\mathfrak{P}^{(2)}}, \lambda_{\mathfrak{P}^{(2)}})
\end{aligned}
$$

The hyperparameter space of the composed operator is then given by the concatenation of the individual hyperparameter spaces

$$\Lambda_{comp} = \Lambda_{\mathfrak{P}^{(1)}} \times \Lambda_{\mathfrak{P}^{(2)}}.$$

Machine Learning pipelines By composing operators, we can build new, more complex learning algorithms. This gives rise to *ML pipelines*, that are compositions of individual operators which mimic a learning algorithm. *First*, composition of different pre-processing operators again results in a pre-processing operator. This allows composing several, often needed operators such as scaling, missing value imputation or feature transformations within a single pipeline. *Second*, composing different post-processing operators again results in a post-processing operator. This allows for sequentially applying widely used post-processing steps such as target transformations or calibration. *Third*, concatenating

any pre-processing operator with a model operator results essentially mimics a learner operator. This means, that input and output types of the composed operators mimic input and output types of a simple learner operator. The same holds, when learner operators are connected with post-processing operators.
In summary, this allows us to express ML pipelines from a set of matching preprocessing operators, a learner operator and a set of postprocessing operators as a sequential composition $\mathfrak{P}^{(k)} \circ \ldots \circ \mathfrak{P}^{(1)}$. The composition than inherits in- and outputs from the first and last operator respectively. The hyperparameter space is similarly given by the concatenation of individual hyperparameter spaces $\Lambda_{\mathfrak{P}^{(1)}} \times \ldots \times \Lambda_{\mathfrak{P}^{(k)}}$.

Limitations Sections above describe general linear pipeline operators that can be composed into larger, more complex ML pipelines. However, the aforementioned categorization into *preprocessing*, *model* and *post-processing* is not exhaustive, as not all processing steps fit perfectly into the categorization. To provide an example, *post-hoc calibration* requires an additional input (a calibration set). Furthermore, the aforementioned composition strategy can only express a subset of pipelines that are typically used in ML contexts. To provide an example, linear pipelines can not easily express parallel processing steps, e.g. based on ensembles, including common strategies such as bagging and stacking. Similarly, we might want to express a choice over different pre-processing steps inside the pipeline, which could, e.g. be expressed via *meta-* operators that act as different operators depending on their hyperparameter configurations. A different, more flexible approach therefore is to express ML pipelines as a *directed acyclic graph* (DAG). This approach is used, e.g. in mlr3pipelines [26]. And has been described in similar forms the context of scikit-learn pipelines [46] and model composition for MLJ [33].

Further Components While the core component of AutoML systems is optimization over a space of available ML pipelines, there are many other aspects that differentiate individual systems. The `auto-sklearn` system e.g. warm starts optimization from a meta-learned portfolio of pipelines similar to *multiple defaults* introduced above. It furthermore aggregates a selection of models fitted during the optimization procedure into an ensemble [51] to arrive at better and more robust predictors. `AutoGluon` [83] provides an additional distillation step, reducing the ensemble into a single model [86]. Further adaptations in many AutoML frameworks often introduce efficient parallelization of the computational workloads or make systems more robust to different datasets often encountered in the practice.

2.2.7 Contributions

Several contributions in this thesis pursue active research directions in the broader field of AutoML.
In the context of HPO defaults, two contributions, [242] in Section 4.1 and [101] in Section 4.2 propose approaches to obtaining *symbolic* defaults. Those contributions introduce

a principled way to obtain *data-dependent* defaults. At the same time, resulting formulas are trivial to implement in existing ML frameworks – required meta-features can be computed cheaply from the data at train time. Several problems in this context provide interesting avenues of future research: The space of possible formulas that can be used to define a symbolic default is enormous, while the number of available data sets for configuration is comparably small. Finding formulas that generalize to future datasets, therefore, requires good inductive biases and strong regularization. In addition, it is unclear which meta-features to use, as readily available meta-features as proposed by [241] do not necessarily contain enough information required to compute good hyperparameters.
In contrast, another contribution [195] provided in Section 4.3 proposes an approach for learning multiple defaults. From the perspective of providing a simple tuning method – simply providing a list of configuration candidates – the method introduces a greedy procedure that iteratively adds a new default configuration to a set of default values. It demonstrates that such lists can be competitive with hyperparameter tuning approaches in the domain of few function evaluations, and a list of, e.g. 8 defaults compares favorably with random search using up to quadrupled budget. If higher budgets are available, the list of defaults can also be used as an initialization strategy e.g. for *Bayesian Optimization* as proposed previously in [253]. One problem when using such lists is that they are optimized for a single performance metric; if a different metric should be considered, a new set of default configurations has to be learned.

Learning defaults relies on evaluating configurations $\boldsymbol{\lambda}$ across a large collection of possible datasets with members $\mathcal{D} \sim \mathbb{D}$. This can be computationally prohibitively expensive and therefore methods often rely on a table of precomputed performances or model the relationship between $\boldsymbol{\lambda}$ and $c(\boldsymbol{\lambda})$ using a surrogate model. A contribution in this thesis, [25] presented in Section 4.4, provides a large collection of such meta-data evaluating 7 algorithms across up to 119 datasets reporting several performance metrics along with additional information such as runtime and memory usage. We leverage this data along with other data sources to construct a *multi-fidelity, multi-objective optimization* benchmark suite. This contribution [192] is presented in Section 4.7. The suite enables benchmarking of single- and multiobjective HPO methods on a large variety of representative benchmark instances stemming from the broader context of HPO and NAS. It leverages efficient surrogate models to allow for evaluation of continuous search spaces $\tilde{\Lambda}$ in contrast to only allowing for evaluation on the discretized search space induced by tabular representations.

In the context of this thesis, one contribution, [187] provided in Section 4.5 proposes a multi-objective AutoML system that allows for simultaneously optimizing multiple criteria $c_1, ..., c_m$. Optimization criteria can be performance criteria, but similarly also criteria that measure *fairness, interpretability* or *robustness* of resulting learners along with computational criteria (e.g. latency or required memory). The system employs the ParEgo algorithm for multiobjective *Bayesian Optimization* introduced in Section 2.2.2 and optimizes only a single learner, `xgboost` [54].
We furthermore introduce a software, `mlr3pipelines` that provides a powerful API for

specifying complex pipelines using simple building blocks. The corresponding contribution [26] is provided in Section 6.4. Each building block (or processing step g comes with a hyperparameter space Λ_g. Powerful *meta-operators*, that e.g. allow selecting between different processing options or processing data in parallel allow specifying simple AutoML systems tailored to the problem at hand using only a few lines of code.

AutoML systems internally rely on efficient HPO methods in order to arrive at good models. Finding good HPO optimizers is essentially an *Algorithm Configuration* problem: Find an optimizer $\mathbf{O}$, that in expectation yields low cost $\zeta(\mathbf{O})$ on a set of problem instances of interest. Although the search space Γ is unknown during the development of new algorithms, we can synthesize new HPO methods similarly from components proven successful in other HPO methods. This can not only result in even better optimizers but simultaneously help to better understand design decisions made in previously proposed methods. This thesis presents one attempt to synthesize a novel *multi-fidelity* HPO optimizer [171] presented in Section 4.6.

2.3 Fairness

Decisions based on predictions made by machine learning systems are abound in practice, not only in industry [96] but increasingly so by governmental actors [8]. To provide an example, in the context of lending, banks collect information about loan applicants such as employment and loan history along with other demographic variables and use them to predict a risk score for a given individual using an ML model. This score is then used by loan officers to arrive at a decision regarding whether a loan should be granted and under what terms [168]. The goal of algorithmic fairness now is to *detect* and possibly *alleviate* ethical issues arising from the use of ML systems as a basis for decision making [13, 227]. Decisions made in such contexts are often assumed to be binary in the fairness literature [168], the loan application is either accepted or rejected.

The credit scoring model used in this context is trained on data from previous loan applicants and the outcome label of interest – whether the applicant paid back the loan. This immediately raises two questions regarding the fairness of models used in this manner: 1) If historical structures are unfair toward individuals, the model might perpetuate this unfairness, 2) if individuals with certain characteristics are denied loans, the decision maker will never obtain ground truth regarding the individual's credit worthiness, and 3) if data indicates, that any subgroup may be less likely to pay back a loan[4], ethical and legal considerations might prohibit the model from acting on such knowledge. Discrepancies such as the ones above are often informally called *bias* [168] in the fair ML literature. Figure 1.2 tries to provide a conceptual overview over such *biases*, which could manifest at any stage of ML model development – already be present in the data, occur during data collection or as a result of miss-specified models.

Given a trained model, we are now interested in determining whether the model is *fair*. Defining what constitutes fair, and subsequently formalizing such notions of fairness is difficult, as ethical and legal considerations of fairness often depend on the context. As a result, different fairness notions have been proposed that try to translate ethical and legal criteria into mathematical fairness definitions [165, 168].

We consider the following scenarios in an attempt to formalize fairness considerations in the context of automated decision making, assuming a binary decision scenario. As a first example, we consider decision over loan applications. The decision space includes the option to decline ($d^{(i)} = 0$) or to accept the loan application ($d^{(i)} = 1$) for an individual i, $\mathcal{Y}^\star = \{0, 1\}$ [168]. The space of corresponding labels $\mathcal{Y}$, in contrast, consists of information regarding whether an individual paid back a previously granted loan ($y^{(i)} = 1$) or defaulted on the loan ($y^{(i)} = 0$). If now $y^{(i)} = d^{(i)}$, we retrospectively consider the decision to be correct and incorrect otherwise. A trained ML model $\hat{f}$ now predicts a real valued score $s^{(i)} = \hat{f}(\mathbf{x}^{(i)})$ that is translated to a decision $d^{(i)} \in \mathcal{Y}^\star$ using a decision function $\delta : \mathbb{R} \to \mathcal{Y}^\star$ (e.g. simple thresholding).

Several fairness criteria consider fairness with respect to demographic subgroups that are

[4]This should not be taken to indicate, that some groups are less trustworthy then others, but that data might show such patterns, e.g. due to systemic discrimination or a variety of other reasons.

legally protected, so called *protected* or *sensitive* groups. In the US, for example it is illegal to treat individuals differently based on their membership in such groups, e.g. *race, gender, religion or sexual orientation* (see, e.g. [53, 62]). As a second example, we consider pre-trial risk assessment. The decision space $\mathcal{Y}^{\star}$ is given by the option to release a defendant prior to the trial ($d^{(i)} = 1$) or to detain the defendant until trial ($d^{(i)} = 0$). The space of possible outcomes encompasses the defendant re-offending ($y^{(i)} = 0$) or not re-offending ($y^{(i)} = 1$). It is important to note, that in both scenarios, that observing the true outcome $y^{(i)}$ is conditioned on the decision $d^{(i)} = 1$.

The features $\mathbf{x}^{(i)}$ in the data $\mathcal{D}$ may include one or more protected attributes $a^{(i)} \subset \mathbf{x}^{(i)}$. We furthermore define with $\mathcal{A}$ the space of all possible values of a. For clarity, we will explicitly denote with A, D, X and Y the random variables from which we draw realizations a, d, $\mathbf{x}$ and y respectively. The decision $D = \delta(S)$ is derived from a score $S = \hat{f}(X)$.

2.3.1 Notions of Fairness

Equipped with the required notation, we are now ready to introduce different *notions* of fairness that can subsequently be translated into fairness metrics. There are three primary flavors of fairness metrics: *individual* fairness metrics, which demand, informally stated, that similar individuals are treated similarly [74]. To do so, they require a metric that allows one to measure distances between individuals. For this reason, those fairness notions are also called metric fairness [168]. Metrics can then be used to determine similar observations and subsequently to compare whether those similar people have similar outcomes. However, in practice, how such metrics should be constructed is not immediately clear. *Causal* notions integrate the causal structure of the data generating process, and define e.g. counterfactual notions of fairness [147, 263]. Those notions answer questions like would the decision have been different, had the individual been from a different protected group by observing differences in prediction between original and counterfactual instances. To provide an example, one could consider a male individual with associated features $\mathbf{x}$ and ask how those features would look like had the individual been born female, arriving at a set of different features $\mathbf{x}^{\star}$. Differences in predictions between those two instances can now be first steps towards detecting potential societal biases. Counterfactual methods require access to the underlying data generating process in order to e.g. compute counterfactual quantities. Both, individual and causal notions define fairness on the individual level and, therefore, allow one to assess the fairness of individual decisions.

Contributions in this thesis mainly focus on *statistical group fairness* definitions. In contrast to individual and causal definitions, these definitions do not define unfairness on the level of an individual decision, but instead on the level of averages over subgroups identified by protected attributes. At the same time, group fairness definitions operate on an observational level, that is, they can be computed from the data alone [168]. Group fairness definitions broadly require that either predictions or errors derived from a model $\hat{f}$ be independent of the protected group A. In the following, important statistical fairness definition will be presented, which are later summarized in Section 2.3.1.

Equal Error based fairness definitions informally require that errors made by a decision system are independent of the protected group. This results in several notions, by either conditioning on the outcome, $D \perp A \,|\, Y = y$, and that therefore requiring, that"... *people with the same outcome are treated the same, regardless of sensitive group membership"* [168]. Similarly, notions can be defined by conditioning on the decision $Y \perp A \,|\, D = d$. This allows the definition of multiple fairness criteria based on the confusion matrix and its extension, the *fairness tensor* [135]. A fairness tensor is constructed from the stacked confusion matrices for each protected group in A. An example for such a fairness tensor is provided in Table 2.1

	$Y = 1$	$Y = 0$	$\sum$
$D = 1$	TP True Positive	FP False Positive	PP Predicted Positive
$D = 0$	FN False Negative	TN True Negative	PN Predicted Negative
$\sum$	P Positives	N Negatives	

$$\begin{array}{c|cc} A=1 & & \\ \hline & TP_1 & FP_1 \\ & FN_1 & TN_1 \end{array} \qquad \begin{array}{c|cc} A=0 & & \\ \hline & TP_0 & FP_0 \\ & FN_0 & TN_0 \end{array}$$

Table 2.1: *Upper:* Confusion matrix for binary decisions D and true outcomes Y.
Lower: Fairness-confusion tensor [135] for binary decisions D and outcomes Y for groups $A = 0$ and $A = 1$. In practice, the fairness tensor is a 3-dimensional tensor containing the stacked confusion matrices for all groups in A.

In the lending example from above, we might define fairness to be equality in true positive rates, i.e. we require that the chance to accept loan applications for individuals who would pay it back is independent of the protected group: $P(D = 1|Y = 1, A = 1) = P(D = 1|Y = 1, A = 0)$ or expressed using entries from the fairness tensor: $TP_0/P_0 = TP_1/P_1$. This notion corresponds to $D \perp A \,|\, Y = 1$ and is called *equal opportunity* [112]. It is often used in context where a positive outcome (e.g. college admission, granted loans) has to be fairly distributed [210]. Note that the equality in true positive rates equates to equality in false negative rates due to symmetry $TPR = 1 - FNR$. Similar notions exist that demand equality in the symmetric true negative and false positive rates [57]. In contrast to equal opportunity, fairness notions based on false positive rates are used in the context of punitive interventions: Considering, e.g. the case of assessing recidivism risk, we could demand that all individuals who are going to reoffend have the same likelihood of receiving the favorable outcome (e.g. early release) independent of their protected status [174, 168].

Fairness definitions based on TPR and FPR implicitly condition on the outcome Y. Similarly, a set of fairness definitions can be defined by conditioning on the decision D, and therefore demanding $Y \perp A \,|\, D = d$. This naturally leads to consideration of the symmetric concepts of equality in positive predictive value and false omission rate [22] and equality of negative predictive value or false discovery rate. This perspective is in line with the point of view of the decision maker: Equal false discovery rates imply that an equal number of people who are denied a loan would have gone on to repay it [168].

Furthermore, fairness notions can be combined into composite fairness notions, such as *equalized odds* [112] or *separation* [13] when both equality in TPR and FPR are satisfied. Similarly, *sufficiency* [13] is satisfied if the decision yields both an equalized false omission rate and an equalized false discovery rate. If both TPR and TNR are equally important, it is also natural to consider equality in accuracy [47] as a measure of fairness.

While fairness definitions have previously been defined based on the decision D, it is naturally to also consider notions that rely on scores S. As a result, we might require that score-based performance metrics (e.g. area under the curve, Brier score) be equal across groups. The notion of **calibration** [13] is satisfied if the models are calibrated within each group: $\mathbb{P}(Y|S, A = a) = S \; \forall \, a \in \{0, 1\}$. This essentially asks, that the score assigned to each individual accurately reflects the true probability of $Y = 1$.

Equal Predictions based metrics only consider the implications of the decision D independent of the outcome Y. This stands in contrast to *equal error* based notions, which imply that a form of *merit*, (e.g. the individual would pay back the loan) should be considered when assessing the fairness of a decision, as they depend on the decision D and the true outcome Y. This can be motivated from several perspectives: Given an intervention that is generally positive (e.g. admission to a promotional program), we could be interested in distributing the intervention fairly across the sub-population and therefore ask that the likelihood of being admitted be independent of the sensitive group. In other scenarios, the true labels might suffer from *differential measurement error*. To provide an example, we cannot truly measure whether a defendant re-offends but are instead measuring if a defendant gets caught re-offending. Differences in policing can now lead to highly different measurement errors in the outcome of interest Y [126]. Lacking reliable labels, a fairness notion could then ask for equal predictions instead. Notions that only depend on predictions have been called **demographic parity** [49] or **independence** [13] in literature.
While many other definitions exist, we refer the interested reader to the relevant literature [22, 135, 165].

From independence statements to fairness metrics What is now left, is to transform the notions introduced above into fairness metrics.
A statistical group fairness metric $\Delta c : \mathcal{Y} \times \mathcal{Y}' \times \mathcal{A} \rightarrow \mathbb{R}$ translates a set of true labels, predictions and protected attributes into a real-valued score. Denoting with $\mathbb{P}_{xy}^{\{A=a\}}$ the data distribution of samples from the protected group a, we can now construct fairness

Equality	Independence Statement	Name / Reference
Predictions:		
PR	$D \perp A$	Demographic Parity [49], Independence [13]
Errors:		
TPR & FNR	$D \perp A \mid Y = 1$	Equality of opportunity [112]
FPR & TNR	$D \perp A \mid Y = 0$	Predictive Equality [57]
FDR & PPV	$Y \perp A \mid D = 1$	Predictive Parity ([57])
FOR & NPV	$Y \perp A \mid D = 0$	[22]
	$Y \perp A \mid S$	Calibration [13]
Composition:		
TPR & FPR	$Y \perp A \mid D$	Equalized Odds [112], Separation [13]
FDR & FOR	$Y \perp A \mid D$	Sufficiency [13]

Table 2.2: Overview of commonly used statistical fairness metrics. Metrics consider the true label Y, predicted scores S and derived decisions D as well as the protected group A.

metrics from different performance metrics or loss functions L. Given a fitted model $\hat{f}$, we can measure metrics of interest[5] in groups $A = 0$ and $A = 1$ and aggregate using an aggregation function h.

$$\Delta c = h\left(\mathbb{E}_{(X,Y)\sim\mathbb{P}_{xy}^{\{A=1\}}}\left[L(Y,\hat{f}(X))\right], \mathbb{E}_{(X,Y)\sim\mathbb{P}_{xy}^{\{A=0\}}}\left[L(Y,\hat{f}(X))\right]\right) \tag{2.16}$$

Typical choices for h are the absolute difference $h(x1, x2) = |x1 - x2|$ or the symmetric relative difference $h(x1, x2) = min(\frac{x1}{x2}, \frac{x2}{x1})$ [6]. To provide an example, we can construct the *absolute differences in false positive rates* as a fairness metric by employing the false positive rate as a loss function L and the absolute difference as an aggregation function. We denote this quantity with $|\Delta FPR|$ as an example for a concrete fairness metric.
In many practical applications, perfect fairness (i.e. $|\Delta FPR| = 0$) is unlikely to be achieved due to noise in the data and labels. Therefore, constraints $|\Delta FPR| \leq \alpha$ are introduced to differentiate between models that satisfy the fairness criterion and those that do not. A typical value for α is 0.05 [213].
Given a set of fairness metrics, it is natural to ask whether models $\hat{f}$ should satisfy only one or multiple fairness notions (e.g. equality of opportunity and predictive parity). A number of impossibility theorems, that establish the incompatibility between different sets of fairness notions exist [140]. Consequentially, several different fairness notions cannot be (perfectly) satisfied jointly except for trivial cases. In many scenarios, however, we are interested in investigating trade-offs between the different notions [65, 6].

[5]e.g. performance metrics or other statistics derived from the confusion matrix, such as positive rates

It is important to note that the observational criteria that have been the focus of this section are by no means infallible. If labels $y^{(i)}$ for example include unrecognized biases (e.g. through a biased annotation procedure), observational criteria have no hopes of recovering this bias from the data and might therefore erroneously certify a model as unbiased. Likewise, if systemic or societal injustices are baked into the data used to arrive at a decision (e.g. in the form of more prior arrests or a lower GPA), producing fair models might require severely deteriorating the utility of the resulting fair model. Causal notions of fairness [147, 134, 263] purport to solve such questions by taking the data generating process into account. However, in order to do so, they require access to the underlying causal graph, which is often prohibitively complex, especially in the context of higher-dimensional data.

2.3.2 Bias Mitigation

A wide variety of bias mitigation techniques have been proposed. Several approaches emphasize improved documentation of data and derived artifacts [19, 98] in order to increase awareness for fairness-related issues. Simultaneously, a set of methods that aim to yield fairer models by adapting data, model, or predictions, exist. Given a fitted ML model $\hat{f}$ that exhibits bias with respect to a chosen fairness metric Δc, we are now interested in obtaining a new model $\hat{f}^*$ that improves over $\hat{f}$ with respect to the fairness criterion (while not severely degrading predictive performance). Several approaches to arrive at a *fairer* model $\hat{f}^*$ have been proposed in the literature. Implementations of such debiasing methods can be found in different toolkits like the Aequitas toolkit [210], IBM AI Fairness 360 [18] or fairlearn [27]. They can mainly be classified into three categories that also constitute parts of the ML pipeline schematic depicted in Figure 1.1, *pre-, in- and postprocessing* introduced in Section 2.2.6. We will present only a few techniques for brevity and instead refer the interested reader to a survey [165].

Preprocessing A pre-processing bias mitigation technique transforms input features $\mathbf{x}$ via a pre-processing operator $\mathfrak{P}_{preproc}$. This operator requires an additional argument $\mathcal{A}$ during training. Note, that we denote with $\mathcal{A}$ the space of protected attributes to make dependence on $\mathcal{A}$ explicit, although technically $\mathcal{A} \subset \mathcal{X}$. The resulting transformed features are then passed on to the next pipeline operator, e.g. a ML algorithm $\mathfrak{P}_{learner}$.
In many scenarios, the reason for *unfair* models is a lack of *sufficient and representative* data across all sub-populations. Preprocessing methods can now augment the training data so that subsequently fitted models become fairer. Reweighing [129] was proposed to enforce demographic parity by assigning each datapoint a weight $w^{(i)} = \frac{P(Y=y^{(i)}) \cdot P(A=a^{(i)})}{P(Y=y^{(i)}, A=a^{(i)})}$, essentially transforming the sampling distribution, such that statistical independence $Y \perp A$ is achieved [6]. Other approaches rely on adapting the associated labels [88], or by learning *fair representations*, e.g. using auto-encoders [160] or adversarial learning strategies [261].

Inprocessing Fairness can also be directly incorporated into the *first level optimization* problem by including additional constraints on fairness criteria that should be satisfied by a model. In this case, a *fair* inducer $\mathcal{I}$ is fitted directly to the data without any need for preprocessing. This directly yields a fair model $\hat{f}' : \mathcal{X} \times \mathcal{A} \to \mathcal{Y}$. Considering this in the context of an ML pipeline, we can plug the fair inducer in at the model step $\mathfrak{P}_{learner}$ and combine it with different pre- and postprocessing steps.
In this vein, fair versions of linear models have been proposed that satisfy different fairness criteria. They operate by incorporating fairness constraints into the optimization problem, for example, by imposing constraints on the solution space [260, 259], by employing regularization [130] or by learning decoupled classifiers [240].

Post-processing A post-processing step $\mathfrak{P}_{postproc}$ takes as input predictions from a previous pipeline operator, e.g. a learner operator $\mathfrak{P}_{model}$. This operator again might require an additional argument $\mathcal{A}$, the protected attribute during training or prediction.
The equalized odds post-processing step introduced by [112, 198] e.g. allows for obtaining prediction that satisfy the equalized odds fairness definition by flipping randomly selected predictions in each group Y and D according to probabilities that are chosen, such that flipped predictions D' satisfies the equalized odds criterion, and that simultaneously D is as close to D' as possible. Other approaches [113, 136] rely on the more general notion of *multi-calibration*, that tries to achieve calibrated predictors in arbitrary sub-groups.

A common weakness of several bias mitigation techniques is that they usually optimize for a single fairness criterion. As a result, for each given fairness criterion, only a subset of bias mitigation techniques are applicable. Several attempts to arrive at more robust, criterion-independent approaches have been proposed [5, 6] in order to address this issue.

2.3.3 Contributions

Several open research directions in the broader context of fairness exist: While different bias mitigation techniques have been proposed, it is by no means clear, which actually reliably work in practice. It is therefore natural to compare the efficacy of different debiasing techniques in practical applications done in [6] presented in Section 5.1.
We find that in practice debiasing techniques do not always improve fairness (especially if fairness criterion and debiasing technique are not aligned), and that improving fairness often comes with steep decreases in *utility*, e.g. predictive performance. We simultaneously find that the estimation of the fairness metric Δc suffers from high variance, which makes obtaining reliable estimates difficult, e.g. in the context of tuning or model selection.
In contrast, other work has observed that fairness-accuracy trade-offs can also be comparably small in public policy scenarios [206]. The question is therefore how big trade-offs are when less simplistic models or more involved tuning methods are employed.
In this vein, it might be possible to combine different pre-, in- and postprocessing steps into a single ML pipeline $\mathfrak{P}_{postproc} \circ \mathfrak{P}_{learner} \circ \mathfrak{P}_{preproc}$. This, in combination with *partial debiasing* [6], could lead to more robust bias mitigation techniques. Simultaneously, the different

processing steps g as well as the inducer $\mathcal{I}$ usually have hyperparameters $\boldsymbol{\lambda}$ that can be tuned for better performance. A first attempt at this was introduced in a contribution in this thesis in the context of a *multi-objective* AutoML system [187] presented in Section 4.5. A step towards reconciling causal fairness notions with practical applications was developed in a publication [68], which employs multi-objective counterfactuals [67] to create counterfactual instances $\mathbf{x}'$ for an individual $\mathbf{x}$ with a different predicted attribute. This allows one to answer questions along the lines of *"what would the model prediction be, had the individual been male instead of female"*, but does not require access to the causal DAG. It is important to note that such procedures are not able to guarantee causal fairness definitions, but they might be a first step towards better integrating the two perspectives.

2.4 Benchmarks & Software

Democratizing machine learning does not only require developing methodology that allows for creating more powerful and accurate ML models, but simultaneously requires that a broader public can actually make use of such technology. This is achieved mainly in three ways: First, by extending education in ML to a broader audience, such that the relevant concepts required for successfully applying ML are thoroughly understood, while simultaneously creating awareness for possible ethical problems that might arise from the development of such systems. Secondly, by determining which methods proposed in scientific literature should actually be used in practice. With the large number of novel methods that are proposed each year, it is unclear which methods should be used and which have not shown promise in practical applications. Such questions can be answered by *benchmarks*, that allow for an unbiased comparison of methods proposed to solve one or several related problems. A third approach towards democratization is creating user-friendly software that provides implementations of relevant methods through an easily understandable API. This takes away the need to implement algorithms together with considerations of efficiency and parallelization that often exhibit considerable difficulty. In lieu of introducing the relevant notation and formalizing the problem to be solved, the following two sections will discuss contributions to the areas of *benchmarking* (in the form of concrete benchmarks) and *software* (in the form of open source software libraries).

2.4.1 Benchmarks

While small benchmarks are usually included with the publication of a new method in order to demonstrate that the new method compares favorably to other methods, those benchmarks are often by no means conclusive (c.f. [107, 137, 257]). This is partially due to the fact that incentives for authors of a publications are not aligned with the goal of an honest benchmark study: While the author's goal is to report favorable outcomes for a novel method, the goal of an impartial benchmark study is to provide a fair comparison of relevant methods on a representative sample of relevant problems. It is important to note that this does not neccessarily imply a lack of honesty in publications. The *multiplicity of design decisions* [177] that are required when designing a benchmark study can lead to different results depending on the exact specifications of the research questions. This comes in tandem with the fact, that authors usually optimize a proposed method for robustness on considered scenarios, while relying on publicly available, potentially less optimized implementations of competitor methods. Another problem is *cherry picking*, also termed *rigging the lottery* [71], that manifests when authors choose to report only the subsets of results from a benchmark that demonstrate a favorable outcome for the proposed method.

Many of the problems related to benchmarking can be ameliorated through the use of standardized testing infrastructure along with a detailed definition of evaluation criteria that are widely adapted or required for publication. Several examples for such testbeds exist, for example, the AutoML benchmark [100] for AutoML systems, the CC-18 collection

[29] of datasets for classification benchmarks, or the WILDS [143] benchmark for domain generalization.

Contributions This thesis encompasses several contributions that provide impartial benchmarks for relevant problems in the broader area of machine learning: The overarching goal of the different benchmark studies is *a*) to provide a fair testbed for algorithms proposed in the broader literature that can be used when new methods for a given setting are developed, *b*) to identify methods that work robustly across a set of representative and realistic problems and *c*) to advance scientific knowledge by identifying the core components that lead to improved results, allowing for more precise insights into scientific progress based on comparison of different techniques as well as ablation analyses.

One such benchmark study investigates different encoding techniques for categorical variables. Such variables are often present in tabular datasets, and pose a problem for the application of different ML algorithms (e.g. Support Vector Machines). As a result, categorical variables have to be encoded using so-called *categorical encodings* that yield a numeric representation of categorical variables. This typically happens in ML pipelines in the form of a pre-processing step. Data including the categorical variables are fed into a *categorical encoder* $\mathfrak{P}_{preproc}$ that transforms the data into a fully numeric dataset, which can subsequently be used to fit and predict using any inducer $\mathcal{I}$. Our contribution, [180] presented in Section 6.2 conducts a benchmark of widely used *categorical encoding* strategies across a variety of datasets including high-cardinality categorical variables. We find, that more sophisticated, regularized categorical encoding schemes that take into account the relationship with the target variable Y indeed outperform simpler strategies.

Another interesting setting is the field of *functional data analysis* (FDA), Functional data, includes functional variables as part of the covariates or as the outcome in a regression setting. Functional variables are derived from a theoretically infinite dimensional function, that in practice has been observed or measured at discrete points. To provide an example, consider recordings from an Electrocardiogram (ECG). While the underlying activity is a continuous signal, we can only measure it at a finite amount of time points, e.g. every $0.01s$ (100 Hz). If we are now interested in e.g. using such ECG records to identify patients with different heart diseases that can be recognized from such records, the data under consideration is potentially very high-dimensional. Such scenarios have been studied in the context of time series classification [12] as well as in the domain of functional data analysis [89].
In one contribution, [186] presented in Section 6.1 we investigate the efficacy of classification methods proposed for functional data along with a variety of baselines that transform the high-dimensional functional space into a lower-dimensional representation using a pre-processing step $g_{preproc}$ and subsequently fit widely used machine learning models for tabular data.

Furthermore, we present a benchmark studying the benefits of employing widely used domain generalization methods in the context of predicting patient survival on clinical datasets. The contribution [188] presented in Section 6.3 specifically adapts methods developed in the context of image classifications and evaluates whether methods can be transferred to other scenarios, such as the low-dimensional, tabular scenario studied in our contributions.

The field of HPO suffers from some of the problems related to unreliable benchmark results described above. HPO methods are often evaluated on synthetic test functions that are not representative for real-world problems they are intended to solve. Simultaneously, realistic test-beds for a systematic evaluation have only recently been proposed [77, 192]. We propose a surrogate based test-bed, especially for the evaluation of multi-objective HPO methods in [192] presented in Section 4.7. Our test-bed is based on surrogate models that strike a favorable trade-off between cheap-to-evaluate objectives $c(\lambda)$ and realistic tuning instances, providing e.g. *mixed, hierarchical* search spaces and allowing for multi-fidelity evaluations. Another contribution, [214], presented in Section 4.8 studies HPO methods developed for the NAS context in more detail and investigates which components of a recently presented NAS system actually lead to improved performance.

2.4.2 Software

Similar to benchmarks, readily available implementations of ML toolboxes can improve access to a broader public through the use of streamlined APIs. In the context of ML, machine learning toolboxes like `scikit-learn` [184], `weka` [110] and `mlr` [30, 148] have transformed the landscape for applying ML, enabling non-experts to fit machine learning models without deeper understanding of the algorithms or significant programming background. In the context of deep learning, the same phenomenon can be observed. Deep learning libraries with intuitive APIs as well as highly optimized implementations have made deep learning accessible to a broad public. Libraries like `keras` [56], `tensorflow`[1], `pytorch` [181] or `jax` [36] have simplified the implementation of neural networks along with complex, distributed training procedures. In the context of this thesis, several software packages implementing relevant novel methods for the Areas of AutoML & Fairness have been developed. It is important to note that software contributions included in this thesis do not merely have the goal to provide reference implementations of relevant ML methods. They instead aim at making methods available to users and therefore are *thoroughly tested* and include *extensive documentation* and, as a consequence, make typical machine learning workflows more efficient.

The `mlr3` [148] framework is a ML toolbox implementing a large variety of machine learning algorithms along with infrastructure for model evaluation and tuning of resulting model. The `mlr3pipelines` [26] framework extends this framework towards machine learning pipelines discussed in Section 2.2.6. It provides an intuitive *"meta-language"* that allows for expression complex machine learning pipelines in the form of chained processing steps.

This is enabled by expressing ML pipelines as a *directed acyclic graph* (DAG) containing nodes that represent pipeline processing steps, so-called *PipeOps*. Data then flows from an input node and is transformed (or turned into model predictions) at every stage of the DAG. Pre- and post-processing steps as well as ML algorithms can now be embedded in PipeOps in order to form typical ML pipelines. This framework allows users to specify complex ML pipelines intuitively in only few lines of code and can simultaneously serve as the basis of powerful AutoML systems. By representing choices between different preprocessing steps as traversals of the DAG that can be controlled by hyperparameters, we can represent a search space over ML pipelines that is typically included in AutoML systems.

This manuscript furthermore includes two software packages aimed at improving the user experience regarding bias auditing and mitigation: The *mlr3fairness* package [196] presented in Section 6.5 provides a toolbox for exactly this purpose. It similarly extends the `mlr3` [148] package by providing measures that implement fairness metrics presented in Section 2.3. It furthermore includes *pre-, in- and post-processing* bias mitigation techniques (in the form of processing steps for `mlr3pipelines`) that can be combined into machine learning pipelines and be automatically tuned using readily available tools like `mlr3tuning`. The `mcboost` package [190] included in Section 6.6 provides a set of additional bias mitigation techniques, *multi-calibration* [113] and *multi-accuracy* [136].

2.5 Further Contributions

Several additional contributions and publications by the author of this dissertation were submitted or published during research for this dissertation. They are not explicitly included in this thesis and instead briefly presented in this section. They span a wide variety of interesting topics in the broader context of data science and machine learning.

The topic of human involvement in AutoML was discussed in a short article [193]. The work argues for incorporating a multi-objective perspective when developing an AutoML system that can incorporate fairness constraints or constraints on the transparency of resulting models.
I also contributed to a benchmark that proposes new problem instances for Quality Diversity Optimization [216] based on HPO problems.

In addition, I contributed to several publications that are not publicly available at the time of writing this thesis:
A survey on multiobjective HPO is presented in [132]. I contributed several sections to this survey, specifically a section on *multi-objective machine learning* and several sections on application areas for multi-objective HPO, specifically on interpretability, fairness, and robustness metrics.
In another collaboration, I contributed to a publication that proposes a geometric framework for outlier detection on functional (time-series) data [115]. It proposes using manifold learning methods to aid in developing and fitting outlier detection models.

Throughout my PhD I furthermore supervised the *"Innovationslabor Big Data Science"*, which resulted in an additional publication, [94]. The publication employs different deep learning methods with the goal to automatically classify general weather conditions over Europe based on atmospheric measurements.

I also contributed to several software packages in the `mlr3` [148] ecosystem. This includes several contributions to improve documentation of `mlr3` and the mlr3 book. I furthermore developed the `mlr3keras` [191] package and helped in developing several packages through feedback and minor contributions, e.g. the `bbotk`, `mlr3multioutput`, `mlr3fda` and the `mlr3temporal` package[6]. In addition, I contributed several articles to the mlr3 gallery[7]

I furthermore contributed to the *deepregression* software [208] that implements a framework for semi-structured deep distributional regression mostly on the software side, e.g. by refactoring of the implementation, fixing implementation issues, and general improvements and additions to the code base. A second manuscript [209] was created based on an extension of the deepregression package towards the flexible estimation of distribution

[6] Available from `https://github.com/mlr-org`
[7] `https://mlr3gallery.mlr-org.com/`

mixture models based on neural networks.

Publications

[94] H. Funk, C. Becker, A. Hofheinz, G. Xi, Y. Zhang, F. Pfisterer, M. Weigert, and M. Mittermeier. Towards an automated classification of Hess & Brezowsky's atmospheric circulation patterns Tief and Trog Mitteleuropa using deep learning methods. In *Environmental Informatics – A bogeyman or saviour to achieve the UN Sustainable Development Goals? - Adjunct Proceedings of the 35th edition of the EnviroInfo – the long standing and established international and interdisciplinary conference series on leading environmental information and communication technologies*, pages 22–30. Shaker Verlag GmbH, 2021
[115] M. Hermann, F. Pfisterer, and F. Scheipl. A geometric framework for outlier detection in high-dimensional data. Manuscript submitted for publication
[132] F. Karl, T. Pielok, J. Moosbauer, F. Pfisterer, S. Coors, M. Binder, L. Schneider, J. Thomas, J. Richter, M. Lang, E. C. Garrido-Merchán, J. Branke, and B. Bischl. Multi-objective hyperparameter optimization – an overview, 2022
[216] L. Schneider, F. Pfisterer, J. Thomas, and B. Bischl. A collection of quality diversity optimization problems derived from hyperparameter optimization of machine learning models. In *GECCO '22: Proceedings of the Genetic and Evolutionary Computation Conference Companion*, Boston, United States of America, 2022. ACM
[148] M. Lang, M. Binder, J. Richter, P. Schratz, F. Pfisterer, S. Coors, Q. Au, G. Casalicchio, L. Kotthoff, and B. Bischl. mlr3: A modern object-oriented machine learning framework in R. *Journal of Open Source Software*, dec 2019
[193] F. Pfisterer, J. Thomas, and B. Bischl. Towards human centered AutoML, 2019, arXiv:1911.02391
[191] F. Pfisterer, J. Poon, and M. Lang. *mlr3keras: mlr3 Keras extension*, 2021. R package version 0.1.3
[209] D. Rügamer, F. Pfisterer, and B. Bischl. Neural mixture distributional regression. 2020, arXiv:2010.06889
[208] D. Rügamer, C. Kolb, C. Fritz, F. Pfisterer, P. Kopper, B. Bischl, R. Shen, C. Bukas, L. B. de Andrade e Sousa, D. Thalmeier, P. Baumann, L. Kook, N. Klein, and C. L. Müller. deepregression: a flexible neural network framework for semi-structured deep distributional regression. *Journal of Statistical Software (provisionally accepted)*, 2022

CHAPTER 3 OUTLOOK & FUTURE DIRECTIONS

The greater goal of *democratizing machine learning* encompasses several aspects that have not been discussed as a part of this thesis. These aspects include access to high-quality data, access to sufficient compute resources, and access to tools and expertise [238]. To provide an example, access to high-quality curated data-sets is often only available at considerable cost, and therefore out of reach for e.g. NGOs with limited budget. In the author's personal experience, this has happened in the context of obtaining satellite data with sufficiently high resolution to allow for road segmentation. Similarly, access to compute resources and expertise is key to equitable access to ML.
Contributions in this thesis have mainly focused on improving the *tooling* aspect of ML: By speeding up, improving, or robustifying AutoML, by studying multi-objective AutoML systems that can include secondary objectives, e.g. *fairness* and lastly by providing readily available implementations and toolboxes for important methods. The remainder of this manuscript now tries to draw a wider frame: All of those aspects constitute important facets of *ideal* AutoML systems, as they reflect the components or capabilities of such systems. It is now natural to ask what other important facets should be considered for AutoML systems in the future and what else should be automated. An answer to these questions is given by *Zoubin Ghahramani* in [122, pp. vii-viii]:

> "... we should try to automate all aspects of the entire machine learning and data analysis pipeline. This includes automating data collection and experiment design; automating data cleanup and missing data imputation; automating feature selection and transformation; automating model discovery, criticism, and explanation; automating the allocation of computational resources; automating hyperparameter optimization; automating inference; and automating model monitoring and anomaly detection. This is a huge list of things, and we'd optimally like to automate all of it. There is a caveat of course. While full automation can motivate scientific research and provide a long-term engineering goal, in practice, we probably want to semi automate most of these

> and gradually remove the human in the loop as needed. Along the way, what is going to happen if we try to do all this automation is that we are likely to develop powerful tools that will help make the practice of machine learning, first of all, more systematic (since it's very ad hoc these days) and also more efficient."

This quote echoes two sentiments that are important in the context of this manuscript: Some things should perhaps not be automated directly but instead involve a human in the loop, which is discussed e.g. in the context of *multi-objective AutoML* (see Section 2.2.2) or in the context of *fairness* (see Section 2.3). The other sentiment is the development of (and need for) powerful tools that improve the practice of machine learning. This is e.g. reflected in the software developed as part of this thesis (see Section 2.4.2) as well as benchmarks (see Section 2.4.1) that systematically assess individual components of AutoML systems.

3.1 AutoML – a holistic perspective

In order to now draw this wider frame, it is important to look at the broader picture and disregard *"implementation"* differences between AutoML systems, e.g. in the form of different optimization routines or search space considerations (e.g. whether NAS or HPO is used). We'll instead focus on more general *capabilities* of such systems, such as the *mode* of data (tabular, images, ...) that systems can consider or the *modeling goal*, e.g. in which context a model should be used and what should be predicted. AutoML systems have become powerful at solving a set of narrow, well-defined problems, e.g. *classification* on *tabular* datasets [100]. Furthermore, systems that solve different problems have been developed, for example, in the context of several AutoML challenges [109]. Similarly, *domain specific* AutoML systems have been developed, e.g. for medical imaging [85], predictive maintenance [237] or in the context of recommender systems [264]. While those systems have proven powerful in the narrow domains they were created for, they do not easily extend to different problem settings or data modalities. In the following, several requests are formulated that should be taken into consideration when developing future AutoML systems. While considerable progress has already been made in some of those directions, the problems described are not solved and especially widely available software that addresses the issues is often still missing.

Different situations require for different systems Developing AutoML systems requires significant software engineering effort - systems need not only provide good performance, but also do so efficiently which requires parallelization and efficient implementations. As a downside, this often results in *monolithic*, closed systems that are only extensible in few directions. If the intended use of AutoML systems now slightly deviates from a problem at hand, the system might be completely unusable, e.g, due to not being able to account for relatively minor problem characteristics.
An (incomplete) collection of such characteristics is provided in Figure 3.1. We consider categories *output modality*, *evaluation details* and *problem details*. While *data modality*

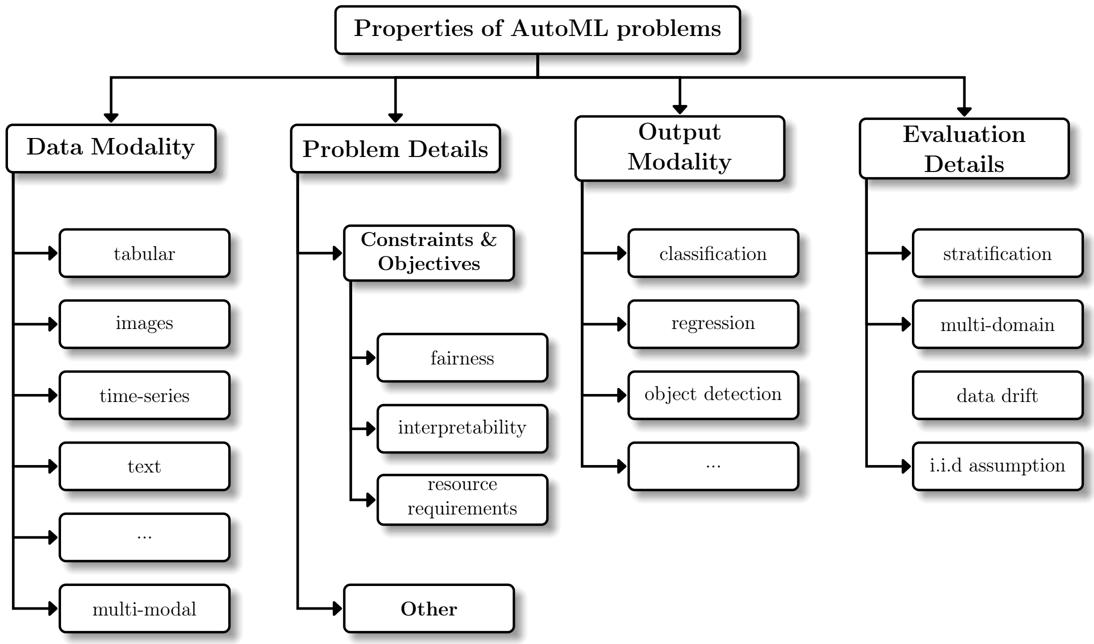


Figure 3.1: Excerpt of typical problem characteristics encountered in practical ML applications.

considers the format of the input features, we call the format of labels / desired outputs *output modality*. Depending on the problem at hand, several other details might be relevant, e.g. deviations from the *i.i.d.* assumption of datasets $\mathcal{D}$, here called *evaluation details*. Depending on the use case, we might also be interested in imposing additional constraints on the resulting model to ensure that it fits the intended purpose.

To provide an example, consider the fairly trivial example of *energy demand forecasting*. Based on tabular features (e.g. forecasted weather, geo-location, ...) detailing past energy usage, we are interested in forecasting demand in the coming month. Valid evaluation of such a task requires that train and test data are stratified according to the date to provide an unbiased estimate of the generalization error on *future* data. An AutoML system needs to be able to incorporate such stratification at the outer (evaluation) as well as the inner (tuning) level. Simultaneously, if it cannot do so, the system might not be useable for the given problem. Such and similar other problems also often occur in other scenarios, e.g. if additional structure is imposed on predictions, or if the system needs to monitor inputs for validity and outliers in production settings, as e.g. implemented in `tfx` [16] and other systems.

Different problem characteristics now often require specific solution components, e.g. a component that continuously monitors inputs and detects deviations from the training data, e.g. in the form of outliers produced through faulty sensors. An important observation in this context is that such components are often orthogonal to other components required to solve a specific problem. Outside of AutoML systems, this enables re-use and composition of different components to develop pipelines that better adapt to the problem characteristics at hand. An example of such a system is the *MLBazaar* proposed in [223], which provides several primitives that can be combined into full AutoML systems that allow multiple different *data* and *output* modalities.

This and other systems provide an important first step towards improving flexibility for AutoML systems, which is an important avenue of future research: Currently, AutoML systems are often tailored to a narrow application domain. While future AutoML systems that can incorporate all possible types of problems and problem characteristics are a worthwhile research goal, this might be out of reach in the near future. A good intermediate step might instead be *user-friendly*, *extensible* AutoML toolboxes that allow for constructing AutoML systems by composing primitives into flexible ML pipelines. This allows for incorporating new problem characteristics while reusing existing pipeline infrastructure where possible. It is important to note that composing AutoML systems does not need to happen at the level of *primitives*. By bundling primitives that are often used together into exchangeable functional components, one could provide a basis for new domain specific AutoML systems that have only few new components while being able to inherit functionality from an abstract baseline AutoML system. This could free up developers of AutoML systems to work on automating new aspects of the data science process or to improve functionality for specific domains where currently no solutions exist.

AutoML systems need to incorporate additional criteria and constraints One contribution in this thesis, [187] makes the case for multi-objective AutoML systems that allow for incorporating additional objectives in AutoML systems. This is required in many practical applications, where, e.g. considerations of energy usage or latency have to be considered when models should be deployed to edge devices [153]. While this is important, existing AutoML systems rarely allow optimizing for multiple objectives or incorporating constraints.

One likely reason for this is, that multi-objective optimization might not solve all problems that exist in such settings. In order to, e.g. reliably asses the energy usage of a model on an edge device, it would need to be deployed to such an edge device, which requires additional and more complicated evaluation protocols that do not only rely on predictions and labels but require the full model. Furthermore, we are often not interested in approximating the Pareto-optimal set. Instead, we might be interested in only a small set of trade-offs between models that is hard to specify a priori. This would then require adapting the goals of the AutoML system during training, e.g. by involving a human in the loop. One downside to this is that the AutoML system becomes less *automated*, defying the original goal of AutoML.

I conjecture that especially in the context of fairness, AutoML might have to fill a difficult role: Obtaining models that strike favorable trade-offs between fairness and utility is often difficult, especially considering the plethora of available bias mitigation techniques. Similarly, models might need to strike further trade-offs between different fairness metrics in order to be acceptable to regulators. Many of those problems might be solved at least to

some degree through AutoML systems in the future. At the same time, fairness can not be assessed based on observational data alone, and additional considerations (see Section 2.3) might be required to guard against potential harms. One possible solution could be *guided bias assessments* where a system and a user interact (e.g. in the form of a questionnaire) to ensure all possible sources of bias are considered.
Opening up AutoML to additional goals also might require to further involve the user in the loop, e.g. in contexts where user preferences need to be considered. In this context, simply providing the user with the Pareto-optimal set might also be suboptimal – if a user is faced with a choice between 100s of Pareto-optimal models that could potentially be further ensembled to form even more powerful predictors, it is unclear which model should be selected (especially if more than two objectives should be considered). Procedures to guide and improve this selection process provide further interesting areas of research.

An AutoML system should know when it does not know Individual models returned by an AutoML system are often ranked according to their estimated generalization error. Estimates of this error can exhibit large variance, e.g. in the context of fairness metrics [6]. This might lead to unreliable rankings or a violation of constraints, should such constraints be imposed on returned models. Current AutoML systems usually simply return a ranking of models according to a performance metric of interest [83] that allows the users to inspect and select a final model (or ensemble thereof). This often disregards information about the variance of said estimates or leads to less principled model selection procedures if variance information has to be considered by the user. Therefore, an interesting future avenue of research would be to consider that AutoML systems should include estimates of the variance of relevant metrics during training and when deciding which model to return.
Given an input $\mathbf{x}$, ML models will always return a prediction $\hat{f}(\mathbf{x})$. When models are used in production, new data points can stem from a distribution other than $\mathbb{P}_{xy}$. Such new samples might e.g. only slightly deviate from the original distribution (e.g. in the context of weak data drift), in which case model performance might only slightly deteriorate. But new samples could also come from a new class unseen during training, which could result in harmful decisions. It is therefore important that such changes in the data distribution or individual outliers are automatically detected even when a model is already deployed. Similarly, when models are employed in critical use cases, models should have the option to *abstain* from a prediction [99], e.g. if the uncertainty becomes too big or to return prediction sets (or intervals) instead, e.g. via *conformal* prediction [220].

AutoML systems should reflect the full pipeline In many settings, ML pipelines only constitute part of the full pipeline from data collection to decisions based on model predictions. To provide an example, data are often preprocessed and aggregated into more easily digestible formats before AutoML systems are used, as e.g. AutoML systems only allow for very specific input formats. The need for incorporating e.g. data management and labeling is emphasized in [9], as changes or updates to data can require re-design of subsequent steps in the ML pipeline. Likewise, an AutoML system might yield a model that produces optimal predictions in the narrow scope defined by the provided performance metric. But if model outputs are used, e.g. together with additional information, fairness guarantees regarding a process might no longer hold [76]. This becomes more problematic if processes involve multiple steps in which individuals are included or excluded from the analysis according to other criteria, or if multiple models are *composed* [167, 75] in different steps. Considering that we are usually interested in optimizing with respect to metrics of such a composite system, it is not sufficient to only consider parts of the pipeline when evaluating the model, as interactions with other components could lead to deteriorate relevant metrics.

This is especially important in the context of algorithmic fairness: If individuals are included or excluded before the ML pipeline (e.g. through fraud screening mechanisms prior to making a credit decision), fairness notions might not be computed with respect to a representative sample and, therefore, yield invalid results. This might lead to mistakenly assuming that the models are fair, although they are not.

The same can be observed in another contexts. If models, for example, should be deployed on a specific device, the evaluation criteria of AutoML systems need to consider the device to make sure that eventual constraints, such as e.g. *system requirements* are satisfied.

This presents a challenge to the current AutoML systems, that are currently often monolithic systems that are highly optimized and efficient for a narrow domain. This challenge is not only conceptual but largely also an engineering challenge: AutoML systems are often highly parallelized and distributed, which might make incorporating e.g. external processes that adequately reflect the full pipeline difficult.

AutoML is more than optimizing accuracy In essence, AutoML parametrizes a variety of design decisions that are typically made when developing ML algorithms. It then optimizes over the space spanned by those design decisions in a principled way, e.g. using a optimization method such as *Bayesian Optimization*.

This approach is not only relevant in the context of developing models but might also help to provide more robust results in the broader area of ML: When benchmarking new algorithms or methods in the broader context of ML and statistics, we are often faced with hyperparameters and design decisions (e.g. preprocessing) that might significantly impact results [177]. If all relevant design decisions are adequately reflected in the hyperparameter space of such an *analysis pipeline*, an optimizer can be used to tune over this space and as a result yield less biased results. This intuitively leads to questions about improving the interpretability of HPO methods [172] that pose interesting directions for future work.

AutoML needs to consider its societal impact AutoML as a field interacts with many other disciplines, since it augments the ability of applicants to obtain more powerful models. It is important to recognize, that more powerful models can also result in adverse outcomes for individuals or groups – if ethical considerations regarding the applications of such models are not adequately addressed. This can be exacerbated if users are not

sufficiently educated regarding potential problems and negative effects of such systems. To put it in different terms, *AutoML provides you with an answer, independently of whether the question is a good one to ask*. To provide an example, AutoML gives state actors the ability to conduct *face recognition* at a large scale and based on well-performing models. This might lead to negative consequences [222] (e.g. erroneous arrests) for individuals or be biased against groups [204] if models are not properly validated and the potential downsides of such systems are not explained sufficiently. AutoML can also partially solve some of the problems associated with this, e.g. by simultaneously providing tools that help better assess models with respect to such critical questions.
Simultaneously, the field of AutoML also needs to consider its environmental impact, especially in the context of NAS. Calls to consider this in the development of AutoML tools have become louder [236]. This could be further improved by stopping optimization when further improvements in the relevant objectives are no longer relevant (as e.g. determined by the user) or unlikely [162].

3.2 Conclusion

Better and more flexible AutoML systems are likely one way towards better democratization of machine learning. This thesis contains several contributions that advance knowledge in the broader field of AutoML. For AutoML systems to truly deliver on their promise of *automating all aspects of machine learning*, several hurdles must be cleared. Systems need to become more flexible in order to better adapt to different problem settings and this flexibility should optimally be available in the form of a user-facing API, such that users can adapt systems to the problem at hand. Furthermore, AutoML systems need to be able to incorporate the broader pipeline from data ingestion to decision-making, such that subtleties in those steps can be taken into account, e.g. during validation. Clearing those hurdles might be one way to drive user adoption of AutoML systems, along with other improvements in the experience, such as better interpretability of AutoML systems [172]. This provides several important future directions of research that might not only improve the state of the art in AutoML, but ultimately also lead to better access to ML for everyone.

In the context of algorithmic fairness, challenges and interesting avenues of research are abound. Causal fairness notions are often difficult to apply as causal DAGs are often ambiguous and complex. User friendly systems, that combine causal fairness notions with (automated) causal discovery while keeping the human-in-the-loop might be one way to drive adaptation of such notions in practice. Furthermore, it is an open question how useful bias mitigation is in many settings, as it often comes with a steep decrease in utility that warrants further investigation [65]. Comprehensive benchmark studies along with experience from real-world applications are needed to better understand the effects of such techniques.

Returning back to the metaphor of golems – this dissertation has presented contributions that enable a broader public to employ *statistical golems*, made out of circuits and numbers. Those golems are already powerful as they extend their master's capabilities in several directions. It is now important to build responsible golems by ensuring that they conform to ethical and legal standards – and ultimately – that they do not destroy their master in the process of fulfilling their mission [45, 97].

REFERENCES

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283, 2016.

[3] A. Abraham and R. Goldberg. *Evolutionary Multiobjective Optimization: Theoretical Advances and Applications*. Advanced Information and Knowledge Processing. Springer London, 2006.

[4] K. Ackermann, J. Walsh, A. De Unánue, H. Naveed, A. Navarrete Rivera, S.-J. Lee, J. Bennett, M. Defoe, C. Cody, L. Haynes, et al. Deploying machine learning models for public policy: A framework. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 15–22, 2018.

[5] A. Agarwal, A. Beygelzimer, M. Dudík, J. Langford, and H. Wallach. A reductions approach to fair classification. In *International Conference on Machine Learning*, pages 60–69. PMLR, 2018.

[6] A. Agrawal, F. Pfisterer, B. Bischl, J. Chen, S. Sood, S. Shah, F. Buet-Golfouse, B. A. Mateen, and S. Vollmer. Debiasing classifiers: is reality at variance with expectation?, 2020, arXiv:2011.02407.

[7] P. E. Agre. Lessons learned in trying to reform AI. *Social science, technical systems, and cooperative work: Beyond the Great Divide*, 131, 1997.

[8] D. Allhutter, F. Cech, F. Fischer, G. Grill, and A. Mager. Algorithmic profiling of job seekers in Austria: how austerity politics are made effective. *Frontiers in Big Data*, 3:5, 2020.

[9] S. Amershi, A. Begel, C. Bird, R. DeLine, H. Gall, E. Kamar, N. Nagappan, B. Nushi, and T. Zimmermann. Software engineering for machine learning: A case study. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 291–300. IEEE, 2019.

[10] K. Arnold, J. Gosling, and D. Holmes. *The Java programming language.* Addison Wesley Professional, 2005.

[11] T. Bäck and H.-P. Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary computation*, 1(1):1–23, 1993.

[12] A. Bagnall, J. Lines, A. Bostrom, J. Large, and E. Keogh. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data mining and knowledge discovery*, 31(3):606–660, 2017.

[13] S. Barocas, M. Hardt, and A. Narayanan. *Fairness and Machine Learning.* fairmlbook.org, 2019. `http://www.fairmlbook.org`.

[14] T. Bartz-Beielstein, J. Branke, J. Mehnen, and O. Mersmann. Evolutionary algorithms. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 4(3):178–195, 2014.

[15] T. Bartz-Beielstein, C. W. Lasarczyk, and M. Preuß. Sequential parameter optimization. In *2005 IEEE congress on evolutionary computation*, volume 1, pages 773–780. IEEE, 2005.

[16] D. Baylor, E. Breck, H.-T. Cheng, N. Fiedel, C. Y. Foo, Z. Haque, S. Haykal, M. Ispir, V. Jain, L. Koc, et al. Tfx: A tensorflow-based production-scale machine learning platform. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1387–1395, 2017.

[17] C. Becker. Multicalibration in survival analysis: Black-box post-processing for fairness. Unpublished master thesis, 2021.

[18] R. K. E. Bellamy, K. Dey, M. Hind, S. C. Hoffman, S. Houde, K. Kannan, P. Lohia, J. Martino, S. Mehta, A. Mojsilović, S. Nagar, K. N. Ramamurthy, J. Richards, D. Saha, P. Sattigeri, M. Singh, K. R. Varshney, and Y. Zhang. AI Fairness 360: An extensible toolkit for detecting and mitigating algorithmic bias. *IBM Journal of Research and Development*, 63(4/5):4:1–15, 2019.

[19] M. Benjamin, P. Gagnon, N. Rostamzadeh, C. Pal, Y. Bengio, and A. Shee. Towards standardization of data licenses: The Montreal data license. *arXiv:1903.12262*, 2019.

[20] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. *Advances in Neural Information Processing Systems*, 24, 2011.

[21] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *The Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[22] R. Berk, H. Heidari, S. Jabbari, M. Kearns, and A. Roth. Fairness in criminal justice risk assessments: The state of the art. *Sociological Methods & Research*, Aug. 2018, 1703.09207.

[23] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah. Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98, 2017.

[24] M. Binder. *mlrCPO: Composable Preprocessing Operators and Pipelines for Machine Learning*, 2021. R package version 0.3.7-3.

[25] M. Binder, F. Pfisterer, and B. Bischl. Collecting empirical data about hyperparameters for data driven AutoML. In *AutoML Workshop at ICML*, 2020.

[26] M. Binder, F. Pfisterer, M. Lang, L. Schneider, L. Kotthoff, and B. Bischl. mlr3pipelines - flexible machine learning pipelines in R. *Journal of Machine Learning Research*, 22(184):1–7, 2021.

[27] S. Bird, M. Dudík, R. Edgar, B. Horn, R. Lutz, V. Milan, M. Sameki, H. Wallach, and K. Walker. Fairlearn: A toolkit for assessing and improving fairness in AI. Technical Report MSR-TR-2020-32, Microsoft Research, Sept. 2020.

[28] B. Bischl, M. Binder, M. Lang, T. Pielok, J. Richter, S. Coors, J. Thomas, T. Ullmann, M. Becker, A.-L. Boulesteix, et al. Hyperparameter optimization: Foundations, algorithms, best practices and open challenges. *arXiv:2107.05847*, 2021.

[29] B. Bischl, B. Bischl, G. Casalicchio, M. Feurer, P. Gijsbers, F. Hutter, M. Lang, R. Gomes Mantovani, J. van Rijn, and J. Vanschoren. OpenML benchmarking suites. In J. Vanschoren and S. Yeung, editors, *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021.

[30] B. Bischl, M. Lang, L. Kotthoff, J. Schiffner, J. Richter, E. Studerus, G. Casalicchio, and Z. M. Jones. mlr: Machine learning in R. *The Journal of Machine Learning Research*, 17(1):5938–5942, 2016.

[31] B. Bischl, O. Mersmann, H. Trautmann, and C. Weihs. Resampling methods for meta-model validation with recommendations for evolutionary computation. *Evolutionary Computation*, 20(2):249–275, 2012.

[32] A. D. Blaom, F. Kiraly, T. Lienart, Y. Simillides, D. Arenas, and S. J. Vollmer. MLJ: A julia package for composable machine learning. *Journal of Open Source Software*, 5(55):2704, 2020.

[33] A. D. Blaom and S. J. Vollmer. Flexible model composition in machine learning and its implementation in MLJ, 2020.

[34] A.-L. Boulesteix, H. Binder, M. Abrahamowicz, W. Sauerbrei, et al. On the necessity and design of studies comparing statistical methods. *Biometrical journal. Biometrische Zeitschrift*, 60(1):216–218, 2017.

[35] A.-L. Boulesteix, S. Hoffmann, A. Charlton, and H. Seibold. A replication crisis in methodological research? *Significance*, 17(5):18–21, 2020.

[36] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[37] J. Branke. *MCDA and Multiobjective Evolutionary Algorithms*, pages 977–1008. Springer New York, New York, NY, 2016.

[38] P. Brazdil, C. Giraud-Carrier, C. Soares, and R. Vilalta. *Metalearning: Applications to Data Mining*. Springer Publishing Company, Incorporated, 1 edition, 2008.

[39] P. B. Brazdil and C. Soares. A comparison of ranking methods for classification algorithm selection. In *European conference on machine learning*, pages 63–75. Springer, 2000.

[40] P. B. Brazdil, C. Soares, and J. P. Da Costa. Ranking learning algorithms: Using IBL and meta-learning on accuracy and time results. *Machine Learning*, 50(3):251–277, 2003.

[41] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[42] L. Breiman and A. Cutler. Random forests Manual, 2020. `https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm`.

[43] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen. *Classification and regression trees*. CRC press, 1984.

[44] E. Brochu, V. M. Cora, and N. De Freitas. A tutorial on Bayesian Optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv:1012.2599*, 2010.

[45] M. Brundage. Taking superintelligence seriously: Superintelligence: Paths, dangers, strategies by Nick Bostrom (oxford university press, 2014). *Futures*, 72:32–35, 2015.

[46] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Müller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, et al. API design for machine learning software: experiences from the scikit-learn project. *arXiv:1309.0238*, 2013.

[47] J. Buolamwini and T. Gebru. Gender shades: Intersectional accuracy disparities in commercial gender classification. In *Conference on fairness, accountability and transparency*, pages 77–91. PMLR, 2018.

[48] K. T. Butler, D. W. Davies, H. Cartwright, O. Isayev, and A. Walsh. Machine learning for molecular and materials science. *Nature*, 559(7715):547–555, 2018.

[49] T. Calders and S. Verwer. Three naive bayes approaches for discrimination-free classification. *Data Mining and Knowledge Discovery*, 21(2):277–292, 2010.

[50] B. Caputo, K. Sim, F. Furesjo, and A. Smola. Appearance-based object recognition using svms: Which kernel should i use? In *Procceedings of NIPS workshop on Statitsical methods for computational experiments in visual processing and computer vision, Whistler*, pages 1–10, 2002.

[51] R. Caruana, A. Niculescu-Mizil, G. Crew, and A. Ksikes. Ensemble selection from libraries of models. In *Proceedings of the twenty-first international conference on Machine learning*, page 18, 2004.

[52] J. Chen. Fair lending needs explainable models for responsible recommendation. In *Proceedings of the 2nd FATREC Workshop on Responsible Recommendation*, Sept. 2018, 1809.04684.

[53] J. Chen, N. Kallus, X. Mao, G. Svacha, and M. Udell. Fairness under unawareness: Assessing disparity when protected class is unobserved. In *Proceedings of the conference on fairness, accountability, and transparency*, pages 339–348, 2019.

[54] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[55] S. Chiappa. Path-specific counterfactual fairness. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7801–7808, 2019.

[56] F. Chollet et al. Keras. `https://keras.io`, 2015.

[57] A. Chouldechova. Fair prediction with disparate impact: A study of bias in recidivism prediction instruments. *Big Data*, 5(2):153–163, June 2017, 1703.00056.

[58] A. Chouldechova and A. Roth. A snapshot of the frontiers of fairness in machine learning. *Communications of the ACM*, 63(5):82–89, 2020.

[59] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, P. Schuh, K. Shi, S. Tsvyashchenko, J. Maynez, A. Rao, P. Barnes, Y. Tay, N. Shazeer, V. Prabhakaran, E. Reif, N. Du, B. Hutchinson, R. Pope, J. Bradbury, J. Austin, M. Isard, G. Gur-Ari, P. Yin, T. Duke, A. Levskaya, S. Ghemawat, S. Dev, H. Michalewski, X. Garcia, V. Misra, K. Robinson, L. Fedus, D. Zhou, D. Ippolito, D. Luan, H. Lim, B. Zoph, A. Spiridonov, R. Sepassi, D. Dohan, S. Agrawal, M. Omernick, A. M. Dai, T. S. Pillai, M. Pellat, A. Lewkowycz, E. Moreira, R. Child, O. Polozov, K. Lee, Z. Zhou, X. Wang, B. Saeta, M. Diaz, O. Firat, M. Catasta, J. Wei, K. Meier-Hellstern, D. Eck, J. Dean, S. Petrov, and N. Fiedel. Palm: Scaling language modeling with pathways, 2022.

[60] J. T. Clemons. Blind injustice: The supreme court, implicit racial bias, and the racial disparity in the criminal justice system. *Am. Crim. L. Rev.*, 51:689, 2014.

[61] H. M. Collins, T. Pinch, et al. *The golem: What you should know about science.* Cambridge University Press, 1998.

[62] U. Congress. S. 1745 (102nd) civil rights act of 1991, 1991.

[63] A. F. Cooper, E. Abrams, and N. NA. Emergent unfairness in algorithmic fairness-accuracy trade-off research. In *Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society*, pages 46–54, 2021.

[64] R. Cooper and M. Foster. Sociotechnical systems. *American Psychologist*, 26(5):467, 1971.

[65] S. Corbett-Davies, E. Pierson, A. Feller, S. Goel, and A. Huq. Algorithmic decision making and the cost of fairness. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, page 797–806, New York, NY, USA, 2017. Association for Computing Machinery.

[66] D. D. Cox and S. John. A statistical method for global optimization. In *Proceedings of the 1992 IEEE International Conference on Systems, Man, and Cybernetics*, pages 1241–1246. IEEE, 1992.

[67] S. Dandl, C. Molnar, M. Binder, and B. Bischl. Multi-objective counterfactual explanations. In *International Conference on Parallel Problem Solving from Nature*, pages 448–469. Springer, 2020.

[68] S. Dandl, F. Pfisterer, and B. Bischl. Multi-objective counterfactual fairness. In *GECCO '22: Proceedings of the Genetic and Evolutionary Computation Conference Companion*, page 328–331, Boston, United States of America, 2022. ACM.

[69] K. A. de Jong. *Evolutionary computation.* Evolutionary Computation. Bradford Books, Cambridge, MA, Feb. 2006.

[70] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[71] M. Dehghani, Y. Tay, A. A. Gritsenko, Z. Zhao, N. Houlsby, F. Diaz, D. Metzler, and O. Vinyals. The benchmark lottery. *arXiv:2107.07002*, 2021.

[72] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255. Ieee, 2009.

[73] D. Dua and C. Graff. UCI machine learning repository, 2017.

[74] C. Dwork, M. Hardt, T. Pitassi, O. Reingold, and R. Zemel. Fairness through awareness. In *Proceedings of the 3rd innovations in theoretical computer science conference*, pages 214–226, 2012.

[75] C. Dwork and C. Ilvento. Fairness under composition. *arXiv:1806.06122*, 2018.

[76] C. Dwork, C. Ilvento, and M. Jagadeesan. Individual fairness in pipelines. *arXiv preprint arXiv:2004.05167*, 2020.

[77] K. Eggensperger, P. Müller, N. Mallik, M. Feurer, R. Sass, A. Klein, N. Awad, M. Lindauer, and F. Hutter. Hpobench: A collection of reproducible multi-fidelity benchmark problems for hpo. In *Proceedings of the international conference on Neural Information Processing Systems (NeurIPS) (Datasets and Benchmarks Track)*, Dec. 2021.

[78] M. Ehrgott. *Multicriteria Optimization (2. ed.)*. Springer, 2005.

[79] C. Elkan. The foundations of cost-sensitive learning. In *International Joint Conference on Artificial Intelligence*, volume 17, pages 973–978. Lawrence Erlbaum Associates Ltd, 2001.

[80] T. Elsken, J. H. Metzen, and F. Hutter. Efficient multi-objective neural architecture search via Lamarckian evolution. In *Proceedings of the International Conference on Learning Representations*, 2019.

[81] T. Elsken, J. H. Metzen, and F. Hutter. Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1):1997–2017, 2019.

[82] M. Emmerich, N. Beume, and B. Naujoks. An emo algorithm using the hypervolume measure as selection criterion. In *International Conference on Evolutionary Multi-Criterion Optimization*, pages 62–76. Springer, 2005.

[83] N. Erickson, J. Mueller, A. Shirkov, H. Zhang, P. Larroy, M. Li, and A. Smola. Autogluon-tabular: Robust and accurate AutoML for structured data. *arXiv:2003.06505*, 2020.

[84] A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542(7639):115–118, 2017.

[85] L. Faes, S. K. Wagner, D. J. Fu, X. Liu, E. Korot, J. R. Ledsam, T. Back, R. Chopra, N. Pontikos, C. Kern, et al. Automated deep learning design for medical image classification by health-care professionals with no coding experience: a feasibility study. *The Lancet Digital Health*, 1(5):e232–e242, 2019.

[86] R. Fakoor, J. W. Mueller, N. Erickson, P. Chaudhari, and A. J. Smola. Fast, accurate, and simple models for tabular data via augmented distillation. *Advances in Neural Information Processing Systems*, 33:8671–8681, 2020.

[87] S. Falkner, A. Klein, and F. Hutter. BOHB: robust and efficient hyperparameter optimization at scale. In J. G. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML*, volume 80 of *Proceedings of Machine Learning Research*, pages 1436–1445. PMLR, 2018.

[88] M. Feldman, S. A. Friedler, J. Moeller, C. Scheidegger, and S. Venkatasubramanian. Certifying and removing disparate impact. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '15*, pages 259–268, New York, New York, USA, 2015. ACM Press, 1412.3756.

[89] F. Ferraty and P. Vieu. *Nonparametric functional data analysis: theory and practice*, volume 76. Springer, 2006.

[90] M. Feurer, K. Eggensperger, S. Falkner, M. Lindauer, and F. Hutter. Practical automated machine learning for the AutoML challenge 2018. In *International Workshop on Automatic Machine Learning at ICML*, pages 1189–1232, 2018.

[91] M. Feurer, K. Eggensperger, S. Falkner, M. Lindauer, and F. Hutter. Auto-sklearn 2.0: Hands-free automl via meta-learning. *Journal of Machine Learning Research*, 23(261):1–61, 2022.

[92] M. Feurer, A. Klein, K. Eggensperger, J. T. Springenberg, M. Blum, and F. Hutter. Auto-sklearn: efficient and robust automated machine learning. In *Automated Machine Learning*, pages 113–134. Springer, Cham, 2019.

[93] R. R. Fletcher, A. Nakeshimana, and O. Olubeko. Addressing fairness, bias, and appropriate use of artificial intelligence and machine learning in global health. *Frontiers in Artificial Intelligence*, 3:116, 2021.

[94] H. Funk, C. Becker, A. Hofheinz, G. Xi, Y. Zhang, F. Pfisterer, M. Weigert, and M. Mittermeier. Towards an automated classification of Hess & Brezowsky's atmospheric circulation patterns Tief and Trog Mitteleuropa using deep learning methods.

In *Environmental Informatics – A bogeyman or saviour to achieve the UN Sustainable Development Goals? - Adjunct Proceedings of the 35th edition of the EnviroInfo – the long standing and established international and interdisciplinary conference series on leading environmental information and communication technologies*, pages 22–30. Shaker Verlag GmbH, 2021.

[95] J. Fürnkranz and E. Hüllermeier. *Preference learning*. Springer, 2010.

[96] A. Fuster, P. Goldsmith-Pinkham, T. Ramadorai, and A. Walther. Predictably unequal? the effects of machine learning on credit markets. *The Journal of Finance*, 77(1):5–47, 2022.

[97] I. Gabriel. Artificial intelligence, values, and alignment. *Minds and Machines*, 30(3):411–437, 2020.

[98] T. Gebru, J. Morgenstern, B. Vecchione, J. W. Vaughan, H. Wallach, H. D. Iii, and K. Crawford. Datasheets for datasets. *Communications of the ACM*, 64(12):86–92, 2021.

[99] Y. Geifman and R. El-Yaniv. Selective classification for deep neural networks. *Advances in Neural Information Processing Systems*, 30:4885–4894, 2017.

[100] P. Gijsbers, E. LeDell, J. Thomas, S. Poirier, B. Bischl, and J. Vanschoren. An open source AutoML benchmark. *arXiv:1907.00909*, 2019.

[101] P. Gijsbers, F. Pfisterer, J. N. van Rijn, B. Bischl, and J. Vanschoren. Meta-learning for symbolic hyperparameter defaults. In *2021 Genetic and Evolutionary Computation Conference Companion (GECCO '21 Companion)*, page 151–152, Lile, France, 2021. ACM.

[102] P. Gijsbers, F. Pfisterer, J. N. van Rijn, B. Bischl, and J. Vanschoren. Meta-learning for symbolic hyperparameter defaults, 2021, arXiv:2106.05767.

[103] P. Gijsbers, J. Vanschoren, and R. S. Olson. Layered tpot: Speeding up tree-based pipeline optimization. *arXiv:1801.06007*, 2018.

[104] P. A. Goff and K. B. Kahn. Racial bias in policing: Why we know less than we should. *Social Issues and Policy Review*, 6(1):177–210, 2012.

[105] J. Gonzalez, Z. Dai, P. Hennig, and N. Lawrence. Batch Bayesian Optimization via local penalization. In A. Gretton and C. C. Robert, editors, *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, volume 51 of *Proceedings of Machine Learning Research*, pages 648–657, Cadiz, Spain, 09–11 May 2016. PMLR.

[106] M. L. Gordon, K. Zhou, K. Patel, T. Hashimoto, and M. S. Bernstein. The disagreement deconvolution: Bringing machine learning performance metrics in line with reality. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–14, 2021.

[107] I. Gulrajani and D. Lopez-Paz. In search of lost domain generalization. In *International Conference on Learning Representations*, 2020.

[108] C. Guo and F. Berkhahn. Entity embeddings of categorical variables. *arXiv:1604.06737*, 2016.

[109] I. Guyon, L. Sun-Hosoya, M. Boullé, H. J. Escalante, S. Escalera, Z. Liu, D. Jajetic, B. Ray, M. Saeed, M. Sebag, et al. Analysis of the AutoML challenge series. *Automated Machine Learning*, page 177, 2019.

[110] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: an update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

[111] J. A. Hanley and B. J. McNeil. The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology*, 143(1):29–36, 1982.

[112] M. Hardt, E. Price, and N. Srebro. Equality of opportunity in supervised learning. *Advances in Neural Information Processing Systems*, 29:3323–3331, Dec. 2016, 1610.02413.

[113] U. Hebert-Johnson, M. P. Kim, O. Reingold, and G. Rothblum. Multicalibration: Calibration for the (Computationally-identifiable) masses. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1939–1948, Stockholmsmässan, Stockholm Sweden, 2018. PMLR.

[114] P. Hennig and C. J. Schuler. Entropy search for information-efficient global optimization. *Journal of Machine Learning Research*, 13(6), 2012.

[115] M. Hermann, F. Pfisterer, and F. Scheipl. A geometric framework for outlier detection in high-dimensional data. Manuscript submitted for publication.

[116] J. M. Hernández-Lobato, M. W. Hoffman, and Z. Ghahramani. Predictive entropy search for efficient global optimization of black-box functions. *Advances in Neural Information Processing Systems*, 27, 2014.

[117] S. Hochreiter, A. S. Younger, and P. R. Conwell. Learning to learn using gradient descent. In *International Conference on Artificial Neural Networks*, pages 87–94. Springer, 2001.

[118] G. Holmes, A. Donkin, and I. H. Witten. Weka: A machine learning workbench. In *Proceedings of ANZIIS'94-Australian New Zealnd Intelligent Information Systems Conference*, pages 357–361. IEEE, 1994.

[119] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In C. A. C. Coello, editor, *Learning and Intelligent Optimization*, pages 507–523, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[120] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*, pages 507–523. Springer, 2011.

[121] F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stützle. ParamILS: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36:267–306, 2009.

[122] F. Hutter, L. Kotthoff, and J. Vanschoren, editors. *Automated Machine Learning - Methods, Systems, Challenges*. Springer, 2019.

[123] K. Jamieson and A. Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. In *Artificial intelligence and statistics*, pages 240–248. PMLR, 2016.

[124] Y. Jin. *Multi-objective machine learning*, volume 16. Springer Science & Business Media, 2006.

[125] Y. Jin and B. Sendhoff. Pareto-based multiobjective machine learning: An overview and case studies. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(3):397–415, 2008.

[126] J. E. Johndrow and K. Lum. An algorithm for removing sensitive information: application to race-independent recidivism prediction. *The Annals of Applied Statistics*, 13(1):189–220, 2019.

[127] D. R. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

[128] F. Kamiran and T. Calders. Classification with no discrimination by preferential sampling. In *Proc. 19th Machine Learning Conf. Belgium and The Netherlands*, pages 1–6. Citeseer, 2010.

[129] F. Kamiran and T. Calders. Data preprocessing techniques for classification without discrimination. *Knowledge and Information Systems*, 33(1):1–33, 2012.

[130] T. Kamishima, S. Akaho, H. Asoh, and J. Sakuma. Fairness-aware classifier with prejudice remover regularizer. *Lecture Notes in Artificial Intelligence*, 7524(PART 2):35–50, 2012.

[131] K. Kandasamy, A. Krishnamurthy, J. Schneider, and B. Poczos. Parallelised bayesian optimisation via thompson sampling. In A. Storkey and F. Perez-Cruz, editors, *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*, volume 84 of *Proceedings of Machine Learning Research*, pages 133–142. PMLR, 09–11 Apr 2018.

[132] F. Karl, T. Pielok, J. Moosbauer, F. Pfisterer, S. Coors, M. Binder, L. Schneider, J. Thomas, J. Richter, M. Lang, E. C. Garrido-Merchán, J. Branke, and B. Bischl. Multi-objective hyperparameter optimization – an overview, 2022.

[133] Z. Karnin, T. Koren, and O. Somekh. Almost optimal exploration in multi-armed bandits. In *International Conference on Machine Learning*, pages 1238–1246. PMLR, 2013.

[134] N. Kilbertus, M. Rojas Carulla, G. Parascandolo, M. Hardt, D. Janzing, and B. Schölkopf. Avoiding discrimination through causal reasoning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[135] J. S. Kim, J. Chen, and A. Talwalkar. Model-agnostic characterization of fairness trade-offs. In *Proceedings of the International Conference on Machine Learning*, volume 37, pages 9339–9349, 2020.

[136] M. P. Kim, A. Ghorbani, and J. Zou. Multiaccuracy: Black-box post-processing for fairness in classification. In *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, AIES '19, pages 247–254, New York, NY, USA, 2019. Association for Computing Machinery.

[137] S. Kim, K. Choi, H.-S. Choi, B. Lee, and S. Yoon. Towards a rigorous evaluation of time-series anomaly detection, 2021.

[138] T.-Y. Kim and K. Leung. Forming and reacting to overall fairness: A cross-cultural comparison. *Organizational Behavior and Human Decision Processes*, 104(1):83–95, 2007.

[139] A. Klein, S. Falkner, J. T. Springenberg, and F. Hutter. Learning curve prediction with Bayesian neural networks. In *International Conference on Learning Representations*, 2017.

[140] J. Kleinberg, S. Mullainathan, and M. Raghavan. Inherent trade-offs in the fair determination of risk scores. In C. H. Papadimitriou, editor, *8th Innovations in Theoretical Computer Science Conference (ITCS 2017)*, volume 67 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 43:1–43:23. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[141] J. Knowles. ParEGO: A hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. *IEEE Transactions on Evolutionary Computation*, 10(1):50–66, 2006.

[142] J. Knowles, D. Corne, and A. Reynolds. Noisy multiobjective optimization on a budget of 250 evaluations. In M. Ehrgott, C. M. Fonseca, X. Gandibleux, J.-K. Hao, and M. Sevaux, editors, *Evolutionary Multi-Criterion Optimization*, pages 36–50, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[143] P. W. Koh, S. Sagawa, H. Marklund, S. M. Xie, M. Zhang, A. Balsubramani, W. Hu, M. Yasunaga, R. L. Phillips, I. Gao, et al. Wilds: A benchmark of in-the-wild distribution shifts. In *International Conference on Machine Learning*, pages 5637–5664. PMLR, 2021.

[144] J. R. Koza. Genetic programming as a means for programming computers by natural selection. *Statistics and computing*, 4(2):87–112, 1994.

[145] D. Kühn, P. Probst, J. Thomas, and B. Bischl. Automatic Exploration of Machine Learning Experiments on OpenML. *arXiv:1806.10961*, 2018.

[146] H. J. Kushner. A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. 1964.

[147] M. J. Kusner, J. R. Loftus, C. Russell, and R. Silva. Counterfactual fairness. *arXiv:1703.06856*, 2017.

[148] M. Lang, M. Binder, J. Richter, P. Schratz, F. Pfisterer, S. Coors, Q. Au, G. Casalicchio, L. Kotthoff, and B. Bischl. mlr3: A modern object-oriented machine learning framework in R. *Journal of Open Source Software*, dec 2019.

[149] A. J. Larrazabal, N. Nieto, V. Peterson, D. H. Milone, and E. Ferrante. Gender imbalance in medical imaging datasets produces biased classifiers for computer-aided diagnosis. *Proceedings of the National Academy of Sciences*, 117(23):12592–12594, 2020.

[150] N. Lavesson and P. Davidsson. Quantifying the impact of learning algorithm parameter tuning. In *Proc. of AAAI*, volume 6, pages 395–400, 2006.

[151] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[152] E. H. Lee, V. Perrone, C. Archambeau, and M. Seeger. Cost-aware Bayesian Optimization. *arXiv:2003.10870*, 2020.

[153] C. Li, Z. Yu, Y. Fu, Y. Zhang, Y. Zhao, H. You, Q. Yu, Y. Wang, C. Hao, and Y. Lin. Hw-nas-bench: Hardware-aware neural architecture search benchmark. In *International Conference on Learning Representations*, 2020.

[154] L. Li, K. G. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18:185:1–185:52, 2017.

[155] L. Li, K. G. Jamieson, A. Rostamizadeh, E. Gonina, J. Ben-tzur, M. Hardt, B. Recht, and A. Talwalkar. A system for massively parallel hyperparameter tuning. In I. S. Dhillon, D. S. Papailiopoulos, and V. Sze, editors, *Proceedings of Machine Learning and Systems 2020, MLSys 2020, Austin, TX, USA, March 2-4, 2020*. mlsys.org, 2020.

[156] M. Lindauer, H. H. Hoos, F. Hutter, and T. Schaub. Autofolio: An automatically configured algorithm selector. *Journal of Artificial Intelligence Research*, 53:745–778, 2015.

[157] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy. Progressive neural architecture search. In *Proceedings of the European conference on computer vision (ECCV)*, pages 19–34, 2018.

[158] H. Liu, K. Simonyan, and Y. Yang. DARTS: Differentiable architecture search. In *Proceedings of the International Conference on Learning Representations*, 2019.

[159] M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, and T. Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

[160] C. Louizos, K. Swersky, Y. Li, M. Welling, and R. Zemel. The variational fair autoencoder. *arXiv:1511.00830*, 2015.

[161] D. Madras, E. Creager, T. Pitassi, and R. Zemel. Learning adversarially fair and transferable representations. In *International Conference on Machine Learning*, pages 3384–3393. PMLR, 2018.

[162] A. Makarova, H. Shen, V. Perrone, A. Klein, J. B. Faddoul, A. Krause, M. Seeger, and C. Archambeau. Overfitting in Bayesian Optimization: an empirical study and early-stopping solution. *arXiv:2104.08166*, 2021.

[163] A. Makarova, H. Shen, V. Perrone, A. Klein, J. B. Faddoul, A. Krause, M. Seeger, and C. Archambeau. Automatic termination for hyperparameter optimization. In *First Conference on Automated Machine Learning*, 2022.

[164] M. Mansoury, H. Abdollahpouri, M. Pechenizkiy, B. Mobasher, and R. Burke. Feedback loop and bias amplification in recommender systems. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 2145–2148, 2020.

[165] N. Mehrabi, F. Morstatter, N. Saxena, K. Lerman, and A. Galstyan. A survey on bias and fairness in machine learning. 54(6), jul 2021.

[166] K. Miettinen. *Nonlinear Multiobjective Optimization.* International Series in Operations Research & Management Science. Springer US, 2012.

[167] I. Misra, A. Gupta, and M. Hebert. From red wine to red tomato: Composition with context. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1792–1801, 2017.

[168] S. Mitchell, E. Potash, S. Barocas, A. D'Amour, and K. Lum. Algorithmic fairness: Choices, assumptions, and definitions. *Annual Review of Statistics and Its Application*, 8:141–163, 2021.

[169] J. Močkus. On bayesian methods for seeking the extremum. In *Optimization techniques IFIP technical conference*, pages 400–404. Springer, 1975.

[170] F. Mohr, M. Wever, and E. Hüllermeier. Ml-plan: Automated machine learning via hierarchical planning. *Machine Learning*, 107(8):1495–1515, 2018.

[171] J. Moosbauer, M. Binder, L. Schneider, F. Pfisterer, M. Becker, M. Lang, L. Kotthoff, and B. Bischl. Automated benchmark-driven design and explanation of hyperparameter optimizers. *To appear in IEEE Transactions on Evolutionary Computation*, 2022.

[172] J. Moosbauer, J. Herbinger, G. Casalicchio, M. Lindauer, and B. Bischl. Explaining hyperparameter optimization via partial dependence plots. *Advances in Neural Information Processing Systems*, 34, 2021.

[173] P. Murray-Rust. Open data in science. *Nature Precedings*, pages 1756–0357, 2008.

[174] A. Narayanan. Translation tutorial: 21 fairness definitions and their politics. In *Proc. Conf. Fairness Accountability Transp., New York, USA*, volume 1170, page 3, 2018.

[175] V. Nguyen, S. Gupta, S. Rana, C. Li, and S. Venkatesh. Regret for expected improvement over the best-observed value and stopping condition. In *Asian Conference on Machine Learning*, pages 279–294. PMLR, 2017.

[176] A. Niculescu-Mizil and R. Caruana. Predicting good probabilities with supervised learning. In *Proceedings of the 22nd International Conference on Machine Learning*, ICML '05, page 625–632, New York, NY, USA, 2005. Association for Computing Machinery.

[177] C. Nießl, M. Herrmann, C. Wiedemann, G. Casalicchio, and A.-L. Boulesteix. Over-optimism in benchmark studies and the multiplicity of design and analysis options when interpreting their results. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 12(2):e1441, 2022.

[178] R. S. Olson and J. H. Moore. Tpot: A tree-based pipeline optimization tool for automating machine learning. In *Workshop on automatic machine learning*, pages 66–74. PMLR, 2016.

[179] M. O'Neill and C. Ryan. Grammatical evolution. *IEEE Transactions on Evolutionary Computation*, 5(4):349–358, 2001.

[180] F. Pargent, F. Pfisterer, J. Thomas, and B. Bischl. Regularized target encoding outperforms traditional methods in supervised machine learning with high cardinality features. *Computational Statistics*, pages 1–22, 2022.

[181] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. 2017.

[182] J. Pearl. Causal inference in statistics: An overview. *Statistics surveys*, 3:96–146, 2009.

[183] J. Pearl and D. Mackenzie. *The Book of Why: The new science of Cause and Effect.* Basic books, 2018.

[184] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[185] J. Petrak. Fast subsampling performance estimates for classification algorithm selection. In *Proceedings of the ECML-00 Workshop on Meta-Learning: Building Automatic Advice Strategies for Model Selection and Method Combination*, pages 3–14, 2000.

[186] F. Pfisterer, L. Beggel, X. Sun, F. Scheipl, and B. Bischl. Benchmarking time series classification – functional data vs machine learning approaches, 2019, arXiv:1911.07511.

[187] F. Pfisterer, S. Coors, J. Thomas, and B. Bischl. Multi-objective automatic machine learning with AutoxgboostMC. In *Automating Data Science Workshop at ECML*, 2019, arXiv:1908.10796.

[188] F. Pfisterer, C. Harbron, G. Jansen, and T. Xu. Evaluating domain generalization for survival analysis in clinical studies. In G. Flores, G. H. Chen, T. Pollard, J. C. Ho, and T. Naumann, editors, *Proceedings of the Conference on Health, Inference, and Learning*, volume 174 of *Proceedings of Machine Learning Research*, pages 32–47. PMLR, 07–08 Apr 2022.

[189] F. Pfisterer, C. Kern, S. Dandl, M. Sun, M. P. Kim, and B. Bischl. mcboost: Multi-calibration boosting for R. *Journal of Open Source Software*, 6(64):3453, 2021.

[190] F. Pfisterer, C. Kern, S. Dandl, M. Sun, M. P. Kim, and B. Bischl. mcboost: Multi-calibration boosting for R. *Journal of Open Source Software*, 6(64):3453, 2021.

[191] F. Pfisterer, J. Poon, and M. Lang. *mlr3keras: mlr3 Keras extension*, 2021. R package version 0.1.3.

[192] F. Pfisterer, L. Schneider, J. Moosbauer, M. Binder, and B. Bischl. Yahpo gym - an efficient multi-objective multi-fidelity benchmark for hyperparameter optimization. In I. Guyon, M. Lindauer, M. van der Schaar, F. Hutter, and R. Garnett, editors, *Proceedings of the First International Conference on Automated Machine Learning*, volume 188 of *Proceedings of Machine Learning Research*, pages 3/1–39. PMLR, 25–27 Jul 2022.

[193] F. Pfisterer, J. Thomas, and B. Bischl. Towards human centered AutoML, 2019, arXiv:1911.02391.

[194] F. Pfisterer, J. N. van Rijn, P. Probst, A. Müller, and B. Bischl. Learning multiple defaults for machine learning algorithms. *arXiv:1811.09409*, 2018.

[195] F. Pfisterer, J. N. van Rijn, P. Probst, A. Müller, and B. Bischl. Learning multiple defaults for machine learning algorithms. In *2021 Genetic and Evolutionary Computation Conference Companion (GECCO '21 Companion)*, page 241–242, Lile, France, 2021. ACM.

[196] F. Pfisterer, S. Wei, S. Vollmer, M. Lang, and B. Bischl. *Fairness Audits And Debiasing Using mlr3fairness*, Manuscript submitted for publication.

[197] H. Pham, M. Guan, B. Zoph, Q. V. Le, and J. Dean. Efficient neural architecture search via parameter sharing. In *Proceedings of the 35th International Conference on Machine Learning*, pages 4095–4104, 2018.

[198] G. Pleiss, M. Raghavan, F. Wu, J. Kleinberg, and K. Q. Weinberger. On fairness and calibration. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30, pages 5680–5689. Curran Associates, Red Hook, NY, 2017, 1709.02012.

[199] M. Poloczek, J. Wang, and P. Frazier. Multi-information source optimization. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[200] W. Ponweiser, T. Wagner, D. Biermann, and M. Vincze. Multiobjective optimization on a limited budget of evaluations using model-assisted $\mathcal{S}$-Metric selection. In *Parallel Problem Solving from Nature - PPSN X*, Lecture Notes in Computer Science, pages 784–794. Springer, Berlin, Heidelberg, 2008.

[201] P. Probst, A.-L. Boulesteix, and B. Bischl. Tunability: Importance of hyperparameters of machine learning algorithms. *The Journal of Machine Learning Research*, 20(53):1–32, 2019.

[202] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 4780–4789, 2019.

[203] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin. Large-scale evolution of image classifiers. In *International Conference on Machine Learning*, pages 2902–2911. PMLR, 2017.

[204] J. P. Robinson, G. Livitz, Y. Henon, C. Qin, Y. Fu, and S. Timoner. Face recognition: too bias, or not too bias? In *Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition workshops*, pages 0–1, 2020.

[205] K. T. Rodolfa, H. Lamba, and R. Ghani. Machine learning for public policy: Do we need to sacrifice accuracy to make models fair? *arXiv:2012.02972*, 2020.

[206] K. T. Rodolfa, H. Lamba, and R. Ghani. Empirical observation of negligible fairness–accuracy trade-offs in machine learning for public policy. *Nature Machine Intelligence*, 3(10):896–904, 2021.

[207] R. Roscher, B. Bohn, M. F. Duarte, and J. Garcke. Explainable machine learning for scientific insights and discoveries. *Ieee Access*, 8:42200–42216, 2020.

[208] D. Rügamer, C. Kolb, C. Fritz, F. Pfisterer, P. Kopper, B. Bischl, R. Shen, C. Bukas, L. B. de Andrade e Sousa, D. Thalmeier, P. Baumann, L. Kook, N. Klein, and C. L. Müller. deepregression: a flexible neural network framework for semi-structured deep distributional regression. *Journal of Statistical Software (provisionally accepted)*, 2022.

[209] D. Rügamer, F. Pfisterer, and B. Bischl. Neural mixture distributional regression. 2020, arXiv:2010.06889.

[210] P. Saleiro, B. Kuester, L. Hinkson, J. London, A. Stevens, A. Anisfeld, K. T. Rodolfa, and R. Ghani. Aequitas: A bias and fairness audit toolkit. *arXiv:1811.05577*, 2018.

[211] J. Schmidhuber. *Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook.* PhD thesis, Technische Universität München, 1987.

[212] R. M. Schmidt, F. Schneider, and P. Hennig. Descending through a crowded valley-benchmarking deep learning optimizers. In *International Conference on Machine Learning*, pages 9367–9376. PMLR, 2021.

[213] R. Schmucker, M. Donini, V. Perrone, M. B. Zafar, and C. Archambeau. Multi-objective multi-fidelity hyperparameter optimization with application to fairness. *NeurIPS Workshop on Meta-Learning*, 2020.

[214] L. Schneider, F. Pfisterer, M. Binder, and B. Bischl. Mutation is all you need. In *AutoML Workshop at ICML*, 2021, arXiv:2107.07343.

[215] L. Schneider, F. Pfisterer, P. Kent, J. Branke, B. Bischl, and J. Thomas. Tackling neural architecture search with quality diversity optimization. In I. Guyon, M. Lindauer, M. van der Schaar, F. Hutter, and R. Garnett, editors, *Proceedings of the First International Conference on Automated Machine Learning*, volume 188 of *Proceedings of Machine Learning Research*, pages 9/1–30. PMLR, 25–27 Jul 2022.

[216] L. Schneider, F. Pfisterer, J. Thomas, and B. Bischl. A collection of quality diversity optimization problems derived from hyperparameter optimization of machine learning models. In *GECCO '22: Proceedings of the Genetic and Evolutionary Computation Conference Companion*, Boston, United States of America, 2022. ACM.

[217] B. Schölkopf, A. J. Smola, F. Bach, et al. *Learning with kernels: Support Vector Machines, regularization, optimization, and beyond.* MIT press, 2002.

[218] D. Sculley, J. Snoek, A. Wiltschko, and A. Rahimi. Winner's curse? on pace, progress, and empirical rigor. 2018.

[219] A. D. Selbst, D. Boyd, S. A. Friedler, S. Venkatasubramanian, and J. Vertesi. Fairness and abstraction in sociotechnical systems. In *Proceedings of the conference on fairness, accountability, and transparency*, pages 59–68, 2019.

[220] G. Shafer and V. Vovk. A tutorial on conformal prediction. *Journal of Machine Learning Research*, 9(3), 2008.

[221] Z. R. Shi, C. Wang, and F. Fang. Artificial intelligence for social good: A survey. *arXiv:2001.01818*, 2020.

[222] M. Smith and S. Miller. The ethical application of biometric facial recognition technology. *Ai & Society*, 37(1):167–175, 2022.

[223] M. J. Smith, C. Sala, J. M. Kanter, and K. Veeramachaneni. The machine learning bazaar: Harnessing the ML ecosystem for effective system development. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 785–800, 2020.

[224] J. Snoek, H. Larochelle, and R. P. Adams. Practical Bayesian Optimization of machine learning algorithms. *Advances in Neural Information Processing Systems*, 25, 2012.

[225] M. Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society: Series B (Methodological)*, 36(2):111–133, 1974.

[226] X. Sun, A. Bommert, F. Pfisterer, J. Rahnenführer, M. Lang, and B. Bischl. High dimensional restrictive federated model selection with multi-objective Bayesian Optimization over shifted distributions. In Y. Bi, R. Bhatia, and S. Kapoor, editors, *Intelligent Systems and Applications*, pages 629–647, Cham, 2020. Springer International Publishing.

[227] H. Suresh and J. V. Guttag. A framework for understanding unintended consequences of machine learning. *arXiv:1901.10002*, 2, 2019.

[228] K. Swersky, J. Snoek, and R. P. Adams. Multi-task Bayesian Optimization. *Advances in Neural Information Processing Systems*, 26, 2013.

[229] R. C. Team et al. R: A language and environment for statistical computing. 2013.

[230] J. Thomas. Gradient boosting in automatic machine learning: feature selection and hyperparameter optimization, April 2019. PhD thesis, LMU Munich.

[231] J. Thomas, S. Coors, and B. Bischl. Automatic gradient boosting. In *International Workshop on Automatic Machine Learning at ICML*, 2018.

[232] R. Thomas and D. Uminsky. The problem with metrics is a fundamental problem for AI. *arXiv:2002.08512*, 2020.

[233] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 847–855, 2013.

[234] S. Thrun and L. Pratt. *Learning to learn*. Springer Science & Business Media, 2012.

[235] E. J. Topol. High-performance medicine: the convergence of human and artificial intelligence. *Nature medicine*, 25(1):44–56, 2019.

[236] T. Tornede, A. Tornede, J. Hanselle, M. Wever, F. Mohr, and E. Hüllermeier. Towards green automated machine learning: Status quo and future directions. *arXiv:2111.05850*, 2021.

[237] T. Tornede, A. Tornede, M. Wever, F. Mohr, and E. Hüllermeier. AutoML for predictive maintenance: One tool to rul them all. In *IoT Streams for Data-Driven Predictive Maintenance and IoT, Edge, and Mobile for Embedded Machine Learning*, pages 106–118. Springer, 2020.

[238] J. Traub, Z. Kaoudi, J.-A. Quiané-Ruiz, and V. Markl. Agora: Bringing together datasets, algorithms, models and more in a unified ecosystem [vision]. *ACM SIGMOD Record*, 49(4):6–11, 2021.

[239] M. Turner and M. McBurnett. Predictive models with explanatory concepts: a general framework for explaining machine learning credit risk models that simultaneously increases predictive power. In *Proceedings of the 15th Credit Scoring and Credit Control Conference*, 2019.

[240] B. Ustun, Y. Liu, and D. Parkes. Fairness without harm: Decoupled classifiers with preference guarantees. In *International Conference on Machine Learning*, pages 6373–6382. PMLR, 2019.

[241] J. N. van Rijn. *Massively Collaborative Machine Learning.* PhD thesis, Leiden University, 2016.

[242] J. N. van Rijn, F. Pfisterer, J. Thomas, B. Bischl, and J. Vanschoren. Meta learning for defaults–symbolic defaults. In *NeurIPS 2018 Workshop on Meta Learning*, 2018.

[243] G. Van Rossum and F. L. Drake Jr. *Python reference manual.* Centrum voor Wiskunde en Informatica Amsterdam, 1995.

[244] J. Vanschoren. Meta-learning: A survey. *arXiv:1810.03548*, 2018.

[245] J. Vanschoren, J. N. Van Rijn, B. Bischl, and L. Torgo. OpenML: networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 15(2):49–60, 2014.

[246] R. Vilalta and Y. Drissi. A perspective view and survey of meta-learning. *Artificial intelligence review*, 18(2):77–95, 2002.

[247] R. Vinuesa, H. Azizpour, I. Leite, M. Balaam, V. Dignum, S. Domisch, A. Felländer, S. D. Langhans, M. Tegmark, and F. F. Nerini. The role of artificial intelligence in achieving the sustainable development goals. *Nature communications*, 11(1):1–10, 2020.

[248] S. Walker, C. Spohn, and M. DeLone. Race, Ethnicity, and Crime in America, 2007.

[249] S. Wei and M. Niethammer. The fairness-accuracy pareto front. *arXiv:2008.10797*, 2020.

[250] C. White, W. Neiswanger, and Y. Savani. BANANAS: Bayesian optimization with neural architectures for neural architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021.

[251] J. Wilson, F. Hutter, and M. Deisenroth. Maximizing acquisition functions for Bayesian Optimization. *Advances in Neural Information Processing Systems*, 31, 2018.

[252] F. Winkelmolen, N. Ivkin, H. F. Bozkurt, and Z. Karnin. Practical and sample efficient zero-shot HPO. *arXiv:2007.13382*, 2020.

[253] M. Wistuba, N. Schilling, and L. Schmidt-Thieme. Learning hyperparameter optimization initializations. In *Proc. of DSAA*, pages 1–10. IEEE, 2015.

[254] D. H. Wolpert. Stacked generalization. *Neural networks*, 5(2):241–259, 1992.

[255] K. Woźnica, M. Grzyb, Z. Trafas, and P. Biecek. Consolidated learning–a domain-specific model-free optimization strategy with examples for xgboost and mimic-iv. *arXiv:2201.11815*, 2022.

[256] J. Wu and P. Frazier. The parallel knowledge gradient method for batch Bayesian Optimization. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[257] R. Wu and E. Keogh. Current time series anomaly detection benchmarks are flawed and are creating the illusion of progress. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1, 2021.

[258] H. Yoganarasimhan. Search personalization using machine learning. *Management Science*, 66(3):1045–1070, 2020.

[259] M. B. Zafar, I. Valera, M. Gomez Rodriguez, and K. P. Gummadi. Fairness beyond disparate treatment & disparate impact. In *Proceedings of the 26th International Conference on World Wide Web*, pages 1171–1180, Geneva, Switzerland, Apr. 2017. International World Wide Web Conferences Steering Committee.

[260] M. B. Zafar, I. Valera, M. Rodriguez, K. Gummadi, and A. Weller. From parity to preference-based notions of fairness in classification. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30, pages 229–239, Red Hook, NY, June 2017. Curran Associates, Inc., 1707.00010.

[261] B. H. Zhang, B. Lemoine, and M. Mitchell. Mitigating unwanted biases with adversarial learning. In *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society*, pages 335–340, 2018.

[262] J. Zhang and E. Bareinboim. Equality of opportunity in classification: A causal approach. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31, pages 3675–3685. Curran Associates, Inc., 2018.

[263] J. Zhang and E. Bareinboim. Fairness in decision-making—the causal explanation formula. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[264] P. Zhao, K. Xiao, Y. Zhang, K. Bian, and W. Yan. Ameir: Automatic behavior modeling, interaction exploration and mlp investigation in the recommender system. In Z.-H. Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 2104–2110. International Joint Conferences on Artificial Intelligence Organization, 8 2021.

[265] K. Zhou, Z. Liu, Y. Qiao, T. Xiang, and C. Change Loy. Domain generalization: A survey. *arXiv e-prints*, pages arXiv–2103, 2021.

[266] L. Zimmer, M. Lindauer, and F. Hutter. Auto-pytorch: Multi-fidelity metalearning for efficient and robust autodl. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[267] I. Zliobaite. On the relation between accuracy and fairness in binary classification. In *The 2nd workshop on Fairness, Accountability, and Transparency in Machine Learning (FATML) at ICML'15*, 2015.

[268] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. In *Proceedings of the International Conference on Learning Representations*, 2017.

[269] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.

[270] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8697–8710, 2018.

CHAPTER 4

CONTRIBUTIONS - AUTOML

4.1 Meta Learning for Defaults–Symbolic Defaults

Contributed Article:
J. N. van Rijn, F. Pfisterer, J. Thomas, B. Bischl, and J. Vanschoren. Meta learning for defaults–symbolic defaults. In *NeurIPS 2018 Workshop on Meta Learning*, 2018

Declaration of contributions The main idea was contributed by JvR, who also implemented the core method, with several adaptations resulting from discussions and advice from FP, JT, BB ans JV. FP and JT drafted large parts of the manuscript with revisions by JvR, BB and JV.

Meta Learning for Defaults – Symbolic Defaults

Jan N. van Rijn
Data Science Institute
Columbia University

Florian Pfisterer
Department of Statistics
LMU Munich

Janek Thomas
Department of Statistics
LMU Munich

Andreas Müller
Data Science Institute
Columbia University

Bernd Bischl
Department of Statistics
LMU Munich

Joaquin Vanschoren
Mathematics and Computer Science
Eindhoven University of Technology

Abstract

In this work we propose to use meta-learning to learn sets of *symbolic* default hyperparameter configurations that work well across many data sets. A well known example for such a symbolic default is the logarithmic relation between the number of features of a dataset and the available features per split of a Random Forest, as observed by Breiman (2001). Symbolic functions allow for a more rich vocabulary to define defaults on. In the past, symbolic and static default values have been obtained either from hand-crafted heuristics or empirical evaluations of specific algorithms. We propose to automatically learn such symbolic configurations, i.e., *formulas* containing meta-features, from a large set of prior evaluations of numeric hyperparameters on multiple data sets via symbolic regression and optimization.

1 Introduction

The performance of most machine learning algorithms is greatly influenced by their hyperparameter settings. Various methods exist to automatically optimize hyperparameters, including random search (Bergstra and Bengio, 2012), Bayesian optimization (Snoek et al., 2012; Hutter et al., 2011), meta-learning (Brazdil et al., 2008) and bandit-based methods (Li et al., 2017). Depending on the algorithm, proper tuning of hyperparameters can yield considerable performance gains (Lavesson and Davidsson, 2006). Despite the acknowledged importance of tuning hyperparameters, the additional run time, code complexity and experimental design questions cause many practitioners to leave many hyperparameters to their default values, especially in real-world machine learning pipelines containing many hyperparameters. Moreover, it seems less than ideal to optimize all hyperparameters from scratch with every new dataset. If the optimal values of an hyperparameter are functionally dependent on properties of the data, we could learn this functional relationship and express them as symbolic default configurations that work well across many data sets. That way we can transfer information from previous optimization runs to obtain better data set dependent defaults and good starting configurations for further tuning.

Some of these functional relationships are reported in the literature, such as is the logarithmic relation between the number of features of a dataset and the available features per split of a Random Forest Breiman (2001). Other examples are the interaction between the RBF kernel bandwidth parameter (gamma) in SVM's and the number of features (Vanschoren et al., 2012) or the median distance between observations (Caputo et al., 2002). Some of these are also implemented in machine-learning workbenches such as `sklearn` (Pedregosa et al., 2011), `weka` (Hall et al., 2009) or `mlr` (Bischl et al., 2016). It is often not clear and rarely reported how such relationships were discovered, nor does there seem to be a clear consensus between workbenches on which symbolic defaults to implement. Also, they are typically limited to a single hyperparameter, and don't take into account how multiple hyperparameters may interact.

Meta-learning approaches have been proposed to learn *static* defaults (Pfisterer et al., 2018; Probst et al., 2018; Weerts et al., 2018; Wistuba et al., 2015), to find which hyperparameters are most important to optimize (van Rijn and Hutter, 2018; Probst et al., 2018; Weerts et al., 2018), or to build meta-models to select the kernel or kernel width in SVMs (Soares et al., 2004; Valerio and Vilalta, 2014; Strang et al., 2018).

This paper addresses a new meta-learning challenge: "Can we *learn* sets of *symbolic* configurations for hyperparameters of state-of-the-art machine learning algorithms?". Contrary to static defaults, symbolic defaults should be a function of the meta-features of the data set at hand. Ideally, these meta-features are easily computed, so that the symbolic default configurations can be easily implemented into software frameworks with little to no computational overhead. We show that such symbolic defaults outperform the best overall static defaults, and propose techniques to learn such symbolic defaults via symbolic regression and optimization.

2 Problem definition

2.1 Preliminaries

Consider a target variable y, a feature vector X, and an unknown joint distribution P on (X, y), from which we have sampled a dataset $\mathcal{D}$ containing $|\mathcal{D}|$ observations. A machine learning (ML) algorithm tries to approximate the functional relationship between X and y by producing a prediction model $\hat{f}_\theta(X)$, controlled by a multi-dimensional hyperparameter configuration $\theta \in \Theta$ of length p: $\theta = \{\phi_1, \ldots, \phi_p\}$. In order to measure prediction performance pointwise between a true label y and its prediction $\hat{f}(X)$, we define a loss function $L(y, \hat{f}(X))$.
We are naturally interested in estimating the expected risk of the inducing algorithm, w.r.t. θ on new data, also sampled from $\mathcal{P}$: $R_\mathcal{P}(\theta) = E(L(y, \hat{f}(X))|\mathcal{P})$. Thus, $R_\mathcal{P}(\theta)$ quantifies the expected predictive performance associated with a hyperparameter configuration θ for a given data distribution, learning algorithm and performance measure. Given a data distribution, a learning algorithm and a performance measure, this mapping encodes the numerical quality for any hyperparameter configuration θ.

Given K different datasets (or data distributions) $\mathcal{P}_1, ..., \mathcal{P}_K$, we arrive at K hyperparameter risk mappings.

$$R_k(\theta) = E(L(y, \hat{f}(X, \theta))|\mathcal{P}_k), \qquad k = 1, ..., K.$$

2.2 Meta Data

Evaluations To learn symbolic defaults, we first gather meta-data that evaluates $R_k(\theta)$ on all K datasets. For a given fixed algorithm with hyperparameter space Θ and a performance measure, e.g., area under the ROC curve (AUC), a large number of experiments is run on datasets $P_1, \ldots, P_K$. These experiments can be generated by a simple random search, i.e., by sampling random hyperparameter configurations from Θ, and evaluating them via cross-validation.

Surrogate Models In principle, it is possible to estimate $R_k(\theta)$ empirically using cross-validation for every $\theta \in \Theta$. However, since each cross-validation involves training many models, this is costly if we want to obtaining results for a large number of configurations. Therefore, we propose to employ *surrogate models* that predict the outcome of a given performance measure and algorithm for a given hyperparameter configuration. We train one model for each dataset (and each algorithm) on a sufficiently large random sample of evaluations (Eggensperger et al., 2015). For this, we can also reuse evaluations shared on OpenML (Vanschoren et al., 2014). These surrogate models provides us with a fast approximate way to evaluate the performance of any given configuration, without the requirement of costly training and evaluating models for every possible configuration.

Considering the fact that performances on different datasets are usually not commensurable (Demšar, 2006), an appropriate scaling is required before training surrogate models to enable a comparison between datasets. This is done in literature by resorting to ranking (Bardenet et al., 2013), or scaling (Yogatama and Mann, 2014) to standard deviations from the mean. We mitigate the problem of lacking commensurability between datasets by normalizing performance results on a per-dataset

Table 1: Simple transformation functions, parameterized by a constant α and a meta-feature value x.

transformation	function
linear	$x \cdot \alpha$
square root	$\sqrt{x} \cdot \alpha$
logaritmic	$\log(x) \cdot \alpha$
inverse	α / x
exponentiation	x^{α}

basis. A drawback to this is that some information regarding the absolute performance of the algorithm and the spread across different configurations is lost.

Data set characteristics In addition to the performance of random hyperparameter-configurations, OpenML contains a range of dataset characteristics, i.e, meta-features. A full list of available characteristics is described by van Rijn (2016). These characteristics include (among many others) the number of observations, the number of features and information regarding class balance. We denote the set of characteristics $\{c_1, c_2, ..., c_L\}$ with C.

2.3 Hypothesis space

Finding an optimal default now corresponds to finding a configuration θ that minimizes the risk $R_k(\theta)$ across K datasets. We define the risk over K datasets $R(\theta) = \frac{1}{K} \sum_1^K R_k(\theta)$, i.e., aggregate over datasets using the mean.

We allow our configurations to be symbolic, i.e., contain formulas instead of static values. For this reason we define a set of transformations T that are functions of the data set's meta-features, and map from the values of these meta-features to a real value for a given numeric hyperparameter θ_i, thus $t(\mathbf{x}) : \mathbb{R} \rightarrow \mathbb{R}$. Table 1 shows a list of simple transformations from a single meta-feature x. Note that although these symbolic function have a parameter (denoted by α), the optimal value for this will be determined by the search procedure. Of course, many more complex transformations can be considered as well.

Note that not all possible combinations will map the input to sensible output ranges. For example, the exponential function may generate unreasonable high values for high values for α. We can either add additional constraints on the transformed values to map them back into a reasonable range, or constrain the search method at these functions to not consider them. In this work, we opted for the latter.

3 Exhaustive search results

To demonstrate the utility of symbolic defaults, we first perform an exhaustive search on the simplified hypothesis space shown in Table 1. Given these transformation functions, a set of meta-features and a set of constant values (for parameter α), we enumerate all possible symbolic functions for a given hyperparameter, evaluate them on a wide set of datasets, and select the optimal one. Also, instead of jointly learning symbolic defaults for all hyperparameters, we only allow one symbolic default at a time, and set all other hyperparameters to a static default such that for the configuration the risk across datasets is minimized.

Setup The experiment is based on datasets from the OpenML100 (Bischl et al., 2017) benchmark suite. A rbf-SVM is used and only the hyperparameters γ and C are optimized. We generate candidate transformations according to Table 1, using a numerical constant (α) geometrically increasing with 10 steps from 0.1 to 2, and 80+ meta-features available from OpenML. This allows for $4{,}000$ symbolic expressions per hyperparameter. The search procedure should select the best among these.

The evaluation is based on a leave-one-dataset-out strategy, where the (symbolic) defaults are computed based on all but one dataset, and compared to the best *vanilla* default values. The vanilla defaults were computed by doing a full grid search over the hyperparameter space (with 8 values per hyperparameter), using the corresponding surrogate model to predict the performance on a specific

Table 2: Comparison between vanilla defaults and symbolic defaults, on 98 datasets from the OpenML100 (Bischl et al., 2017). Full results are displayed in Table 3 in the appendix.

strategy	wins	symbolc default configuration
symbolic	59	$\gamma = 0.189824/\mathrm{NumberOfFeatures}$, $C = 86.13$
vanilla	36	$\gamma = 0.001078$, $C = 4522.35$

dataset. The best overall configuration across all datasets is the best vanilla default value. The meta-data used to train the surrogate models is the same as used by van Rijn and Hutter (2018).

Results The results of the experiment can be seen in Table 2, in the appendix. The OpenML100 consists of 100 datasets; for 2 datasets the meta-data was incomplete. Out of the 98 datasets on which the defaults were evaluated, the symbolic defaults outperform the vanilla defaults in 59 cases, lose in 36 and draws in 3 cases. In all cases, the found default configuration was consistent across all leave-one-out cross-validation folds, for both symbolic and vanilla defaults. The latter indicates that (i) the set of datasets is large enough to learn meaningful defaults on, and (ii) the learned defaults generalize over tasks. Moreover. since these results were obtained from a simplified search space, it is quite possible that even better symbolic defaults can be discovered, as well as configurations in which multiple hyperparameters have symbolic defaults.

Note that these findings are in line what was reported by Vanschoren et al. (2012), who stated that they could not find a direct correlation, but that high gamma values are predominantly performing well on datasets with a low number of features.

4 Outlook

The method detailed in the previous sections demonstrated the feasibility of learning a set of *simple* data dependent defaults. In future work we first of all plan to extend the search space: we want to find formulas not for a single hyperparameter, but instead for all sensible hyperparameters of an algorithm. The current experiment additionally introduces prior assumptions with regards to the kind of functions we are able to learn, and we currently limit our approach to transformations that contain a single meta-feature. In future work, we want to introduce fewer restrictions to the space of possible transformations. As such, we plan to include combinations of meta-features, as well as introduce a host of significantly more complex transformations. Allowing for more complex formulas thus reduces the amount of prior assumptions we have to introduce. This comes with a cost: it is no longer sensible, or depending on the search space impossible, to exhaustively search through the space of possible formulas, even when using a surrogate model. One possible approach to solve this is to represent the space of functions as a grammar in Backus-Naur form and represent generated formulas as integer vectors where each entry represents which element of the right side of the grammar rule to follow (O'Neill and Ryan (2001), Noorian et al. (2016)). Using this representation, more advanced techniques like genetic algorithms can also be used to search larger and more complex sets of transformation functions in a much more efficient way.

Multiple challenges with this approach still exist. A search across the space of all possible functions may result in invalid values, or values out of the valid range of the hyperparameter for specific datasets. This does not necessarily pose a problem for genetic algorithms, as a few valid formulas already suffice, but hampers the efficiency of the search procedure. Additionally, a concurrent search for optimal formulas of all hyperparameters of an algorithm is difficult, because obtaining a bad value for only a single hyperparameter ϕ out of the full configuration θ can result in a bad performance overall. We propose to solve this using a round-robin approach, where we repeatedly iterate over all parameters and only learn a formula for one hyperparameter at a time.

On the other hand, we hope to gain several insights that do not only advance the state of research, but also improve the performance and robustness of many widely used machine learning algorithms, and thus widely influence the quality of learned models for users who are not able to tune all model hyperparameters.

Acknowledgments This material is based upon work supported by the National Science Foundation under Grant No. 1740305 and by DARPA under Grant No. DARPA-BAA-16-51.

References

Bardenet, R., Brendel, M., Kégl, B., and Sebag, M. (2013). Collaborative hyperparameter tuning. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, pages II–199–II–207. JMLR.org.

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305.

Bischl, B., Casalicchio, G., Feurer, M., Hutter, F., Lang, M., Mantovani, R. G., van Rijn, J. N., and Vanschoren, J. (2017). OpenML Benchmarking Suites and the OpenML100. *arXiv preprint arXiv:1708.03731*.

Bischl, B., Lang, M., Kotthoff, L., Schiffner, J., Richter, J., Studerus, E., Casalicchio, G., and Jones, Z. M. (2016). mlr: Machine learning in R. *JMLR*, 17(170):1–5.

Brazdil, P., Giraud-Carrier, C., Soares, C., and Vilalta, R. (2008). *Metalearning: Applications to Data Mining*. Springer Publishing Company, Incorporated, 1 edition.

Breiman, L. (2001). Random forests. *Mach. Learn.*, 45(1):5–32.

Caputo, B., Sim, K., Furesjo, F., and Smola, A. (2002). Appearance-based object recognition using svms: which kernel should i use? In *Procceedings of NIPS workshop on Statitsical methods for computational experiments in visual processing and computer vision, Whistler*, pages 1–10.

Demšar, J. (2006). Statistical Comparisons of Classifiers over Multiple Data Sets. *The Journal of Machine Learning Research*, 7:1–30.

Eggensperger, K., Hutter, F., Hoos, H., and Leyton-Brown, K. (2015). Efficient benchmarking of hyperparameter optimizers via surrogates. In *Proc. of AAAI 2015*, pages 1114–1120.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. (2009). The WEKA data mining software: an update. *SIGKDD Explorations*, 11(1):10–18.

Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization*, pages 507–523. Springer.

Lavesson, N. and Davidsson, P. (2006). Quantifying the impact of learning algorithm parameter tuning. In *AAAI*, volume 6, pages 395–400.

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. (2017). Hyperband: Bandit-Based Configuration Evaluation for Hyperparameter Optimization. In *Proc. of ICLR 2017*.

Noorian, F., de Silva, A. M., Leong, P. H., et al. (2016). gramevol: Grammatical evolution in r. *Journal of Statistical Software*, 71(i01).

O'Neill, M. and Ryan, C. (2001). Grammatical evolution. *IEEE Transactions on Evolutionary Computation*, 5(4):349–358.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Pfisterer, F., van Rijn, J. N., Probst, P., Müller, A. M., and Bischl, B. (2018). Learning Multiple Defaults for Machine Learning Algorithms. *arXiv preprint arXiv:1811.09409*.

Probst, P., Bischl, B., and Boulesteix, A. (2018). Tunability: Importance of hyperparameters of machine learning algorithms. *arXiv preprint arXiv:1802.09596*.

Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'12, pages 2951–2959, USA. Curran Associates Inc.

Soares, C., Brazdil, P., and Kuba, P. (2004). A meta-learning method to select the kernel width in support vector regression. *Mach. Learn.*, 54:195–209.

Strang, B., van der Putten, P., van Rijn, J. N., and Hutter, F. (2018). Don't rule out simple models prematurely: A large scale benchmark comparing linear and non-linear classifiers in openml. In *International Symposium on Intelligent Data Analysis*, pages 303–315. Springer.

Valerio, R. and Vilalta, R. (2014). Kernel selection in support vector machines using gram-matrix properties. In *NIPS Workshop on Modern Nonparametrics: Automating the Learning Pipeline*, volume 14.

van Rijn, J. N. (2016). *Massively Collaborative Machine Learning*. PhD thesis, Leiden University.

van Rijn, J. N. and Hutter, F. (2018). Hyperparameter importance across datasets. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2367–2376. ACM.

Vanschoren, J., Blockeel, H., Pfahringer, B., and Holmes, G. (2012). Experiment databases. *Machine Learning*, 87(2):127–158.

Vanschoren, J., van Rijn, J. N., Bischl, B., and Torgo, L. (2014). OpenML: networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 15(2):49–60.

Weerts, H., Meuller, M., and Vanschoren, J. (2018). Importance of tuning hyperparameters of machine learning algorithms. Technical report, TU Eindhoven.

Wistuba, M., Schilling, N., and Schmidt-Thieme, L. (2015). Learning hyperparameter optimization initializations. In *Data Science and Advanced Analytics (DSAA), 2015. 36678 2015. IEEE International Conference on*, pages 1–10. IEEE.

Yogatama, D. and Mann, G. (2014). Efficient Transfer Learning Method for Automatic Hyperparameter Tuning. In Kaski, S. and Corander, J., editors, *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*, volume 33 of *Proceedings of Machine Learning Research*, pages 1077–1085, Reykjavik, Iceland. PMLR.

A Results per dataset

Table 3: The difference between symbolic defaults and vanilla defaults, for 98 datasets from the OpenML100 (Bischl et al., 2017). The best found symbolic default was '$\gamma = 0.189824/\text{NumberOfFeatures}$, $C = 86.13$', consistent across all tasks. The best found vanilla default was '$\gamma = 0.001078$, $C = 4522.35$', consistent across all tasks.

dataset	symbolic	vanilla
kr-vs-kp	**0.993986**	0.993256
letter	**0.962879**	0.913438
balance-scale	**0.948774**	0.915973
mfeat-factors	0.997986	**0.998242**
mfeat-fourier	**0.994684**	0.991689
breast-w	0.878953	**0.936559**
mfeat-karhunen	**0.993776**	0.989547
mfeat-morphological	0.974419	**0.976140**
mfeat-pixel	**0.993166**	0.982842
car	**0.972400**	0.971796
mfeat-zernike	0.968154	**0.976384**
cmc	0.740027	**0.864494**
mushroom	0.999673	**0.999885**
optdigits	**0.998058**	0.993982
credit-approval	0.905408	**0.951653**
credit-g	**0.771748**	0.682432
pendigits	**0.995317**	0.989628
segment	0.976898	**0.991487**
diabetes	0.835865	**0.874568**
soybean	**0.991540**	0.987747
spambase	**0.993783**	0.982050
splice	**0.993554**	0.976631
tic-tac-toe	0.980773	**0.989053**
vehicle	**0.952675**	0.921425
waveform-5000	**0.959832**	0.922108
electricity	**0.499086**	0.440656
satimage	**0.959691**	0.958328
eucalyptus	0.865020	**0.950900**
sick	**0.968849**	0.927118
vowel	**0.997565**	0.967072
isolet	0.996529	**0.999196**
scene	**0.945410**	0.923344
monks-problems-1	**0.999999**	0.978106
monks-problems-2	**0.997830**	0.888671
monks-problems-3	0.993750	**0.997656**
JapaneseVowels	0.956405	**0.958125**
synthetic_control	**0.988586**	0.986362
irish	0.999619	**1.000000**
analcatdata_authorship	0.997976	**0.998362**
analcatdata_dmft	**0.689489**	0.681528
profb	0.631895	**0.931988**
collins	1.000000	1.000000
mnist_784	0.869533	**0.995322**
sylva_agnostic	**0.974975**	0.958614
gina_agnostic	0.949198	**0.994346**
ada_agnostic	**0.939905**	0.915434
mozilla4	**0.799870**	0.696705
pc4	**0.973807**	0.939981
pc3	**0.987253**	0.987021
jm1	**0.519439**	0.411916
kc2	**0.717854**	0.453129
kc1	**0.800003**	0.477086
pc1	**0.411465**	0.367177
KDDCup09_upselling	0.988281	0.988281
MagicTelescope	**0.947935**	0.839991
adult	0.879703	**0.960864**
wilt	**0.998123**	0.988539
wdbc	**0.986231**	0.959469
micro-mass	**0.960787**	0.837078
phoneme	**0.636225**	0.422257
one-hundred-plants-margin	**0.986408**	0.984546
one-hundred-plants-shape	0.935152	**0.937328**
one-hundred-plants-texture	**0.988585**	0.982143
qsar-biodeg	**0.953944**	0.949305
wall-robot-navigation	**0.965796**	0.918964
semeion	0.986686	**0.989221**
steel-plates-fault	0.998367	**0.999982**
tamilnadu-electricity	1.000000	1.000000
hill-valley	0.442016	**0.766961**
ilpd	**0.907537**	0.898711
madelon	0.859188	**0.908491**
nomao	**0.998656**	0.990892
ozone-level-8hr	**0.920647**	0.832482
cardiotocography	0.998393	**0.999983**
climate-model-simulation-crashes	**0.917094**	0.901476
cnae-9	**0.989397**	0.930784
eeg-eye-state	0.341995	**0.376861**
first-order-theorem-proving	0.635427	**0.683760**
gas-drift	**0.998310**	0.997685
banknote-authentication	**1.000000**	0.978950
blood-transfusion-service-center	**0.801340**	0.734690
artificial-characters	**0.562517**	0.473162
bank-marketing	0.767096	**0.902828**
Bioresponse	**0.936898**	0.882554
cjs	0.970188	**0.994433**
cylinder-bands	**0.943682**	0.914568
GesturePhaseSegmentationProcessed	**0.748805**	0.652932
har	0.996902	**0.999590**
PhishingWebsites	**0.937853**	0.914617
MiceProtein	0.788341	**0.858647**
Amazon_employee_access	**0.550310**	0.334129
dresses-sales	0.779151	**0.788013**
LED-display-domain-7digit	0.946904	**0.967014**
texture	0.998820	**0.999320**
Australian	0.927873	**0.953403**
connect-4	**0.950426**	0.940032
higgs	**0.954954**	0.844238
SpeedDating	**0.736372**	0.724337

4.2 Meta-Learning for Symbolic Hyperparameter Defaults

Contributed Article:
P. Gijsbers, F. Pfisterer, J. N. van Rijn, B. Bischl, and J. Vanschoren. Meta-learning for symbolic hyperparameter defaults. In *2021 Genetic and Evolutionary Computation Conference Companion (GECCO '21 Companion)*, page 151–152, Lile, France, 2021. ACM

This chapter includes the short-form publication [101]. The interested reader is refered to the long-form version of the article [102] [1].

Declaration of contributions PG and FP contributed equally. The main method was jointly developed by PG and FP based on advice from their co-authors and previous work in [242] by JvR and FP. PG and FP built the implementation, carried out the experiments, and wrote the initial manuscript. JvR, BB and JV advised throughout the project and further improved the manuscript.

[1] `https://arxiv.org/abs/2106.05767`

Meta-Learning for Symbolic Hyperparameter Defaults

Pieter Gijsbers*
University of Eindhoven
Eindhoven, Netherlands

Florian Pfisterer*
Ludwig-Maximilians-University
Munich, Germany

Jan N. van Rijn
LIACS, Leiden University
Leiden, Netherlands

Bernd Bischl
Ludwig-Maximilians-University
Munich, Germany

Joaquin Vanschoren
University of Eindhoven
Eindhoven, Netherlands

ABSTRACT

Hyperparameter optimization in machine learning (ML) deals with the problem of empirically learning an optimal algorithm configuration from data, usually formulated as a black-box optimization problem. In this work, we propose a zero-shot method to meta-learn symbolic default hyperparameter configurations that are expressed in terms of the properties of the dataset. This enables a much faster, but still data-dependent, configuration of the ML algorithm, compared to standard hyperparameter optimization approaches. In the past, symbolic and static default values have usually been obtained as hand-crafted heuristics. We propose an approach of learning such symbolic configurations as formulas of dataset properties from a large set of prior evaluations on multiple datasets by optimizing over a grammar of expressions using an evolutionary algorithm. We evaluate our method on surrogate empirical performance models as well as on real data across 6 ML algorithms on more than 100 datasets and demonstrate that our method indeed finds viable symbolic defaults.

CCS CONCEPTS

• **Computing methodologies → Supervised learning by classification**.

KEYWORDS

Hyperparameter Optimization, Metalearning

ACM Reference Format:
Pieter Gijsbers, Florian Pfisterer, Jan N. van Rijn, Bernd Bischl, and Joaquin Vanschoren. 2021. Meta-Learning for Symbolic Hyperparameter Defaults. In *2021 Genetic and Evolutionary Computation Conference Companion (GECCO '21 Companion), July 10–14, 2021, Lille, France.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3449726.3459532

1 INTRODUCTION & RELATED WORK

The performance of most machine learning (ML) algorithms is greatly influenced by their hyperparameter settings. While various methods exist to automatically optimize them, the additional complexity and effort cause many practitioners to forgo optimization. Defaults provide a fallback but are often static and do not take properties of the dataset into account, even though the success of tuning hyperparameters suggests values should be based on properties of the data. Contrary to static defaults, *symbolic defaults* should be a function of the meta-features (dataset characteristics) of the dataset, such as the number of features. A well-known example for a symbolic default is the random forest algorithm's default $mtry = \sqrt{p}$ for the number of features sampled in each split. In this paper we explore how such formulas can be obtained in a principled, empirical manner, especially when multiple hyperparameters interact, and have to be considered simultaneously[1]. We propose to learn such symbolic default configurations by optimizing over a grammar of potential expressions, in a manner similar to symbolic regression [3] using evolutionary algorithms. We validate our approach across a variety of state-of-the-art ML algorithms and propose default candidates for use by practitioners. The proposed approach is general and can be used for any algorithm as long as their performance is empirically measurable on instances in a similar manner.

2 METHOD

A symbolic configuration is a set of functions, one for each hyperparameter of the algorithm. Each function maps the meta-features of the given dataset to a value for a hyperparameter, e.g. $mtry = \sqrt{p}$. Note that it is not needed for any or all meta-features to be used in the mapping, i.e. the function may be constant (static) or only use few meta-features. We want to learn a symbolic default configuration $\lambda(.)$ for algorithm $\mathcal{A}$ that minimizes the expected risk induced by the model produced by $\mathcal{A}_{\lambda}$ across datasets.

We define a context-free grammar of transformations, which define the space of potential expressions for all functions $\lambda(.)$. We select a small set of simple dataset characteristics for use in formulas, e.g. number of observations, features, or missing values. Given K datasets, a risk function $R(\lambda(.), \mathcal{D}_i)$ that denotes the risk induced by the model learnt using algorithm $\mathcal{A}$ with symbolic configuration $\lambda(.)$ on dataset $\mathcal{D}_i$, we can formulate a global objective to minimize: $R_{\mathcal{D}}(\lambda(.)) = \frac{1}{K}\sum_{i=1}^{K} R(\lambda(.), \mathcal{D}_i)$. As estimating $R_{\mathcal{D}}(\lambda(.))$ empirically using cross-validation (CV) is costly in practice, we instead employ *surrogate models* that approximate $R_{\mathcal{D}}(\lambda(.))$.

Meta-learning. To create surrogate models, we collect data about the performance of randomly sampled constant configurations. These configurations are evaluated across all datasets using 10-fold CV. For each dataset we train a random forest model mapping hyperparameter configurations to expected performance. We can then approximate the average risk of $\lambda(.)$ by querying each surrogate

*Both authors contributed equally to the paper

[1]Code available at https://github.com/PGijsbers/symbolicdefaults

model after first computing the real configuration values using the dataset's characteristics.

Optimization. The problem we aim to solve requires optimization over a space of mathematical expressions. Several options to achieve this exist [4, 5]. We opt for a tree representation of individuals, where nodes correspond to operations and leaves to terminal symbols or numeric constants, and optimize this via genetic programming [3]. We differentiate between real-valued and integer-valued terminal symbols to account for the difference in algorithm hyperparameters. We use a $\mu + \lambda$ algorithm to evolve candidate solutions via crossover and mutation. We jointly optimize performance of solutions while preferring formulas with smaller structural depth using NSGA-II selection [1] without explicitly limiting length of the expressions.

3 RESULTS

We investigate symbolic defaults for 6 ML algorithms using a large set of meta-data, containing evaluations of over a hundred datasets available from OpenML [6]. We optimize the average logistic loss (normalized to [0,1]), but our methodology trivially extends to other performance measures. We evaluate using a leave-one-dataset-out strategy to obtain symbolic defaults. As baselines, we employ *random search* and *1-nearest neighbour*, an approach that selects the configuration that worked best on the most similar dataset, comparable to warm-starting in auto-sklearn [2].

Table 1 shows the mean and standard deviation of the normalized log-loss for each algorithm across all tasks, as predicted by surrogate models. The symbolic and constant columns denote the performance of defaults found with our approach including and excluding symbolic terminals respectively. The package column shows the best result obtained from either the scikit-learn or mlr default, and the last column denotes the best-found performance sampling 8 random real-world scores on the task for the algorithm. Note that the best rank can deviate from the best average performance.

The default mean rank is never significantly lower than that of other approaches, but in some cases, it is significantly higher. The only implementation default which does not score a significantly lower mean rank than our approach is the default for SVM, which has carefully hand-crafted defaults. For more nuance about the performance differences per dataset, Figure 1 shows the predicted normalized log-loss per dataset for SVM configurations obtained by different methods.

We further show the non-normalized log-loss per dataset obtained with 10-fold CV experiments in Figure 2. The median performance for symbolic defaults is slightly lower, though overall very similar performance is achieved by this automatically obtained symbolic default to the hand-crafted one in scikit-learn, or per-dataset recommendations from 1NN.

ACKNOWLEDGMENTS

This material is based upon work supported by the Data Driven Discovery of Models (D3M) program run by DARPA and the Air Force Research Laboratory, and by the German Federal Ministry of Education and Research (BMBF) under Grant No. 01IS18036A. The authors of this work take full responsibility for its content.

algorithm	symbolic	constant	package	opt. RS 8
glmnet	**0.917(.168)**	**0.928(.158)**	0.857(.154)	0.906(.080)
knn	**0.954(.148)**	0.947(.156)	0.879(.137)	**0.995(.009)**
rf	**0.946(.087)**	0.951(.074)	0.933(.085)	0.945(.078)
rpart	**0.922(.112)**	**0.925(.093)**	0.792(.141)	**0.932(.082)**
svm	**0.889(.178)**	**0.860(.207)**	**0.882(.190)**	**0.925(.084)**
xgboost	**0.995(.011)**	**0.995(.011)**	0.925(.125)	0.978(.043)

Table 1: Mean normalized log-loss (standard deviation) across all tasks with baselines. Boldface values indicate the average rank was not significantly worse than the best (underlined) of the four settings.

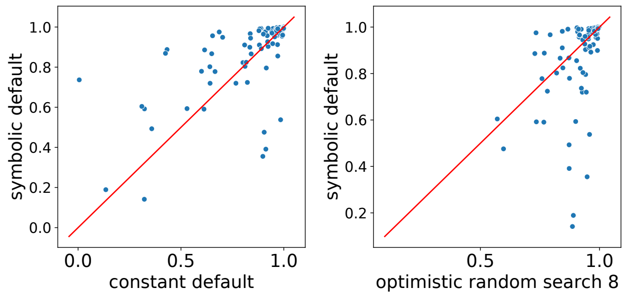


Figure 1: Normalized log-loss comparison of symbolic defaults to constant defaults (left) and budget 8 random search (right).

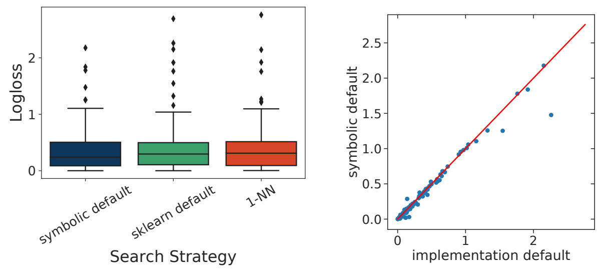


Figure 2: Comparison of 1NN, symbolic, and implementation default using log-loss across all datasets performed on real data.

REFERENCES

[1] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation* 6, 2 (2002), 182–197.
[2] M. Feurer, A. Klein, K. Eggensperger, J. T. Springenberg, M. Blum, and F. Hutter. 2015. Efficient and Robust Automated Machine Learning. In *Advances in Neural Information Processing Systems 28*. Curran Associates, Inc., 2962–2970.
[3] John R Koza. 1994. Genetic programming as a means for programming computers by natural selection. *Statistics and computing* 4, 2 (1994), 87–112.
[4] M. O'Neill and C. Ryan. 2001. Grammatical evolution. *IEEE Transactions on Evolutionary Computation* 5, 4 (Aug 2001), 349–358.
[5] Jan N. van Rijn, Florian Pfisterer, Janek Thomas, Andreas Müller, Bernd Bischl, and Joaquin Vanschoren. 2018. Meta learning for defaults : symbolic defaults. In *Workshop on Meta-Learning @ NeurIPS2018*.
[6] J. Vanschoren, J. N. van Rijn, B. Bischl, and L. Torgo. 2014. OpenML: networked science in machine learning. *ACM SIGKDD Explorations Newsletter* 15, 2 (2014), 49–60.

4.3 Learning Multiple Defaults for Machine Learning Algorithms

Contributed Article:
F. Pfisterer, J. N. van Rijn, P. Probst, A. Müller, and B. Bischl. Learning multiple defaults for machine learning algorithms. In *2021 Genetic and Evolutionary Computation Conference Companion (GECCO '21 Companion)*, page 241–242, Lile, France, 2021. ACM

This chapter includes the short-form publication in [195]. The interested reader is refered to the long-form version of the article in [194] [2].

Declaration of contributions The core idea to develop sets of defaults was proposed by AM, JvR and BB. FP implemented the method based on previous code originally developed by PP. FP furthermore conducted the experimental evaluation of the method, with JvR contributing the exhaustive search baseline. FP and JvR developed the manuscript with refinement and additional input from AM, BB and PP. The theoretical formulation of multiple defaults as a *maximum coverage problem* was contributed by JvR.

[2]`https://arxiv.org/abs/1811.09409`

Learning Multiple Defaults for Machine Learning Algorithms

Florian Pfisterer
Ludwig-Maximilians-University
Munich, Germany

Jan N. van Rijn
LIACS, Leiden University
Leiden, Netherlands

Philipp Probst
Benediktbeuern, Germany

Andreas C. Müller
Microsoft
Sunnyvale, U.S.A.

Bernd Bischl
Ludwig-Maximilians-University
Munich, Germany

ABSTRACT

Modern machine learning methods highly depend on their hyperparameter configurations for optimal performance. A widely used approach to selecting a configuration is using default settings, often proposed along with the publication of a new algorithm. Those default values are usually chosen in an ad-hoc manner to work on a wide variety of datasets. Different automatic hyperparameter configuration algorithms which select an optimal configuration per dataset have been proposed, but despite its importance, tuning is often skipped in applications because of additional run time, complexity, and experimental design questions. Instead, the learner is often applied in its defaults. This principled approach usually improves performance but adds additional algorithmic complexity and computational costs to the training procedure. We propose and study using a set of complementary default values, learned from a large database of prior empirical results as an alternative. Selecting an appropriate configuration on a new dataset then requires only a simple, efficient, and embarrassingly parallel search over this set. To demonstrate the effectiveness and efficiency of the approach, we compare learned sets of configurations to random search and Bayesian optimization. We show that sets of defaults can improve performance while being easy to deploy in comparison to more complex methods.

CCS CONCEPTS

• **Computing methodologies** → **Supervised learning by classification**.

KEYWORDS

AutoML, Hyperparameter Optimization, Metalearning

ACM Reference Format:
Florian Pfisterer, Jan N. van Rijn, Philipp Probst, Andreas C. Müller, and Bernd Bischl. 2021. Learning Multiple Defaults for Machine Learning Algorithms. In *2021 Genetic and Evolutionary Computation Conference Companion (GECCO '21 Companion), July 10–14, 2021, Lille, France.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3449726.3459523

1 INTRODUCTION

Hyperparameter settings for machine learning algorithms are often optimized via hyperparameter optimization e.g. using random search, Bayesian optimization, or meta learning. While not tuning parameters at all can be detrimental, defaults provide a simple and fast fall-back, that is easy to implement and use while providing strong anytime performance. We describe a general, learner-agnostic procedure, to (meta-)learn not one, but a (sequential) list of default configurations, which complement each other. These sets are ordered so that the earlier elements in the sequence provide greater benefits on average.[1] While traditional optimization methods are to be preferred when time and expertise are available, we conjecture that sets of defaults work well across a large variety of datasets. We leverage a large set of historic performance results of prior experiments that are available on OpenML [4]. Several approaches attempt to combine the paradigms of meta-learning and hyperparameter optimization, for example by warm starting hyperparameter optimization methods [2, 5]. While all these methods yield convincing results, they are by no means easy to deploy. Similar to our work, Wistuba et al. [6] learn a set of defaults from a fixed grid of evaluations, requiring hyperparameters evaluated on a grid across several datasets scaling exponentially with hyperparameter dimensionality. This is practically infeasible when there are large numbers of hyperparameters.

2 METHOD

Consider a target variable y, a feature vector $\boldsymbol{x}$, and an unknown joint distribution P on $(\boldsymbol{x}, y)$, from which we have sampled a i.i.d dataset $\mathcal{D}$. A machine learning *algorithm* $\mathcal{A}_{\boldsymbol{\lambda}}(\mathcal{D})$ learns a prediction model $\hat{f}(\boldsymbol{x})$. $\mathcal{A}_{\boldsymbol{\lambda}}$ is controlled by a multi-dimensional hyperparameter configuration $\boldsymbol{\lambda} \in \Lambda$ of length D, where Λ_j is usually a bounded real or integer interval, or a finite set of categorical values. We are interested in estimating the expected risk of the inducing algorithm w.r.t. $\boldsymbol{\lambda}$ on new data, also sampled from $\mathcal{P}$:$R_{\mathcal{P}}(\boldsymbol{\lambda}) = E_{\mathcal{P}}(L(y, \mathcal{A}_{\boldsymbol{\lambda}}(\mathcal{D})(\boldsymbol{x})))$, where the expectation above is taken over all data sets $\mathcal{D}$ from $\mathcal{P}$ and the test observation $(\boldsymbol{x}, y)$. Thus, $R_{\mathcal{P}}(\boldsymbol{\lambda})$ quantifies the expected predictive performance associated with a hyperparameter configuration $\boldsymbol{\lambda}$ for a given data distribution, learning algorithm and performance measure. In practice, given K different data sets we define K hyperparameter risk mappings: $R_k(\boldsymbol{\lambda}) = E_{\mathcal{P}_k}(L(y, \mathcal{A}_{\boldsymbol{\lambda}}(\mathcal{D})(\boldsymbol{x})))$ and the average risk of $\boldsymbol{\lambda}$ over K data sets: $R(\boldsymbol{\lambda}) = \frac{1}{K}\sum_{k=1}^{K} R_k(\boldsymbol{\lambda})$. Our goal now is to find a fixed-size set Λ_{def} of size T, that works well over a variety of datasets, in the sense that for each dataset $\mathcal{D}$, Λ_{def} contains at least one configuration that works well on $\mathcal{D}$. The risk of a set of configurations Λ_{def} of size T, aggregation function h (e.g.

[1]Full version of this article: https://arxiv.org/abs/1811.09409

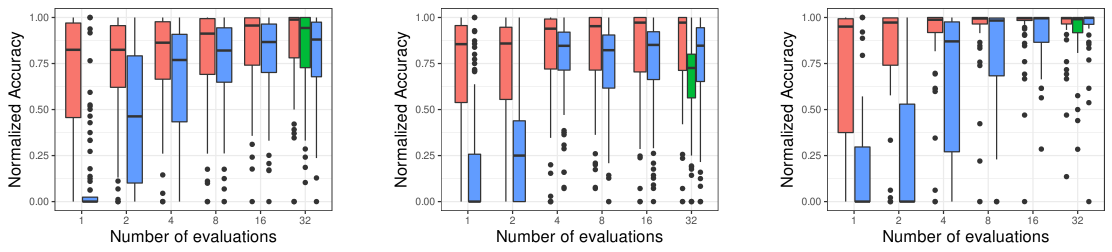


Figure 1: *Defaults* (red), *random search* (blue) and *Bayesian optimization* (green) across several budgets for Adaboost (left), Random Forest (middle) and SVM (right)

mean) and datasets $1, \dots, K$ is then given by:

$$G(\Lambda_{\text{def}}) = h\left(\min_{t=1,\dots,T} R_1(\boldsymbol{\lambda}_t), \dots, \min_{t=1,\dots,T} R_K(\boldsymbol{\lambda}_t))\right)$$

Finding an optimal subset Λ_{def} defines a (meta)-learning problem, that can be solved exactly or using a greedy approximation.

The exact version can be formulated as an instance of Mixed Integer Programming. In order to obtain a set of n defaults, the goal is to minimize

$$\sum_{k=1}^{K}\sum_{m=1}^{M} \Psi_{k,m} \cdot R_k(\boldsymbol{\lambda}_m) \tag{1}$$

subject to

$$\sum_{m=1}^{M} \phi_m = n \qquad \forall k : \forall m : \Psi_{k,m} \geq 0$$

$$\forall k : \forall m : \Psi_{k,m} \geq \phi_m - \sum_{s \in Q(k,m)} \phi_s \qquad \forall k : \sum_{m=1}^{M} \Psi_{k,m} = 1$$

After the optimization procedure, element $\Psi_{k,m}$ will be 1 if and only if configuration Λ_m has the lowest risk on distribution i out of all the configurations that are in the set of defaults. ϕ_m is an auxiliary variable. Since the exact solution is computationally prohibitively expensive, we adopt a greedy procedure for t = $1, \dots, T$:

$$\boldsymbol{\lambda}_{\text{def},t} := \underset{\boldsymbol{\lambda} \in \Lambda}{\arg\min} \quad G(\{\boldsymbol{\lambda}\} \cup \Lambda_{\text{def},t-1}) \tag{2}$$

$$\Lambda_{\text{def},t} := \{\boldsymbol{\lambda}_{\text{def},1}, \dots, \boldsymbol{\lambda}_{\text{def},t}\} \tag{3}$$

where $\Lambda_{def,0} = \emptyset$, and the final solution $\Lambda_{\text{def}} = \Lambda_{\text{def},T}$. It is possible to estimate $R_k(\boldsymbol{\lambda})$ empirically using cross-validation, but since this is computationally expensive, we employ *surrogate models* that predict the performance for a given hyperparameter configuration resulting in a fast approximate way to evaluate performances. This approach can be extended to a set of defaults across algorithms.

3 EXPERIMENTAL EVALUATION

We estimate the generalization performance of our approach on future datasets by running a leave-one-dataset-out CV scheme over K datasets, estimating performances for each held-out dataset using outer 10-fold CV and *nested* 5-fold CV for choosing the hyperparameter. We compare to *random search* with several budgets and *Bayesian optimization* with 32 iterations. We use ±137.000 experimental results available on OpenML [4] to evaluate the lists of defaults on three algorithms from `scikit-learn` and 100 datasets from the OpenML100 [1]. We evaluate using Adaboost (5), SVM (6), and random forest (6 hyperparameters) optimizing predictive accuracy. Hyperparameters and their respective ranges are the same as used in [3]. Figure 1 presents the results of the set of defaults obtained by our approach and baselines across 3 algorithms normalized to $[0, 1]$ per algorithm and task and aggregate using the mean. For defaults and random search more iterations strictly improves performance. As expected, random search with only 1 or 2 iterations performs poorly, while Bayesian optimization is often among the best strategies. We further observe that using only a few defaults is already competitive with Bayesian optimization and higher budget random search, often competitive with random search with $4 - 8$ times more budget. We note that using sets of defaults is especially worthwhile when either computation time or expertise on hyperparameter optimization is lacking. Especially in the regime of few function evaluations, sets of defaults seem to work well and are statistically equivalent to state-of-the-art techniques. A potential drawback is that the defaults are optimal with respect to a single metric such as accuracy or AUC, and thus might need to be used separately for different evaluation metrics. Our results can readily be implemented in machine learning software as simple, hard-coded lists of parameters. These will require less knowledge of hyperparameter optimization from the users than current methods, and lead to faster results in many cases.

Acknowledgements. This work has been funded by the German Federal Ministry of Education and Research (BMBF) under Grant No. 01IS18036A. The authors of this work take full responsibility for its content.

REFERENCES

[1] Bernd Bischl, Giuseppe Casalicchio, Matthias Feurer, Frank Hutter, Michel Lang, Rafael G Mantovani, Jan N van Rijn, and Joaquin Vanschoren. 2017. OpenML Benchmarking Suites and the OpenML100. *arXiv preprint arXiv:1708.03731v1* (2017).
[2] Matthias Feurer, Jost Tobias Springenberg, and Frank Hutter. 2015. Initializing Bayesian Hyperparameter Optimization via Meta-learning. In *Proc. AAAI* (Austin, Texas). AAAI Press, 1128–1135.
[3] Jan N. van Rijn and Frank Hutter. 2018. Hyperparameter Importance Across Datasets. In *Proc. of KDD*. ACM, 2367–2376.
[4] J. Vanschoren, J. N. van Rijn, B. Bischl, and L. Torgo. 2014. OpenML: networked science in machine learning. *ACM SIGKDD Explorations Newsletter* 15, 2 (2014), 49–60.
[5] Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. 2015. Learning hyperparameter optimization initializations. In *Proc. of DSAA*. IEEE, 1–10.
[6] Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. 2015. Sequential Model-Free Hyperparameter Tuning. In *Proc. of ICDM*. 1033–1038.

4.4 Collecting Empirical Data About Hyperparameters for Data Driven AutoML

Contributed Article:
M. Binder, F. Pfisterer, and B. Bischl. Collecting empirical data about hyperparameters for data driven AutoML. In *AutoML Workshop at ICML*, 2020

Declaration of contributions The project was proposed by FP and MB based on a previous experiment by [145]. MB developed the evaluation protocol and scripts together with the scheduling methodology that was integral to the project success. FP contributed large parts of the experimental design, such as datasets, learners and search spaces to evaluate, as well as predictive memory allocation based on data from a smaller pilot run. FP and MB jointly extended the original scope to include multi-fidelity as well as repeated evaluations. The experimental evaluation was largely done by MB with minor assistance by FP. MB and FP jointly drafted the manuscript with refinement by BB. BB assisted with advice throughout the project.

7th ICML Workshop on Automated Machine Learning (2020)

Collecting Empirical Data About Hyperparameters for Data Driven AutoML

Martin Binder MARTIN.BINDER@STAT.UNI-MUENCHEN.DE
Florian Pfisterer FLORIAN.PFISTERER@STAT.UNI-MUENCHEN.DE
Bernd Bischl BERND.BISCHL@STAT.UNI-MUENCHEN.DE
Department of Statistics, LMU Munich, Germany

Abstract

All optimization needs some kind of prior over the functions it is optimizing over. We used a large computing cluster to collect empirical data about the behavior of ML performance, by randomly sampling hyperparameter values and performing cross-validation. We also collected information about cross-validation error by performing some evaluations multiple times, and information about progression of performance with respect to training data size by performing some evaluations on data subsets. We present how we collected data, make some preliminary analyses on the surrogate models that can be built with them, and give an outlook over interesting analysis this should enable.

1. Introduction

Hyperparameter optimization (HPO) is an important aspect of automated machine learning (AutoML) and is often tackled as a black-box optimization problem. The No Free Lunch (NFL) theorem for optimization (Wolpert and Macready, 1997) states that the performance of optimization algorithms, averaged over all possible problem sets, is constant, and that it is therefore necessary to make a-priori assumptions about the characteristics of the problem at hand. A well-known manifestation of this is the superiority of random search over grid search on typical hyperparameter optimization problems, which is a consequence of them often having low effective dimensionality (Bergstra et al., 2011). Another instance of this is the favorable performance of model-based optimization (Snoek et al., 2012) for hyperparameter optimization, which often uses explicit Bayesian priors (Bayesian optimization) to achieve good performance with few function evaluations. We performed evaluations of different ML algorithms with randomized hyperparameters on a variety of datasets to gather empirical data about the influence of these hyperparameters on algorithm performance.

Model-based optimization makes use of surrogate models, which are regression models fitted to evaluated objective values of the problem. Surrogate models can also be used to benchmark or tune optimization algorithms (Eggensperger et al., 2013). With the large amount of performance data that we have collected it is possible to fit high fidelity surrogate models that can be used to analyse the behavior of classical machine learning algorithms with respect to their hyperparameters.

The importance of hyperparameter optimization for deep learning (neural architecture search, Elsken et al. (2019)) has been growing in recent years. Because of their nature – large hyperparameter spaces and very long model fitting times – they have led to more research into multi-fidelity approaches for HPO (Li et al., 2017; Tan and Le, 2019). Multi-fidelity approaches vary certain fidelity hyperparameters that trade-off between model performance evaluation cost and model performance. Here, an important prior assumption about the behavior of ML algorithms with respect to the fidelity parameters is that performance at low fidelity is somehow predictive for performance at high fidelity. Because this assumption is so crucial, it is important to conduct empirical studies, and to gather data on the effect of approaches towards trade-offs between model performance and computation time.

We have done a very large number of performance evaluations of machine learning algorithms on a diverse set of 119 datasets with randomly sampled hyperparameters to gather empirical data about the influence of hyperparameter configurations on machine learning performance. We performed some of these evaluations on different subsamples of the data, which can be used to empirically evaluate a sub-sampling approach to multi-fidelity optimization. We further used repeated cross-validation for some evaluations, to get an estimate of (some of the) uncertainty of the cross-validation estimators.

2. Related Work

The importance of gathering empirical data about hyperparameter influence on performance has been recognized and several projects exist that do this on different classes of machine learning algorithms and problems. Collections of experimental results like ours are often published with papers for new methods (Wistuba et al., 2015a; van Rijn and Hutter, 2018), but they are rarely as comprehensive and often only suffice for the particular task being presented. Especially in the context of deep learning, several collections of experimental data have been made available lately. Kühn et al. (2018) publish results on 38 tabular datasets across 6 algorithms for further analysis. NASBENCH-101 (Ying et al., 2019) provide a collection of experimental results across 423.000 convolutional neural network architectures on CIFAR-10 for faster and more reproducible analysis of neural architecture search strategies. (Metz et al., 2020) publish experimental results for neural network optimizers across 1162 diverse datasets and propose sets of default configurations.

Experimental data is already being used to improve optimization performance. For one, the HPOLib benchmark suite for black-box optimization (Eggensperger et al., 2013), which can be used by researchers to evaluate their black box optimization algorithms, uses surrogate models for some benchmarks (Eggensperger et al., 2015; Klein et al., 2019). Optimization algorithms may get tuned on, or at least chosen by their performance on, these surrogate models. The experimental data – on which the models are based – are thus influencing the implicit prior assumptions made by these algorithms in an indirect way. A more direct influence of experimental data on optimization is meta-learning (Brazdil et al., 2008; Vanschoren, 2019), e.g. by studying the importance of various hyperparameters (van Rijn and Hutter, 2018; Probst et al., 2018), or try to find initial configurations (Wistuba et al., 2015b; Pfisterer et al., 2018; van Rijn et al., 2018) that perform well on many datasets.

3. Setup

We executed a large quantity of machine learning performance evaluations on a large grid of computers of 3168 compute nodes, each with 48 physical (96 logical) CPU cores and 80 GB working memory, for 48 hours.

The evaluations were performed on a collection of 119 classification task datasets chosen from the OpenML-CC18 (Bischl et al., 2017) benchmark suite, as well as the AutoML benchmark (Gijsbers et al., 2019). These datasets cover a large variety of different challenges for machine learning, such as many missing values, large cardinality of factorial features, large number of features, or great imbalance of outcome classes. We obtained datasets, as well as resampling splits from OpenML (Vanschoren et al., 2014).

We investigate an array of classical machine learning algorithms: decision trees, random forests, svm, gradient boosting, approximate KNN (Malkov and Yashunin, 2020), fully connected neural networks, and regularized logistic regression (Zou and Hastie, 2005). Because not all these algorithms can natively handle all datasets, we performed data-preprocessing: missing value imputation, factorial feature cardinality reduction, and factor one-hot encoding among others. The specific search spaces (including software libraries used) as well as the preprocessing setup are detailed in Appendix A and B, respectively.

Values for each hyperparameter were sampled independently from individual univariate distributions. A common problem in previous data collections is a limited hyperparameter search space. If good values lie on the border of investigated hyperparameter spaces (e.g. large number of gradient boosting iterations), then meta-learning approaches might miss important facts about algorithm behavior, such as eventual overfitting beyond the investigated region. Using too broad limits for sampling, on the other hand, can lead to many evaluations in uninteresting regions where the algorithm crashes or predicts constant. We alleviate this problem by defining intervals to sample from that were chosen informally and from previous experience. However, we purposely sample outside of these intervals with a small probability in order to obtain a more complete picture w.r.t. algorithm behavior beyond the regions. For this we sample a mixture distribution: uniformly distributed inside a specified range with probability 5/6, and normally distributed centered in the middle of this range and standard deviation half the range width with probability 1/6. The investigator-chosen ranges were therefore *soft bounds* that contain about $5/6 + 1/6 \times 68\% = 95\%$ of all sampled points[1]. We made the prior assumption that many hyperparameters contain more variation close to 0 than further away from it. These were sampled as described here, but on a log-scale (including mixture uniform-normal sampling) and then exponentiated. Hyperparameters with nominal discrete values were sampled uniformly. The specific hyperparameters, their bounds and transformation are listed in Appendix A.

The same hyperparameter samples were used on all datasets. This makes it possible to investigate the direct effect of dataset properties on machine learning performance while keeping hyperparameters constant, without having to resort to surrogate modelling.

Performance evaluation was performed using 10-fold cross-validation, using pre-defined, balanced (with respect to the outcome class) cross-validation folds provided by OpenML. To make it possible to investigate the effect of subsampling before model training, or the effect of the specific resampling split on the performance estimate, we also performed what we call *super-evals* (supererogatory evaluations) on 10% of all sampled hyperparameter points. (Which configurations were super-evaluated was kept constant across all datasets). For super-evals, we employed: (1) the 10-fold cross-validation used on all other configuration points; (2) another 10-times-10-fold repeated cross-validation; and (3) a sequence of

1. Some hyperparameter also presented *hard bounds*, e.g. when negative values are forbidden, in which case hyperparameter values were sampled from a truncated distributions.

Algorihtm	N	# of DS	avg. N	min N	max N	OpenML ID
glmnet	104820	114	919	58	2235	42578
ranger	278863	119	2343	356	4754	42580
knn	111753	116	963	146	2792	42582
rpart	92067	115	801	85	1382	42583
svm	540576	106	5100	174	13352	42577
xgboost	2955210	119	24834	2294	41147	42584
feed-forward NN	171691	107	1605	730	4133	42579

Table 1: Information about generated data: Experiment counts across different algorithms, number of datasets for which data could be generated, average, min, and max number of values per dataset, and OpenML ID of the performance data. Values not counting super-evals.

cross-validations with reduced training set size using subsampling (see Appendix C for the subsample sizes), simulating one approach to multi-fidelity cross-validation. The specific subsamples are a tower of subsets of the training sets used for (1) and fixed for each dataset.

Performance was evaluated by training the machine learning methods on the chosen training data subset and predicting on all other data-samples. Individual evaluation threads were limited w.r.t. the amount of working memory (up to 55 GB; memory limits were different depending on the algorithm and dataset) they could consume, and the amount of time they could use for a single cross-validation fold (4 hours). Available resources (both time and memory) for each learner and dataset were determined beforehand in a small pilot experiment. Therefore, the amount of data generated varies across them. A small number of datasets were too large for some ML algorithms and no data could be generated with the randomly sampled hyperparameters given the time or memory constraints.

Besides machine learning performance data, we also collected information about the working memory required for each configuration, as well as the training and prediction runtime. This data can be used to investigate memory and time requirements for different algorithms based on dataset properties and hyperparameters, and may help to improve AutoML systems to better adhere to runtime and memory limits in the future. Table 1 gives an overview of the generated data.

4. Analysis

Several routes for the analysis of this data can be envisioned. In this work we decide to study the quality of resulting surrogate models and the effect of the number of evaluated configurations on that quality. We use the collected data to fit random forest surrogate models. The usefulness of a surrogate model depends on its predictive performance, which we can estimate using cross-validation.

It is often not clear how much experimental data is needed to build sufficiently accurate surrogate models. The large amount of data that we have gathered makes it possible to analyse the behavior of surrogate model fidelity with respect to data size, and to estimate the marginal gain in model quality that would have been possible if even more data had been collected. We set the resampling error, i.e. the error introduced by using surrogate models

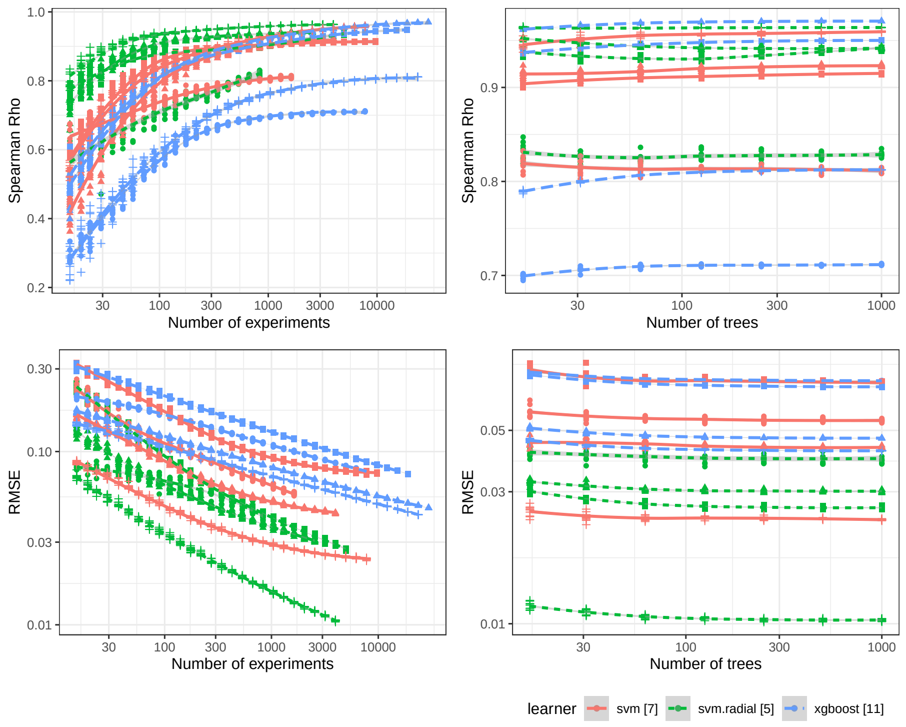


Figure 1: Resampling error of surrogate models with respect to number of included performance data points (left) and number of trees for the surrogate model (right) across 4 different datasets. Algorithm svm is restricted to the radial kernel. It can be seen that surrogate model performance varies with dataset as well as the learning algorithm.

instead of actual function evaluations, in relationship to the random noise introduced by having finitely many trees in the random forest model, by analysing the progression of error with different number of trees in the model.

Figure 1 shows the (ten-fold cross-validation) resampling error, both in terms of root mean square error (RMSE) as well as Spearman rank correlation (Spearman Rho), and how it progresses with different training set sizes, exemplary for a few datasets and algorithms evaluated on them. Furthermore, the resampling error w.r.t. the number of trees used in the surrogate model is shown. It becomes obvious that a few hundred random forest trees are enough for all shown datasets, and that the limiting factor is the sample size.

4.1 Hyperparameter Response Surface

Figure 2 shows the average and standard deviation of the normalized accuracy for the SVM `cost` and `gamma` parameters evaluated on a grid with resolution 200. Accuracy was normalized to $[0, 1]$ for each task to improve commensurability. Standard deviation seems to be low in areas with generally good performance and high for larger values of `gamma`. As we gathered a large number of evaluations for each dataset, we can build high-fidelity surrogate models which allow for a more realistic analysis of corresponding response surface.

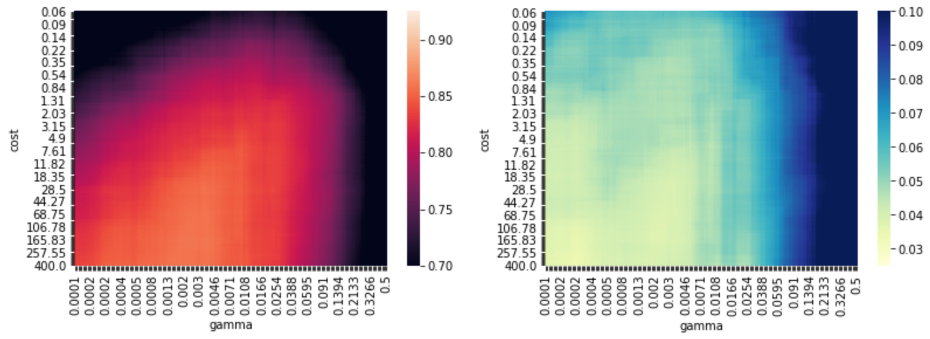


Figure 2: Response surfaces of surrogate models for the performance of SVM with respect to its `gamma` and `cost` parameters. Mean (left) and standard deviation (right) of normalized classification accuracy across the 106 datasets for which SVM generated results.

5. Further Analysis & Future Work

We presented the large collection of resampling performance data which we have generated, and showed some statistics and illustrative analysis results. There are, however, many directions to analyse and make use of the produced hyperparameter data in this work. We would like to use this data in order to reproduce and extend the results of several methods for the analysis of hyperparameter importance (van Rijn and Hutter, 2018; Probst et al., 2018). Furthermore the collected data allows for further investigation on hyperparameter response surfaces. Meta-learning often assumes that knowledge for the optimization of one task can be transferred to others. This would require a certain degree of similarity across response surfaces, which can be empirically validated using our data. The runtime and memory measurement also allows to build models predicting resource requirements (c.f. Hutter et al. (2014)). This could be used to improve multi-fidelity approaches or to perform automated scheduling for parallel workflows in AutoML systems.

We make the generated data publicly available on OpenML, with IDs listed in Table 1. The code used to generate the the data is available online[2].

Acknowledgments

The authors gratefully acknowledge the Gauss Centre for Supercomputing e.V.[3] for funding this project by providing computing time on the GCS Supercomputer SuperMUC-NG at Leibniz Supercomputing Centre[4].

This work has been funded by the German Federal Ministry of Education and Research (BMBF) under Grant No. 01IS18036A. The authors of this work take full responsibilities for its content.

2. `https://github.com/compstat-lmu/randombot_ng/`
3. `www.gauss-centre.eu`
4. `www.lrz.de`

Appendix A. Search Spaces

The following tables list the learning algorithms and their respective sampled hyperparameter bounds. Besides their natural hyperparameters, algorithms are also equipped with the `num.impute.selected.cpo` hyperparameter, which controls imputation for missing values in numeric features: mean, median, or histogram sampling imputation. All algorithms were used as implemented or interfaced to the R (R Core Team, 2019) programming language for statistics and the `mlr` framework for machine learning in R (Bischl et al., 2016), using the `mlrCPO` (Binder et al., 2020) set of composable preprocessing operators for preprocessing. "KerasFF" is a fully connected neural network implemented via `keras` (Chollet et al., 2015) with hyperparameters controlling the architecture (number of neurons and layers, magnitude of dropout, ...) and the optimizer (learning rate, weight decay, ...). We additionally vary the network's seed in order to obtain more reliable estimates with respect to randomness induced by different weight initializations. "RcppHNSW" is an approximate k-nearest neighbor implementation based on hierarchical navigable small world graphs (Malkov and Yashunin, 2020). All other learners directly interface existing implementations, information on their hyperparameters and meaning can be obtained from the respective software's documentation: xgboost: (Chen and Guestrin, 2016), Random Forest: (Wright and Ziegler, 2017), Elastic Net: (Friedman et al., 2010) and Decision Trees: (Therneau and Atkinson, 2018).

Hyperparameter	Range
epochs	$[2^3, 2^7](log)$
optimizer	sgd, rmsprop, adam
lr	$[5^{-5}, 5^0](log)$
decay	$[5^{-8}, 5^0](log)$
momentum	$[5^{-8}, 5^0](log)$
layers	[1, 4]
batchnorm_dropout	batchnorm, dropout, none
input_dropout_rate	$[3^{-5/2}, 3^0](log)$
dropout_rate	$[3^{-5/2}, 3^0](log)$
units_layer1	$[2^3, 2^9](log)$
units_layer2	$[2^3, 2^9](log)$
units_layer3	$[2^3, 2^9](log)$
units_layer4	$[2^3, 2^9](log)$
act_layer	relu, tanh
init_layer	glorot_normal, glorot_uniform, he_normal, he_uniform
l1_reg_layer	$[5^{-10}, 5^{-2}](log)$
l2_reg_layer	$[5^{-10}, 5^{-2}](log)$
learning_rate_scheduler	TRUE, FALSE
init_seed	1, 11, 101, 131, 499
num.impute.selected.cpo	impute.mean, impute.median, impute.hist

Table 2: Sample bounds of the KerasFF learning algorithm.

Hyperparameter	Range
k	[1, 50]
distance	l2, cosine, ip
M	[18, 50]
ef	$[2^3, 2^8](log)$
ef_construction	$[2^4, 2^9](log)$
num.impute.selected.cpo	impute.mean, impute.median, impute.hist

Table 3: Sample bounds of the RcppHNSW learning algorithm.

Hyperparameter	Range
nrounds	$[2^3, 2^{11}](log)$
eta	$[2^{-10}, 2^0](log)$
gamma	$[2^{-15}, 2^3](log)$
lambda	$[2^{-10}, 2^{10}](log)$
alpha	$[2^{-10}, 2^{10}](log)$
subsample	[0.1, 1]
max_depth	[1, 15]
min_child_weight	$[2^0, 2^7](log)$
colsample_bytree	[0.01, 1]
colsample_bylevel	[0.01, 1]
num.impute.selected.cpo	impute.mean, impute.median, impute.hist

Table 4: Sample bounds of the XGBoost learning algorithm.

Hyperparameter	Range
num.trees	[1, 2000]
replace	TRUE, FALSE
sample.fraction	[0.1, 1]
mtry.power	[0, 1]
respect.unordered.factors	ignore, order, partition
min.node.size	[1, 100]
splitrule	gini, extratrees
num.random.splits	[1, 100]
num.impute.selected.cpo	impute.mean, impute.median, impute.hist

Table 5: Sample bounds of the Ranger (random Forest) learning algorithm.

Hyperparameter	Range
kernel	linear, polynomial, radial
cost	$[2^{-12}, 2^{12}](log)$
gamma	$[2^{-12}, 2^{12}](log)$
degree	[2, 5]
tolerance	$[2^{-12}, 2^{-3}](log)$
shrinking	TRUE, FALSE
num.impute.selected.cpo	impute.mean, impute.median, impute.hist

Table 6: Sample bounds of the SVM learning algorithm.

Hyperparameter	Range
cp	$[2^{-10}, 2^{0}](log)$
maxdepth	[1, 30]
minbucket	[1, 100]
minsplit	[1, 100]
num.impute.selected.cpo	impute.mean, impute.median, impute.hist

Table 7: Sample bounds of the RPART (decision tree) learning algorithm.

Hyperparameter	Range
alpha	[0, 1]
s	$[2^{-10}, 2^{10}](log)$
num.impute.selected.cpo	impute.mean, impute.median, impute.hist

Table 8: Sample bounds of the glmnet (elastic net) learning algorithm.

Appendix B. Preprocessing

We preprocess data using `mlrCPO` using the following procedure:

1. fixfactors: Set all categorical features missing not present during training to MISSING in the prediction phase
2. imputation: Impute using a new level for categorical features and mean, median or histogram for numerics. The latter is a hyperparameter exposed for all learners and explored during the sampling procedure.
3. We add indicator columns for missing numeric values.
4. We limit the number of factor levels for a given categorical variable to 32. All other columns with lower cardinality are collapsed to a "other" category.
5. We drop constant features.
6. If learners can not handle categorical features natively (xgboost, keras, rcpphnsw), we encode those using dummy encoding.

Appendix C. Subsampling

We use subsampling with the following factions of the training data:
$0.05, 0.1, 0.15, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9$

References

James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, pages 2546–2554, 2011.

Martin Binder, Lars Kotthoff, Michel Lang, and Bernd Bischl. *mlrCPO: Composable Preprocessing Operators and Pipelines for Machine Learning*, 2020. R package version 0.3.6.

B. Bischl, M. Lang, L. Kotthoff, J. Schiffner, J. Richter, E. Studerus, G. Casalicchio, and Z. M. Jones. mlr: Machine learning in R. *JMLR*, 17(170):1–5, 2016.

Bernd Bischl, Giuseppe Casalicchio, Matthias Feurer, Frank Hutter, Michel Lang, Rafael G Mantovani, Jan N van Rijn, and Joaquin Vanschoren. Openml benchmarking suites and the openml100. *arXiv preprint arXiv:1708.03731*, 2017.

Pavel Brazdil, Christophe Giraud Carrier, Carlos Soares, and Ricardo Vilalta. *Metalearning: Applications to data mining.* Springer Science & Business Media, 2008.

Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4232-2.

François Chollet et al. Keras. `https://keras.io`, 2015.

Katharina Eggensperger, Matthias Feurer, Frank Hutter, James Bergstra, Jasper Snoek, Holger Hoos, and Kevin Leyton-Brown. Towards an empirical foundation for assessing bayesian optimization of hyperparameters. In *NIPS workshop on Bayesian Optimization in Theory and Practice*, volume 10, page 3, 2013.

Katharina Eggensperger, Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. Efficient benchmarking of hyperparameter optimizers via surrogates. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20:1–21, 2019.

Jerome Friedman, Trevor Hastie, and Rob Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1–22, 2010.

Pieter Gijsbers, Erin LeDell, Janek Thomas, Sébastien Poirier, Bernd Bischl, and Joaquin Vanschoren. An open source automl benchmark. *arXiv preprint arXiv:1907.00909*, 2019.

Frank Hutter, Lin Xu, Holger H Hoos, and Kevin Leyton-Brown. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence*, 206:79–111, 2014.

Aaron Klein, Zhenwen Dai, Frank Hutter, Neil Lawrence, and Javier Gonzalez. Meta-surrogate benchmarking for hyperparameter optimization. In *Advances in Neural Information Processing Systems*, pages 6270–6280, 2019.

Daniel Kühn, Philipp Probst, Janek Thomas, and Bernd Bischl. Automatic Exploration of Machine Learning Experiments on OpenML. *arXiv preprint arXiv:1806.10961*, 2018.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816, 2017.

Yury A Malkov and Dmitry A Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(4):824–836, 2020.

Luke Metz, Niru Maheswaranathan, Ruoxi Sun, C Daniel Freeman, Ben Poole, and Jascha Sohl-Dickstein. Using a thousand optimization tasks to learn hyperparameter search strategies. *arXiv preprint arXiv:2002.11887*, 2020.

Florian Pfisterer, Jan N. van Rijn, Philipp Probst, Andreas M. Müller, and Bernd Bischl. Learning Multiple Defaults for Machine Learning Algorithms. *arXiv preprint arXiv:1811.09409*, 2018.

P. Probst, B. Bischl, and A. Boulesteix. Tunability: Importance of hyperparameters of machine learning algorithms. *arXiv preprint arXiv:1802.09596*, 2018.

R Core Team. *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria, 2019.

Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'12, pages 2951–2959, USA, 2012. Curran Associates Inc.

Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114, 2019.

Terry Therneau and Beth Atkinson. *rpart: Recursive Partitioning and Regression Trees*, 2018. R package version 4.1-13.

Jan N. van Rijn and Frank Hutter. Hyperparameter importance across datasets. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2367–2376. ACM, 2018.

Jan N van Rijn, Florian Pfisterer, Janek Thomas, Andreas Muller, Bernd Bischl, and Joaquin Vanschoren. Meta learning for defaults–symbolic defaults. In *Neural Information Processing Workshop on Meta-Learning*, 2018.

Joaquin Vanschoren. Meta-learning. In *Automated Machine Learning*, pages 35–61. Springer, Cham, 2019.

Joaquin Vanschoren, Jan N Van Rijn, Bernd Bischl, and Luis Torgo. Openml: networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 15(2):49–60, 2014.

Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. Hyperparameter search space pruning–a new component for sequential model-based hyperparameter optimization. In *Proc. of ECML/PKDD 2015*, pages 104–119. Springer, 2015a.

Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. Learning hyperparameter optimization initializations. In *Data Science and Advanced Analytics (DSAA), 2015. 36678 2015. IEEE International Conference on*, pages 1–10. IEEE, 2015b.

D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

Marvin N Wright and Andreas Ziegler. ranger: A fast implementation of random forests for high dimensional data in C++ and R. *Journal of Statistical Software*, 77(1):1–17, 2017.

Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. NAS-bench-101: Towards reproducible neural architecture search. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 7105–7114, Long Beach, California, USA, 2019. PMLR.

Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the royal statistical society: series B (statistical methodology)*, 67(2):301–320, 2005.

4.5 Multi-Objective Automatic Machine Learning with AutoxgboostMC

Contributed Article:
F. Pfisterer, S. Coors, J. Thomas, and B. Bischl. Multi-objective automatic machine learning with AutoxgboostMC. In *Automating Data Science Workshop at ECML*, 2019, arXiv:1908.10796

Declaration of contributions The idea of developing a multi-objective AutoML system originated from FP as a result of discussions with JT and BB. FP developed code for the system and developed adaptations made throughout the paper such as adaption of the search procedure and thresholding. Developed code was based on a previous project [231] by JT. JT and BB provided helpful feedback during method development. FP wrote the manuscript with minor contributions by JT, SC and BB. SC furthermore contributed the main experimental evaluation in the paper and helped in refining the manuscript.

Multi-Objective Automatic Machine Learning with AutoxgboostMC

Florian Pfisterer, Stefan Coors, Janek Thomas, and Bernd Bischl

LMU Munich

Abstract. AutoML systems are currently rising in popularity, as they can build powerful models without human oversight. They often combine techniques from many different sub-fields of machine learning in order to find a model or set of models that optimize a user-supplied criterion, such as predictive performance. The ultimate goal of such systems is to reduce the amount of time spent on menial tasks, or tasks that can be solved better by algorithms while leaving decisions that require human intelligence to the end-user. In recent years, the importance of other criteria, such as fairness and interpretability, and many others have become more and more apparent. Current AutoML frameworks either do not allow to optimize such secondary criteria or only do so by limiting the system's choice of models and preprocessing steps. We propose to optimize additional criteria defined by the user directly to guide the search towards an optimal machine learning pipeline. In order to demonstrate the need and usefulness of our approach, we provide a simple multi-criteria AutoML system and showcase an exemplary application.

1 Introduction

While many stages of a data analysis project still need to be done manually by human data scientists, other parts, such as model selection and algorithm configuration can be efficiently handled by algorithms. This does not only reduce the time required by humans, but also allows to leverage parallelization. A typical challenge is the selection of appropriate algorithms and corresponding hyperparameters for a given problem. Multiple methods for solving this *Combined Algorithm Selection and Hyperparameter optimization (CASH)* problem (Thornton et al., 2013) already exist and are typically referred to as *Automatic Machine Learning* (AutoML).

There is a growing number of approaches for AutoML available to non-specialists. As one of the first frameworks, Auto-WEKA (Thornton et al., 2013) introduced a system for automatically choosing from a broad variety of learning algorithms implemented in the open source software *WEKA* (Hall et al., 2009). Auto-WEKA simultaneously tunes hyperparameters over several learning algorithms using the Bayesian optimization framework SMAC (Hutter et al., 2011). Similar to Auto-WEKA is *auto-sklearn* (Feurer et al., 2015), which is based on the scikit-learn toolkit for python and includes all of its learners as well as available preprocessing operations. It stacks multiple models to achieve high predictive performance. Another python-based AutoML tool is called *Tree-based Pipeline Optimization Tool (TPOT)* by Olson et al. (2016) and uses genetic programming instead of Bayesian optimization to tune over a similar space as auto-sklearn.

In this work we consider an approach that configures a *machine learning pipeline*, i.e. an approach that optimizes pre- and post-processing steps along with algorithm hyperparameters of a single gradient boosting model (Friedman, 2001). By focusing on a single learning algorithm, hyperparameters can be optimized much more thoroughly and the resulting model can be analyzed and deployed more easily. Gradient boosting models can vary from very simple to highly complex models through the choice of appropriate hyper-parameters. Single learner systems reduce the complexity of the configuration space, however, a drawback is, that this search-space possibly does not include optimal configurations, benefits from stacking and ensembling are not explored, and that thus optimal predictive performance may not be achieved.
Only few single-learner AutoML methods exist. The *autoxgboost* software proposed by Thomas et al. (2018) is a single-learner strategy using the *xgboost* (Chen and Guestrin (2016)) algorithm, with model based optimization for hyperparameter tuning. The success of such single-learner strategies was shown in the NIPS 2018 AutoML Challenge (Guyon et al., 2019): The winning entry, *AutoGBT* (Wilson et al. (2018)) only used LightGBM (Ke et al., 2017) models with a simple preprocessing scheme.

Several new challenges occur when adapting AutoML systems for multi-criteria optimization. Depending on the objective to be optimized, different pre- and post-processing methods might be required in order to obtain optimal performances. Additionally, different user-preferences regarding which trade-offs between objectives a user is willing to make have to be incorporated. We argue that giving the user the opportunity to intervene in the process can be beneficial here. Lastly, measures that quantify objectives such as *fairness*, *interpretability* and *robustness* are often not readily available. In this work, we want to i) emphasize the necessity for considering multiple objectives in AutoML, ii) provide several measures that can be useful in such a context and iii) propose a simple system that allows the user to automatically optimize over a set of measures. In contrast to previous work, we focus on optimizing multiple user-defined criteria simultaneously. Being able to transparently optimize multiple criteria is a crucial missing step in many existing frameworks. In order to underline the need for several different criteria, we demonstrate the functionality of our proposed framework in a practical use case.

2 The case for additional criteria in AutoML

Multi-criteria optimization is well-established in machine learning for example in ROC analysis (Everson and Fieldsend (2006)), computational biology (Handl et al. (2007)) and other fields. Jin and Sendhoff (2008) study various use cases, among others, models are optimized jointly with respect to interpretability and predictive performance. Multi-criteria optimization is also actively researched in the field of Algorithm Configuration. Blot et al. (2016) introduce a multi-criteria iterative local search procedure for configuring SAT solvers, while Zhang et al. (2015) introduce a racing-based approach. Different Multi-criteria Bayesian Optimization approaches have also been proposed (c.f. Paria et al. (2018)), but it has not been thoroughly investigated as a part of AutoML frameworks until now. Many different algorithms, such as approaches based on iterated local search or racing as well as genetic algorithm based approaches can be used to optimize machine learning pipelines, given that they can deal with hierarchical mixed continuous and discrete spaces. We choose Bayesian Optimization because it has been shown to work well with relatively small budgets (Bischl et al., 2018) and complex hierarchical spaces can be optimized by using random forests as surrogate models.

Only being able to optimize a single performance measure entails multiple pitfalls that can possibly be avoided when multiple performance measures are optimized jointly. This has been emphasized recently in the **FatML** (Fairness, Accountability, and Transparency in Machine Learning) community which made the case for models that emphasize transparency and fairness (Barocas et al., 2018). The need for models that do not discriminate against parts of the population in order to achieve optimal predictive performance has garnered widespread support, yet no real options that allow users to jointly search for fair, transparent and well-performing models are available. The case for other criteria, that might be relevant to a user has been made in many other areas of machine learning. Examples include models that emphasize sparseness, a lower inference time, i.e., when searching for items in databases (Johnson et al., 2017), a low memory footprint, for example when deploying models on mobile phones (Howard et al., 2017) or a combination of those when doing inference on edge devices (Huang et al. (2016)). Similarly, the case for requiring *robust* models, i.e., models that are robust to *perturbations* in the data (adversarial perturbations, c.f Papernot et al. (2015)) can be made. Models that satisfy a user-desired trade-off might not be found using single-criteria optimization (Jin and Sendhoff (2008)). It is important to distinguish between jointly optimizing multiple optimization criteria, and constrained optimization (c.f. Hernández-Lobato et al. (2016) for an overview). Achieving a certain model size might be paramount to be able to deploy a machine learning pipeline to a end user device, but having a model smaller than this size threshold is only of minor interest. The concept of constraints in multi-criteria optimization is a well researched topic (c.f. Fan et al. (2017)).

2.1 Human in the loop approaches in AutoML

The original aim of AutoML systems is to transfer the CASH problem from the hands of a human to the machine. This does not only allow experienced machine learning researchers to focus on other tasks, such as validating data and feature engineering leveraging domain knowledge, but also enables a broader public to apply Machine Learning, as steps that require a machine learning expert, like selecting algorithms and tuning their hyperparameters, are fully automated. The AutoML system is thus treated as a black-box, that can only be influenced by some hyperparameters at the beginning of the training, essentially removing the human from the optimization loop.

In situations, where multiple criteria have to be optimized simultaneously, a trade-off between the different measures is often required. Specifying this trade-off *a priori* can be difficult when possible trade-offs are not known. Hakanen and Knowles (2017) propose an interactive Bayesian Optimization extension to parEgo (Knowles, 2004), that allows a user to iteratively select preferred ranges for the different optimization criteria. This does not only allow to search for solutions in the region a user is interested in, but also allows the user to adapt preferences throughout the procedure. This emphasizes the need for users to guide the AutoML process, essentially putting the user back into the loop, albeit in a different fashion. Instead of manually configuring the pipeline, the user is now able to occasionally supervise the search process and make adjustments where needed. A different approach, that allows the user to guide the search process by adapting the search space and tries to visualize and explain decisions made within AutoML systems has been proposed in (Wang et al., 2019).

2.2 Measures for Multi-Criteria AutoML

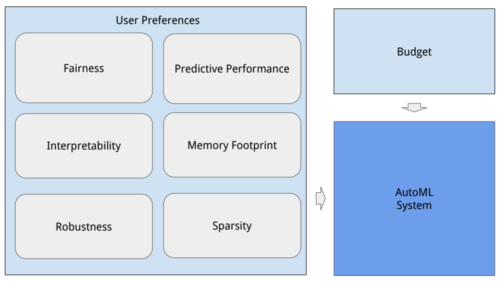


Fig. 1: User Input to AutoML Systems.

Multi-criteria optimization methods usually explore the whole pareto front defined by trade-offs between the different objectives. In our work, we mainly allow the user to guide the search process in two ways: We enable the user to focus on exploring different parts of the pareto front by selecting upper and lower trade-offs relevant to the user. Second, we allow the user to adapt the search space, by adjusting hyperparameter ranges and activating or deactivating processing steps. This allows the user to shape the result towards personal preferences.

In order to start the investigation into Multi-Criteria AutoML, we aim to provide a list of measures that cover a wide variety of use-cases. We want to stress, that the proposed measures are not comprehensive or final, but instead can be thought of as exchangeable building blocks that can serve as a useful proxy in the AutoML process. We hope to emphasize the necessity for measures, that better reflect the underlying model characteristics we aim to optimize.

Predictive Performance can be quantified using many different measures, such as Accuracy, F-Score or Area under the Curve for classification and Mean Squared Error or Mean Absolute Error for regression. As those measures are already widely known, we refrain from going into more detail in this work.

Interpretability In order to make a machine learning model's decisions more transparent, different methods that aim at providing human-understandable explanations have been proposed (c.f. Molnar (2019)). Many of those work in a *model-agnostic* and *post-hoc* fashion, which is desirable for AutoML processes, as this allows the user to explain arbitrary models resulting from AutoML processes. Interpretability methods can produce misleading results if a model is too complex. Quantifying interpretability, i.e. determining how complex predictive decisions of a given model are could be a first step, making it a useful criterion to optimize for AutoML systems. A first approach has been proposed in Molnar et al. (2019), describing 3 measures that can be used as a proxy for interpretability. We implement those measures and briefly present each:

- **Complexity of main effects** Molnar et al. (2019) propose to determine the average shape complexity of ALE (Apley, 2016) main effects by the number of parameters needed to approximate the curve with linear segments.
- **Interaction Strength** Quantifying the impact of interaction effects is relevant when explanations are required, as most interpretability techniques use linear relationships to obtain explanations. Interaction Strength is measured as the fraction of variance that can not be explained by main effects.
- **Sparsity** can be a desired property in case a simple explanation of a model is required, or obtaining features is costly and can potentially be avoided. In this work we measure sparsity as the fraction of features used.

A different approach towards achieving interpretability, would to instead focus on limiting an AutoML system to models, that are inherently interpretable. As those models rarely achieve optimal performances and trade-offs between interpretability and predictive perfromance cannot be assessed, we resort instead to look for models that are well-suited for post-hoc interpretability.

Fairness has been established as a relevant criterion in Machine Learning when humans are subject to algorithmic decisions. The aim of the field is to encourage models that do not discriminate between certain sub-populations in the data. Hardt et al. (2016) define the concept of *equalized odds* and *equal opportunity*. Given a protected attribute A (e.g. gender), an outcome X, a binary predictor Y_b, several criteria can be derived.

- **Independence** or equalized odds can be measured as follows:
$$Pr\{Y_b = 1|A = 0, Y = 1\} = Pr\{Y_b = 1|A = 1, Y = 1\}$$
i.e. if the true positive rate is equal in sub-populations indicated by A.
- **Sufficiency** or equality of opportunity can be measured as follows:
$$Pr\{Y_b = 1|A = 0, Y = y\} = Pr\{Y_b = 1|A = 1, Y = y\}, y \in \{0, 1\}$$
i.e. if the false positive and the false negative rates are equal in sub-populations.
- **Calibration** is another desirable criterion for classifier, especially in the context of fairness, where we might want to have calibrated probabilities in all groups. Pleiss et al. (2017) show, that models that are well-calibrated but also have equalized odds are only possible in case the predictor is *perfect*, i.e does not make any errors.

Žliobaitė (2017) provide a review of various discrimination measures that can be used in this context. A score for fairness can now be derived for example from the absolute differences of the given measure in each subgroup. In the use-case below, we use the differences in F1-Scores as a measure we want to minimize. The F1 score is the harmonic mean between the True Positive Rate and the Positive predictive value, and thus trades off true positives, false negatives and false positives.

Addendum[1]

The use of machine learning in situations where individuals are affected by model decisions harbors opportunities as well as dangers. In the context of fairness, ML models can exhibit bias, similarly to humans. But in opposition to humans, bias in ML systems can often be explicitly measured. On the other hand, humans can be asked to justify or explain decisions, while this does not necessarily hold for ML models. It is important to note that fairness can not be achieved solely through a reduction into mathematical criteria e.g. statistical and individual notions of fairness such as disparate treatment, disparate impact etc. . Many problems with such metrics still persist and require additional research. Furthermore, practitioners need not only take into account the model itself, but also the data used to train the algorithm, the process behind the collection and labeling of such data and eventual feedback loops arising from use of potentially biased models. It is of utmost importance to scrutinize data and resulting models from the perspective of all sub-groups (and intersections of those) in order to avoid introducing bias and causing harm to individuals.

Robustness as a concept, describes the behaviour of machine learning algorithms in situations where the data originally used to train a model is changed. A formal definition of robustness is currently lacking, which might arise from the many different concepts such a definition would need to cover. Bousquet and Elisseeff (2002) define the notion of stability, which essentially measures how much a pre-defined loss-function deteriorates if an observation is held-out during training. This is not exactly what we are interested in as it requires extensive retraining. Instead we require a *post-hoc* method that operates on a fitted model and training or testing data. A different approach, also coined *stability* is provided in Lange et al. (2003). Their notion of stability measures the disagreement between a trained model on training data and test data. In this work, we detail three measures of robustness, which we deem helpful in certain situations.

- **Perturbations** A very simple measure of robustness could be a classifiers' robustness to minimal perturbations in the input data. We create a copy $X^\star$ of our data X by adding a small magnitude noise $N(0, \epsilon)$ scaled by ϵ, typically $0.001 - 0.01$ times the range of the numerical feature. The robustness to perturbations can then be measured via the absolute difference of some loss L, for example *accuracy*.

$$|L(X, Y) - L(X^\star, Y)|$$

- **Adversarial Examples** A widely researched area of robustness is the field of Adversarial Examples Szegedy et al. (2013); Papernot et al. (2015). Various different *adversarial* attacks and defenses against such attacks have been proposed. A variety of robustness measures can be derived from the different types of attacks proposed.

[1] The paragraph was added after presentation at ECML-PKDD ADS 2019 Workshop. Nonetheless, the authors deem it important to make the statement in order to avoid harm caused by biased machine learning systems. An excellent resource for additional information is e.g. 2020 CVPR Tutorial on Fairness Accountability Transparency and Ethics in Computer Vision /`https://sites.google.com/view/fatecv-tutorial`)

– **Distribution shift** is a concept that is gathering widespread interest not only as a research field Zhang et al. (2013), but also as a problem in AutoML, which became evident from the AutoML Challenge organized at the NIPS 2018 conference Guyon et al. (2019). To the author's knowledge, no measure that serves as a proxy for a model's robustness to distribution shift is available.

Inference Time and Memory requirements have been widely used as a measurement of the performance of machine learning algorithms. The time required for inference can be incorporated as a criterion.

Sparsity is also an important desideratum in other contexts, where interpretability is not necessarily required. In cases, where observing each feature incurs different costs, a user might want to find a model that achieves optimal performances using as few features as possible.

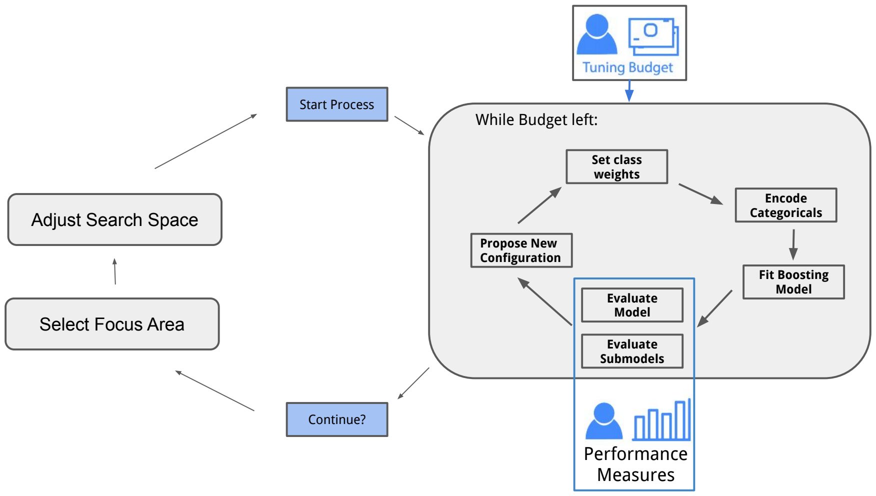


Fig. 2: Workflow for multi-criteria Autoxgboost. The user selects a measure and a budget, starts the process and then adapts the optimization before starting further evaluations.

3 Method

This section introduces the structure of a first simple approach for multi-criteria AutoML. We heavily base our software on Thomas et al. (2018), and include several design choices, such as the selection of preprocessing steps. The general workflow is detailed in Figure 2. The implementation can be obtained from github [2]. Automatic gradient

[2] `https://github.com/pfistfl/autoxgboostMC`

boosting simplifies AutoML to a fixed choice of machine learning algorithm by only using gradient boosting with trees (GBT). Gradient Boosted Decision Trees are widely successfull for learning on tabular data and have desirable properties for AutoML systems, as they can deal with missing observations, are insensitive to outliers and can handle large amounts of features and data points. Additionally they are numerically stable and memory efficient. Additionally modern GBT frameworks like *xgboost* Chen and Guestrin (2016) or *lightgbm* (Ke et al., 2017) are highly configurable with a large number of hyperparameters for regularization and optimization. As a result, they can approximate or cover many other scenarios, such as decision trees, random forests or linear models. Categorical feature transformation is performed as a preprocessing step. We employ Sequential model-based optimization (SMBO), also known as Bayesian Optimization as a hyperparameter optimization strategy (Snoek et al., 2012). The hyperparameter space we optimize is identical to Thomas et al. (2018). We use multi-criteria Bayesian Optimization (Bischl et al. (2016, 2018); Horn and Bischl (2016)) (c.f section 3.2) in order to optimize the machine-learning pipeline.

3.1 Sub-evaluations

In the context of multi-criteria optimization, early stopping is no longer trivial, as multiple pareto-optimal solutions might exist. The same holds for the selection of an optimal classification threshold as a postprocessing step in case a measure requiring binary outcomes instead of probabilities. At the same time, evaluating different thresholds or a sub-model using only a fraction of a model's gradient boosting iterations is very cheap after fitting a full model. In order to make use of this information we adopt the following procedure:

Algorithm 1 Bayesian Optimization using sub-evaluations

```
Require:
  $S_m \leftarrow \bigcup_1^m (\theta_i^\star, y_i)$: $m$ Initial evaluations
  $j \leftarrow m + 1$
  while Budget left do
      $\theta_j^\star \leftarrow$ proposed using Bayesian Optimization on $S_{j-1}$
      $f_{\theta^\star,j} \leftarrow$ fitted on data using $\theta_j, nrounds_j$ and applying $thr_j$.
      $y_j \leftarrow$ obtained by evaluating $f_{\theta^\star,j}$ for each measure.
      $S_j \leftarrow S_{j-1} \cup (\theta_j^\star, y_j)$
      $S_{sub,j} \leftarrow$ obtain sub-evaluations (Algorithm 2)
      $S_{sub,j} \leftarrow$ keep only $S_{sub,j}$ which are on the pareto front of $S_j \cup S_{sub,j}$
      $S_j \leftarrow S_j \cup S_{sub,j}$
      $j \leftarrow j + 1$
  end while
```

A full pipeline configuration $\theta^\star \in \Theta^\star$ is composed of a threshold $thr \in [0; 1]$, the number of boosting iterations $nrounds$ and several other pipeline hyperparameters, denoted by θ for simplicity. From a set of m randomly chosen initial configurations and their corresponding performances S_m we start multi-criteria Bayesian Optimization as described in Algorithm 1. The method for obtaining sub-evaluations is described in Algorithm 2. In order to decrease the number of sub-evaluations, we resort to evaluating only $25\%, 50\%, 75\%$ and 90% of $nrounds$ (rounded to the next integer). Although this section describes the case of classifying a *binary* target variable, extensions to multi-class classification can be made by instead using a vector of thresholds instead.

Algorithm 2 Obtaining Sub-evaluations

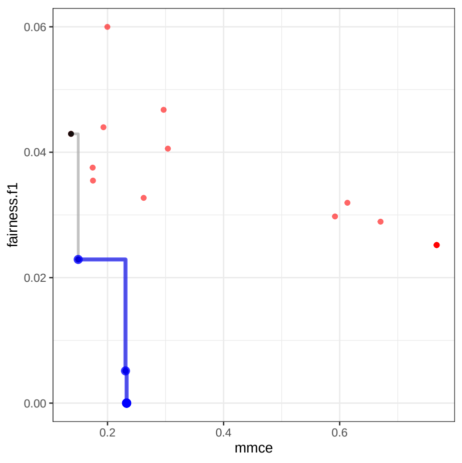
```
Require:
  Model $f_{\theta^\star,j}$; $i \leftarrow 1$
  for n in {1, ..., nrounds} do
      for $thr$ in $\{0, 0.1, 0.2, ..., 1\}$ do
          $S_i \leftarrow$ evaluate $f^\star_\theta$ using $n$ iterations, applying threshold $thr$
          $i \leftarrow i + 1$
      end for
  end for
  $S_{sub} \leftarrow \bigcup_1^i S_i$
```

3.2 Multi-Criteria Bayesian Optimization

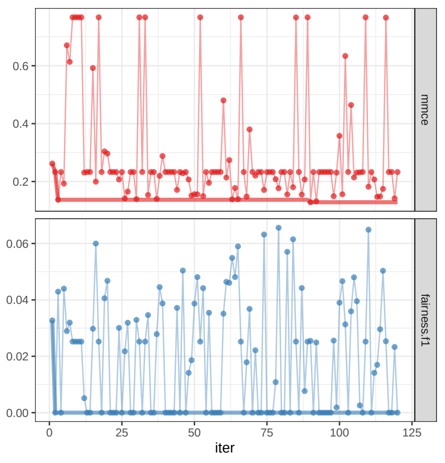
Formally multi-criteria optimization problems are defined by a set of target functions $f(\theta) = (f_1(\theta), \dots, f_k(\theta))$ which should be optimized simultaneously. As there is no inherent order between the targets, the concept of *Pareto dominance* is used to rank different candidate configurations. One configuration θ pareto-dominates another configuration $\tilde{\theta}$, $\theta \preceq \tilde{\theta}$, if $f_i(\theta) \leq f_i(\tilde{\theta})$ for $i = 1, \dots, k$ and $\exists j\, f_j(\theta) < f_j(\tilde{\theta})$, i.e., θ needs to be as good as $\tilde{\theta}$ in each component and strictly better in at least one. A configuration θ is said to be *non-dominated* if it is not dominated by any other configuration. The set of all non-dominated points is the *Pareto set*, which contains all trade-off solutions. Finally, the *Pareto front* is evaluation of all configurations in the *Pareto set*. The goal of multi-criteria optimization is to learn the Pareto set. There exists a plethora of different ways to extend Bayesian optimization to the multi-criteria case. We choose **parEgo** Knowles (2004), as it is a simple method, and it naturally lends itself to focussing regions of the pareto front. **parEgo** is a rather simple extension, which scalarizes the set of target functions by using the augmented Tchebycheff norm

$$\max_{i=1,\dots,k}(w_i f_i(\theta)) + \rho \sum_{i=i}^{k} w_i f_i(\theta),$$

with a different uniformly sampled weight vector w such that $\sum_{i=1}^{k} w_i = 1$ in each iteration. The augmentation term $\rho \sum_{i=i}^{k} w_i f_i(\theta); \rho > 0$ is used to guarantee pareto-optimal solutions (Miettinen and Mäkelä, 2002). This allows to apply standard single-criteria Bayesian optimization to the scalarized target function. Furthermore the use of the augmented Tchebycheff norm allows to exclude regions of the pareto front which are



(a) Full pareto front of first AutoxgboostMC run. Not pareto-optimal points are displayed in red.



(b) Final optimization path.

not of practical relevance, e.g., models with extremely low predictive accuracy do not have to be considered regardless of their interpretability or fairness (Steuer and Choo (1983), Hakanen and Knowles (2017)). This is done by constraining the values of some w_i between certain values. We adapt a similar procedure, where the user can choose ranges for the weights w_i, such that the algorithm focuses on a selected region of the pareto front (see e.g. the blue line in Figure 3a).
For the case of $k = 2$ objectives, the weight vector $w \in [0; 1]^k$ can range from $(1, 0)$ (only optimize first objective) to $(0, 1)$ (only optimizing the second objective). By limiting w to $[l, 1 - l] \times [u, 1 - u]; 0 < l < u < 1$ we can effectively limit the possible trade-offs we might be willing to make.

4 Application: A Fair Model for Income Prediction

Income prediction of employers is a versatile use-case for the application of multi-criteria AutoML, as several important criteria for a model can be derived. While trying to minimize the missclassification error, moral and ethical principles must also be adhered to. Thus, a model cannot be biased or unfair towards different sub-populations, e.g. men can not be systematically favoured over women regarding their income. As a third possible criterion, we might require an interpretable model, as a model might need to be accepted by regulatory bodies.

We use a fairness measure described in section 2.2, namely the absolute difference in F1-Scores between two sub-populations *male* and *female*. In order to start the AutoML process, we simply need to specify our tuning budget by either setting the number of MBO-iterations or the desired time for tuning. To get a first impression of the pareto front, we start with a tuning budget of only 20 iterations. Figure 3a shows the resulting pareto front. We can now use this, to focus the search towards trade-offs we are interested in. This is done by limiting the range of projections available to **parEgo**. For a first investigation, we choose values between 0.1 and 0.9. Those lower and upper limits on the projections can be adapted throughout the process.

Afterwards, we can simply continue training with additional budget. Continuing this training twice, we can also access the final optimization path, which is shown in Figure 3b. For each chosen measure, it shows the achieved performance for each function evaluation, and the (single-criteria) optimum achieved. Finally, we observe the pareto fronts as illustrated in Figure 4 for final tuning after 20, 70 and 120 iterations.
We compare to Thomas et al. (2018), optimizing a single objective (mmce). Note that solely optimizing for Fairness is not sensible, as many models achieve a fairness score of 0 (on validation data). The user can then choose an optimal hyperparameter configuration from the pareto front which matches her preferences best. Table 1 displays different points from the pareto front as well as the single-objective method evaluated on test data.

	Valid.		Test	
Method	mmce	fairF1	mmce	fairF1
autoxgboost	**0.125**	-	0.165	0.038
PF (ours)	0.129	0.023	0.139	0.061
PF (ours)	0.129	0.020	0.131	0.064
PF (ours)	0.130	0.002	**0.130**	0.080
PF (ours)	0.131	0.001	0.159	0.059
PF (ours)	0.132	0.001	0.157	0.067
PF (ours)	0.133	0.001	0.156	0.060
PF (ours)	0.135	**0.000**	0.142	0.096
PF (ours)	0.137	**0.000**	0.147	0.059
PF (ours)	0.142	**0.000**	0.146	0.077
PF (ours)	0.158	**0.000**	0.284	0.116
PF (ours)	0.238	**0.000**	0.244	**0.000**

Table 1: Performances of models from the pareto-front on held-out test set. We compare solutions from the Pareto front (PF) to Thomas et al. (2018) (optimizing mmce) after 120 iterations.

5 Outlook

In this work we conduct a first investigation into AutoML systems that can optimize a machine learning pipeline with respect to many different criteria. We provide several

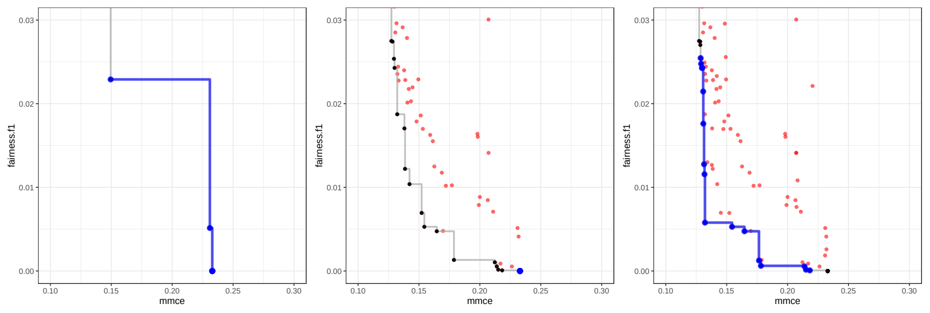


Fig. 4: Pareto fronts after each AutoxgboosMC run (20, 70, 120 tuning iterations). Zoomed in, in order to better show the pareto-front. The region we focus on is coloured in blue.

measures, that can be used as proxies for concepts such as *Fairness, Interpretability, Robustness* and others. Additionally, we implement a simplified AutoML system, that can optimize multiple objectives simultaneously, and can therefore serve as a tool for investigating such scenarios. The potential and necessity of our approach is demonstrated in a use-case.

The proposed method can be extended and improved in multiple directions. In a first iteration, we aim to include a wider array of gradient boosting methods, such as LightGBM (Ke et al. (2017)) and catboost (Dorogush et al. (2017)) into our framework. By combining this with a larger set of different pre- and post-processing methods, which can be tailored towards improving the different measures listed in section 2.2, we hope to obtain a toolbox that is suitable for many different situations where multiple criteria are required. Several interesting enhancements to the optimization procedure could also be made, either by adopting promising approaches from Bayesian Optimization (c.f. Paria et al. (2018)), or by adopting other search procedures.

The real underlying preferences a user has towards selecting a model might not always be easily quantify-able, because they rely on previous experience, implementation or other details. At the same time, a user can be asked to provide (*noisy*) labels for a set of models or to indicate preferences of one model over another (c.f González et al. (2017)). In future research, this might serve as an interesting avenue towards more human-centered AutoML. A third important part of research we aim to conduct is towards making AutoML methods more readily available to other user-groups, while at the same time providing them with sufficient tools to obtain models tailored towards the specific applications needs. In order to achieve this, we aim to research User Interfaces that make the AutoML more transparent to the user, while at the same time ensuring reproducibility.

Acknowledgements. This work has been funded by the German Federal Ministry of Education and Research (BMBF) under Grant No. 01IS18036A.

Bibliography

Apley, D. W. (2016). Visualizing the effects of predictor variables in black box supervised learning models. *arXiv preprint arXiv:1612.08468*.

Barocas, S., Hardt, M., and Narayanan, A. (2018). *Fairness and Machine Learning*. fairmlbook.org. `http://www.fairmlbook.org`.

Bischl, B., Lang, M., Kotthoff, L., Schiffner, J., Richter, J., Studerus, E., Casalicchio, G., and Jones, Z. M. (2016). mlr: Machine learning in R. *Journal of Machine Learning Research*, 17(170):1–5.

Bischl, B., Richter, J., Bossek, J., Horn, D., Thomas, J., and Lang, M. (2018). mlrMBO: A Modular Framework for Model-Based Optimization of Expensive Black-Box Functions.

Blot, A., Hoos, H. H., Vermeulen-Jourdan, L., Kessaci-Marmion, M.-É., and Trautmann, H. (2016). Mo-paramils: A multi-objective automatic algorithm configuration framework. In *LION*.

Bousquet, O. and Elisseeff, A. (2002). Stability and generalization. *J. Mach. Learn. Res.*, 2:499–526.

Chen, T. and Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA. ACM.

Dorogush, A. V., Ershov, V., and Gulin, A. (2017). Catboost: gradient boosting with categorical features support.

Everson, R. M. and Fieldsend, J. E. (2006). Multi-class roc analysis from a multi-objective optimisation perspective. *Pattern Recognition Letters*, 27(8):918–927.

Fan, Z., Fang, Y., Li, W., Lu, J., Cai, X., and Wei, C. (2017). A comparative study of constrained multi-objective evolutionary algorithms on constrained multi-objective optimization problems. In *2017 IEEE Congress on Evolutionary Computation (CEC)*, pages 209–216. IEEE.

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., and Hutter, F. (2015). Efficient and robust automated machine learning. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems 28*, pages 2962–2970. Curran Associates, Inc.

Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *Ann. Statist.*, 29(5):1189–1232.

González, J., Dai, Z., Damianou, A., and Lawrence, N. D. (2017). Preferential bayesian optimization. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1282–1291. JMLR. org.

Guyon, I., Sun-Hosoya, L., Boullé, M., Escalante, H. J., Escalera, S., Liu, Z., Jajetic, D., Ray, B., Saeed, M., Sebag, M., Statnikov, A., Tu, W.-W., and Viegas, E. (2019). *Analysis of the AutoML Challenge Series 2015–2018*, pages 177–219. Springer International Publishing, Cham.

Hakanen, J. and Knowles, J. D. (2017). On using decision maker preferences with parego. In *EMO*.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. (2009). The weka data mining software: An update. *SIGKDD Explor. Newsl.*, 11(1):10–18.

Handl, J., Kell, D. B., and Knowles, J. (2007). Multiobjective optimization in bioinformatics and computational biology. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 4(2):279–292.

Hardt, M., Price, E., Srebro, N., et al. (2016). Equality of opportunity in supervised learning. In *Advances in neural information processing systems*, pages 3315–3323.

Hernández-Lobato, J. M., Gelbart, M. A., Adams, R. P., Hoffman, M. W., and Ghahramani, Z. (2016). A general framework for constrained bayesian optimization using information-based search. *The Journal of Machine Learning Research*, 17(1):5549–5601.

Horn, D. and Bischl, B. (2016). Multi-objective parameter configuration of machine learning algorithms using model-based optimization. In *Computational Intelligence (SSCI), 2016 IEEE Symposium Series on*, pages 1–8. IEEE.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861.

Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., Fischer, I., Wojna, Z., Song, Y., Guadarrama, S., and Murphy, K. (2016). Speed/accuracy trade-offs for modern convolutional object detectors. *CoRR*, abs/1611.10012.

Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2011). *Sequential Model-Based Optimization for General Algorithm Configuration*, pages 507–523. Springer Berlin Heidelberg, Berlin, Heidelberg.

Jin, Y. and Sendhoff, B. (2008). Pareto-based multiobjective machine learning: An overview and case studies. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(3):397–415.

Johnson, J., Douze, M., and Jégou, H. (2017). Billion-scale similarity search with gpus. *CoRR*, abs/1702.08734.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T.-Y. (2017). Lightgbm: A highly efficient gradient boosting decision tree. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30*, pages 3149–3157. Curran Associates, Inc.

Knowles, J. (2004). Parego: A hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. Technical Report TR-COMPSYSBIO-2004-01, University of Manchester.

Lange, T., Braun, M. L., Roth, V., and Buhmann, J. M. (2003). Stability-based model selection. In *Advances in neural information processing systems*, pages 633–642.

Miettinen, K. and Mäkelä, M. M. (2002). On scalarizing functions in multiobjective optimization. *OR spectrum*, 24(2):193–213.

Molnar, C. (2019). *Interpretable Machine Learning*. `https://christophm.github.io/interpretable-ml-book/`.

Molnar, C., Casalicchio, G., and Bischl, B. (2019). Quantifying interpretability of arbitrary machine learning models through functional decomposition. *arXiv preprint arXiv:1904.03867*.

Olson, R. S., Urbanowicz, R. J., Andrews, P. C., Lavender, N. A., Kidd, L. C., and Moore, J. H. (2016). Automating biomedical data science through tree-based pipeline optimization. In Squillero, G. and Burelli, P., editors, *Applications of Evolutionary Computation*, pages 123–137, Cham. Springer International Publishing.

Papernot, N., McDaniel, P. D., Wu, X., Jha, S., and Swami, A. (2015). Distillation as a defense to adversarial perturbations against deep neural networks. *CoRR*, abs/1511.04508.

Paria, B., Kandasamy, K., and Póczos, B. (2018). A flexible multi-objective bayesian optimization approach using random scalarizations. *CoRR*, abs/1805.12168.

Pleiss, G., Raghavan, M., Wu, F., Kleinberg, J., and Weinberger, K. Q. (2017). On fairness and calibration. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pages 5684–5693, USA. Curran Associates Inc.

Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'12, pages 2951–2959, USA. Curran Associates Inc.

Steuer, R. E. and Choo, E.-U. (1983). An interactive weighted tchebycheff procedure for multiple objective programming. *Math. Program.*, 26(3):326–344.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. (2013). Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*.

Thomas, J., Coors, S., and Bischl, B. (2018). Automatic gradient boosting. In *International Workshop on Automatic Machine Learning at ICML*.

Thornton, C., Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2013). Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *Proc. of KDD-2013*, pages 847–855.

Wang, Q., Ming, Y., Jin, Z., Shen, Q., Liu, D., Smith, M. J., Veeramachaneni, K., and Qu, H. (2019). Atmseer: Increasing transparency and controllability in automated machine learning. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, pages 681:1–681:12, New York, NY, USA. ACM.

Wilson, J., Meher, A. K., Bindu, B. V., Sharma, M., Pareek, V., Chaudhury, S., and Lall, B. (2018). Autogbt:automatically optimized gradient boosting trees for classifying large volume high cardinality data streams under concept-drift. `https://github.com/flytxtds/AutoGBT`.

Zhang, K., Schölkopf, B., Muandet, K., and Wang, Z. (2013). Domain adaptation under target and conditional shift. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, pages III–819–III–827. JMLR.org.

Zhang, T., Georgiopoulos, M., and Anagnostopoulos, G. C. (2015). Sprint multi-objective model racing. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, GECCO '15, pages 1383–1390, New York, NY, USA. ACM.

Žliobaitė, I. (2017). Measuring discrimination in algorithmic decision making. *Data Mining and Knowledge Discovery*, 31(4):1060–1089.

4.6 Automated Benchmark-Driven Design and Explanation of Hyperparameter Optimizers

Contributed Article:
J. Moosbauer, M. Binder, L. Schneider, F. Pfisterer, M. Becker, M. Lang, L. Kotthoff, and B. Bischl. Automated benchmark-driven design and explanation of hyperparameter optimizers. *To appear in IEEE Transactions on Evolutionary Computation*, 2022

Declaration of contributions The idea for the algorithmic framework (SMASHY) considered in the project originated from MB and BB, with input from JM. The idea to put it in context as automatic configuration and analysis of algorithm components for hyperparameter optimizers originated from JM and MB. The code for the algorithmic framework was developed by MB based on a previous project[3]. Code for the experiments for algorithm configuration was written by MBe and MB. Benchmarks of competing state-of-the-art algorithm implementations were conducted by JM. Code for the experiments for algorithm analysis was written by LS (ablation studies), MB, and JM. The manuscript was written by JM and MB, with refinement from LS, LK, MBe, FP, and BB. LS, MB, FP, LS, ML, LK advised throughout the whole project.
The article is available under a Creative Commons License under the DOI: `https://doi.org/10.1109/TEVC.2022.3211336`.

[3]https://github.com/mlr-org/miesmuschel

Automated Benchmark-Driven Design and Explanation of Hyperparameter Optimizers

Julia Moosbauer*, Martin Binder*,
Lennart Schneider, Florian Pfisterer, Marc Becker, Michel Lang, Lars Kotthoff, Bernd Bischl

Abstract—**Automated hyperparameter optimization (HPO) has gained great popularity and is an important component of most automated machine learning frameworks.**

However, the process of designing HPO algorithms is still an unsystematic and manual process: New algorithms are often built on top of prior work, where limitations are identified and improvements are proposed. Even though this approach is guided by expert knowledge, it is still somewhat arbitrary. The process rarely allows for gaining a holistic understanding of which algorithmic components drive performance and carries the risk of overlooking good algorithmic design choices.

We present a principled approach to automated benchmark-driven algorithm design applied to multi-fidelity HPO (MF-HPO). First, we formalize a rich space of MF-HPO candidates that includes, but is not limited to, common existing HPO algorithms and then present a configurable framework covering this space. To find the best candidate automatically and systematically, we follow a programming-by-optimization approach and search over the space of algorithm candidates via Bayesian optimization. We challenge whether the found design choices are necessary or could be replaced by more naive and simpler ones by performing an ablation analysis. We observe that using a relatively simple configuration (in some ways, simpler than established methods) performs very well as long as some critical configuration parameters are set to the right value.

Index Terms—**Algorithm design, algorithm analysis, hyperparameter optimization, multifidelity, automated machine learning**

I. Introduction

Machine learning (ML) is, in many regards, an optimization problem, and many ML methods can be expressed as algorithms that perform loss minimization with respect to a given objective function. The higher-level task of selecting the ML method and its configuration is often framed as an optimization problem as well, sometimes referred to as a *hyperparameter optimization* (HPO) [1] or *combined algorithm selection and hyperparameter optimization* (CASH) problem [2]. Successfully addressing this problem can lead to large performance gains compared to simply using defaults, and in the context of automated machine learning (AutoML), the use of HPO can make ML more accessible to non-experts. Because of their potential benefits to ML performance and usability, it is of particular interest to design optimization algorithms that perform particularly well on the HPO problem.

Optimization problems arise in many fields of science and engineering, but as the no-free-lunch theorem states, there is no one optimization algorithm that solves all problems equally well [3]. To design suitable optimizers, it is therefore important to understand the characteristics of HPO:

- **Black-box**: The objective usually provides no analytical information [4] – such as a gradient. Thus, the application of many traditional optimization methods – such as BFGS – is rendered inappropriate or at least questionable.
- **Complex search space**: The search space of the optimization problem is often high-dimensional and may contain continuous, integer-valued and categorical dimensions. Often, there are dependencies between dimensions or even specific hyperparameter values [5].
- **Expensive**: A single evaluation of the objective function may take hours or days. Thus, the total number of possible function evaluations is often severely limited [4].
- **Low-fidelity approximations possible**: An approximation of the true objective value at lower expense can often be obtained, for example, through a partial evaluation [6].
- **Low effective dimensionality**: The landscape of the objective function can usually be approximated well by a function of a small subset of all dimensions [7].

Recent HPO and AutoML research has focused on finding and improving optimization algorithms that work particularly well under these conditions. A common approach is to tackle HPO by estimating a local or global structure of the objective landscape by some form of predictive model. This introduces additional overhead and complexity with the aim of reducing the overall number of expensive objective evaluations necessary to find an approximate optimum. Typical representatives of this approach are Bayesian optimization (BO) [8] algorithms and frameworks based on BO, which are global optimization schemes based on a non-linear regression model, e.g., a Gaussian process or random forest. They have shown significant improvements in performance compared to other methods [9] but carry a significant overhead. Furthermore, BO is somewhat difficult to parallelize due to its sequential nature, although many variants exist (e.g. [10]–[13]).

Multi-fidelity HPO (MF-HPO) algorithms aim to accelerate the optimization process by exploiting cheaper proxy functions of the objective function itself (e.g., by training ML models on a smaller subsample of the available training data, or by running fewer training iterations). Bandit-based algorithms like Hyperband (HB) [14] have become particularly popular because of their good trade-off between optimization performance and simplicity.

Progress in the field of HPO often consists of iterative improvements of established algorithms. Considerable work exists, for example, to improve the limitations of HB: Asynchronous successive halving (ASHA) [15] proposes a sophisticated way to make efficient use of parallel resources, BO Hyperband (BOHB) [16] improves performance during later parts of a run by incorporating surrogate assistance into HB, and asynchronous BOHB (A-BOHB) [17] unites a bandit-based optimization scheme using model-based guidance with asynchronous parallelization.

*Equal Contribution

While these conceptual extensions of HPO all have their respective merit, it is often somewhat overlooked that the simplicity of an optimization algorithm (i.e., how difficult modifications and extensions are, and on how many dependencies a system relies [18]) heavily influences its adoption in practice. Random search (RS), for example, still enjoys great popularity, as it is extremely simple to implement and parallelize, has almost no overhead, and is able to take advantage of the aforementioned low effective dimensionality [7]. Furthermore, algorithmic developments identify and address limitations of prior research, but rarely question core algorithmic choices that have been made in the original implementation. Many multi-fidelity algorithms, for example, are extensions and further developments of HB that take the fixed successive halving schedule [19] for granted. The process of designing a good MF-HPO optimizer in practice – and many other algorithmic solutions in science in general – can therefore often feel somewhat like a "manual stochastic local search on the meta level". The drawback of this manual procedure is that the design space of all HPO algorithms is not systematically searched, and parts of the design space are excluded by prior algorithmic decisions. If "established" algorithms are not challenged, there is a risk that algorithms that work well will be overlooked., and it is often hard to identify what algorithmic components make a difference. In particular, it is possible that overly complicated algorithms are developed by extending "established" designs, only some of which contribute meaningfully to performance gains. Sometimes certain technical components of an algorithm, which are neither exposed nor discussed in detail, may also influence performance significantly.

A. Contributions

We make a principled demonstration of how HPO algorithm design can be performed systematically and automatically with a benchmark-driven approach following the programming-by-optimization paradigm [20]. In particular, the contributions of this work are:

- **Formalization**: We formalize the design space of MF-HPO algorithms and demonstrate that established MF-HPO algorithms represent instances within this space.
- **Framework**: Based on this formalization, we present a rich, configurable framework for MF-HPO algorithms, whose software implementation we call SMASHY (Surrogate Model Assisted HYperband).
- **Configuration**: Based on the formalization and framework, we follow an empirical approach to design an MF-HPO algorithm by optimization, given a large benchmark suite. This configuration procedure does not only consider performance, but also, e.g., the simplicity of the design.
- **Benchmark**: As in general any HPO algorithm will be applied in a diverse set of application scenarios, we evaluate the performance of our newly designed algorithm on a representative set of problems that were not previously used for its configuration (i.e., a clean test-set approach on the meta-level) and compare them with established implementations of HPO methods.
- **Explanation**: For the resulting MF-HPO system, we systematically assess and explain the effect of different design choices on overall algorithmic performance. Furthermore, we investigate the behavior of algorithmic design components in the context of specific problem scenarios; i.e., we investigate which algorithmic components lead to performance improvements for simple HPO with numeric hyperparameters, AutoML pipeline configuration, and neural architecture search.

II. Related Work

HPO is one of the most essential components of current AutoML methods [1], and MF-HPO has recently become more prominent, given that cheap, low-fidelity evaluations have proven useful to speed up optimization, especially for expensive HPO of complex ML algorithms on larger data sets [14]. While AutoML tools have historically relied on a limited set of HPO methods, we argue that the optimal HPO method depends on problem characteristics, and therefore a systematic development of HPO methods under consideration of problem characteristics is required. Approaches towards such systematic development have often relied on a *high-level* language or template that allows expressing solutions to a given problem, e.g. to solve constraint satisfaction problems [21]–[23], satisfiability problems [24], or scheduling problems [25].

Even if a high-level language is available, manual configuration of such frameworks is laborious and requires expert knowledge. This motivates the design philosophy of "Programming by Optimization" [20] (PBO), which advocates for allowing algorithmic choices in a software system (instead of fixing them at the time of implementation) and automatic configuration by optimization for a given problem context.

As one approach to automatic and efficient algorithm configuration, racing-based strategies have been used to design optimization algorithms. For example, F-RACE [26] has been used for the automatic design of multi-objective ant colony optimization algorithms [27]. Similarly, IRACE [28] has been used for the automatic design multi-objective evolutionary algorithms [29] or to meta-configure the parameters IRACE itself [30]. Another commonly used framework is SMAC [5], which extends the sequential model-based optimization paradigm (SMBO, see also Section IV-A2) to an algorithm configuration setting. This is achieved through the use of an intensification procedure that governs across how many problem instances each configuration is evaluated, trading off computational cost against confidence regarding the superiority of a given configuration. Furthermore, instance features describing properties of a problem instance are used to train the empirical performance model predicting the performance of a configuration on a new problem instance. For example, SMAC has been used by the authors of the SATenstein [24] framework to automatically tailor SATenstein to particular sets of problems. Besides racing and sequential model-based approaches, genetic algorithms have also been used to evolve optimal solvers [31].

We argue that the design of HPO algorithms can be seen as an instance of PBO. However, while there are many approaches that focus on individual algorithmic choices (e.g., the choice of a surrogate model for BO [32]), we are not aware of many cases where PBO is applied to designing HPO systems themselves. One exception is [33], who use SMACv3 [34] to automatically configure Bayesian optimization (BO) for HPO from a flexible search space of components. We take a similar approach here in that the algorithmic choices are exposed as hyperparameters that can be tuned. However, unlike [33], we do not configure an established HPO method (such as BO) with a predefined structure and associated control parameters (e.g., varying the surrogate model of BO). Instead, we introduce a *new* configurable algorithmic framework, which covers many different MF-HPO structures, including well-established principles for multi-fidelity handling (e.g., Successive Halving) as well as new approaches (e.g., equal batch size in all proposals).

In addition to designing well-performing algorithms, it is equally important to facilitate an understanding of the effects of all considered design choices. The field of *sensitivity analysis* (SA) comprises a multitude of methods to assess the importance of input factors on the output of a mathematical model [35]. Functional ANOVA (fANOVA) methods, which decompose the response of a (mathematical) model or function into lower-order components, are a widely studied method in the field of SA, dating back to [36]. This class of methods has also become popular in the field of ML to analyze the importance of hyperparameters [37].

Popular ways of analyzing effects of algorithmic effects in ML and algorithm configuration are *ablation studies* [38]. This involves measuring the performance when removing one or more of algorithmic subcomponents to understand the relative contribution of the ablated components to overall performance. There are different ways of performing an ablation analysis; probably the most common approach is *leave-one-component-out* (LOCO) ablation [39]. In the context of algorithm configuration, [38] proposes an ablation approach that links a source configuration (e.g., the default) to a target (e.g., the optimized configuration) through an ablation path.

Nevertheless, many existing works that propose or improve HPO or algorithm configuration systems do not analyze the algorithmic choices of an optimized system, and the ones that do perform relatively straightforward analyses. For example, [21] compare the designs their approach finds automatically to the designs expert humans generated. [40] perform ANOVA and non-parametric Friedman tests to investigate in detail the effects that algorithmic choices have for ant colony optimization algorithms. [41] incrementally investigate the effect of different design choices for ModCMA as individual components are changed.

III. Methodology

A. Supervised Machine Learning

Supervised ML typically deals with a dataset (which is, mathematically speaking, a tuple) $\mathcal{D} = \left((\mathbf{x}^{(i)}, y^{(i)})\right) \in (\mathcal{X} \times \mathcal{Y})^n$ of n observations, assumed to be drawn i.i.d. from a data-generating distribution $\mathbb{P}_{xy}$. An ML model is a function $\hat{f} : \mathcal{X} \to \mathbb{R}^g$ that assigns a prediction to a feature vector from $\mathcal{X}$.[1] $\hat{f}$ is itself constructed by an *inducer* function $\mathcal{I}$, i.e., the model-fitting algorithm. The inducer $\mathcal{I} : (\mathcal{D}, \boldsymbol{\lambda}) \mapsto \hat{f}$ uses training data $\mathcal{D}$ and a vector of *hyperparameters* $\boldsymbol{\lambda} \in \Lambda$ that govern its behavior. The overall goal of supervised ML is to derive a model $\hat{f}$ from a data set $\mathcal{D}$ so that $\hat{f}$ predicts data sampled from $\mathbb{P}_{xy}$ best. The quality of a prediction is measured as the discrepancy between predictions and ground truth. This is operationalized by the loss function $L : \mathcal{Y} \times \mathbb{R}^g \to \mathbb{R}_0^+$, which is to be minimized during model fitting. In contrast to the optimisation problems that we will define in Sections III-B and III-C, we term this the "first level" optimisation problem.

[1]where g allows handling of multi-output regression, as well as multiclass classification with g classes by returning decision scores.

The expectation of the loss value of predictions made for data samples drawn from $\mathbb{P}_{xy}$ is the *generalization error*

$$GE := \mathbb{E}_{(\mathbf{x},y)\sim\mathbb{P}_{xy}}\left[L(y, \hat{f}(x))\right] \tag{1}$$

which cannot be computed directly if $\mathbb{P}_{xy}$ is not known beyond the available data $\mathcal{D}$. Therefore, one often uses so-called *resampling* techniques that fit models on N_{iter} subsamples $\mathcal{D}[J_j]$ and evaluate them on complements $\mathcal{D}[-J_j]$ of these subsets to obtain an estimate of the generalization error

$$\widehat{GE}(\mathcal{I}, \boldsymbol{\lambda}, \mathbf{J}) = \frac{1}{N_{\text{iter}}} \sum_{j=1}^{N_{\text{iter}}} L\big(y[-J_j], \mathcal{I}\left(\mathcal{D}[J_j], \boldsymbol{\lambda}\right)(x[-J_j])\big). \tag{2}$$

Depending on the resampling method, the inducer $\mathcal{I}$, and the quantity of data in $\mathcal{D}$, estimating the generalization error $\widehat{GE}(\mathcal{I}, \boldsymbol{\lambda}, \mathbf{J})$ can require large amounts of computational resources.

B. Hyperparameter Optimization

The goal of HPO is to identify a hyperparameter configuration that performs well in terms of the estimated generalization error in Equation (2). Often, optimization only concerns a subspace of available hyperparameters because some hyperparameters might be set based on prior knowledge or due to other constraints. One would therefore split up the space of hyperparameters Λ into a subspace of hyperparameters Λ_S over which optimization takes place, and the remaining hyperparameters $\Lambda_C = \Lambda / \Lambda_S$ for which values $\boldsymbol{\lambda}_C$ are given exogenously. We define the HPO problem as:

$$\boldsymbol{\lambda}_S^* \in \operatorname*{argmin}_{\boldsymbol{\lambda}_S \in \Lambda_S} c(\boldsymbol{\lambda}_S) = \operatorname*{argmin}_{\boldsymbol{\lambda}_S \in \Lambda_S} \widehat{GE}(\mathcal{I}, (\boldsymbol{\lambda}_S, \boldsymbol{\lambda}_C), \mathbf{J}). \tag{3}$$

Here, $\boldsymbol{\lambda}_S^*$ denotes a theoretical optimum, and $c(\boldsymbol{\lambda}_S)$ is a shorthand for the estimated generalization error in Equation (2). We refer to Problem 3 as the "second level" optimisation problem.

Hyperparameters can be either continuous, discrete, or categorical, and search spaces are often a mix of the different types. The search space may be hierarchical, i.e., some subordinate hyperparameters can only be set in a meaningful way if another parent hyperparameter takes a certain value. In particular, many AutoML frameworks perform optimization over a hierarchical hyperparameter space that represents the components of a complex ML pipeline [1].

Many HPO algorithms can be characterized by how they handle two different trade-offs: (a) The exploration vs. exploitation trade-off refers to how much budget an optimizer spends on either trying to directly exploit the currently available knowledge base by evaluating very close to the currently best candidates (e.g., local search) or whether it explores the search space to gather new knowledge (e.g., random search). (b) The inference vs. search trade-off refers to how much time and overhead is spent to induce a model from the currently available archive data in order to exploit past evaluations as much as possible. Other relevant aspects that HPO algorithms differ in are: *Parallelizability*, i.e., how many configurations a tuner can (reasonably) propose at the same time; *global vs. local* behavior of the optimizer, i.e., if updates are always quite close to already evaluated configurations; *noise handling*, i.e., if the optimizer takes into account that the estimated generalization error is noisy; *search space complexity*, i.e., if and how hierarchical search spaces can be handled; *multi-fidelity*, i.e., if the optimizer uses cheaper evaluations to infer performance on the full data.

Multi-fidelity methods make use of the fact that the resampling procedure in Equation (2) can be modified in multiple ways to make evaluation cheaper: one can (i) reduce the training sizes $|J_j|$ via subsampling, as model evaluation complexity is often at least linear in training set size, or (ii) change some components in $\boldsymbol{\lambda}$ in a way that makes model fits cheaper. Examples of (ii) are reducing the overall number of training cycles performed by a neural network fitting process, or reducing the number of base learner fits in a bagging or boosting method. These modifications can both increase the variance of $\widehat{GE}$ and introduce an (often pessimistic) bias, as models trained on smaller datasets or with values of $\boldsymbol{\lambda}$ that make fitting cheaper often have worse generalization errors.

We introduce a *fidelity* parameter $r \in (0, 1]$ that influences the resource requirements of the evaluation of $\widehat{GE}$ and define

$$c(\boldsymbol{\lambda}_S; r) := \widehat{GE}\left(\mathcal{I}, (\boldsymbol{\lambda}_S, \boldsymbol{\lambda}_C(r)), \mathbf{J}(r)\right). \tag{4}$$

With this definition we make the choice that r should influence the evaluation cost of $\widehat{GE}$ only by modifying the resampling, $J(r)$ or by modifying a hyperparameter $\boldsymbol{\lambda}_C(r)$. Typically, r only affects one of these aspects at a time, and if it affects $\boldsymbol{\lambda}_C$, it only affects a single hyperparameter dimension.

Note that we normally assume that a higher fidelity r returns a better model in terms of the estimate of the generalization error, and the best estimate is returned for $r = 1$. Therefore, r enters the expression in a way where it can influence performance, but is not searched over. We define $c(\boldsymbol{\lambda}_S) := c(\boldsymbol{\lambda}_S; 1)$ as in [42], and the optimization problem remains as in Equation (3).

This assumption may be violated in some scenarios, and model performance could worsen for a higher value of r (e.g., a neural network, which may overfit on a small dataset if trained for too many epochs). In this case, we define the optimization problem as $(\boldsymbol{\lambda}_S^*, r^*) \in \mathsf{argmin}_{\lambda_S \in \Lambda_S, r \in (0,1]} c(\lambda_S; r)$.

The resource requirements of evaluating $c(\boldsymbol{\lambda}; r)$ can have a complicated relationship with $\boldsymbol{\lambda}$ and r; in practice, r is chosen in such a way that it has an overwhelming and linear influence on resource demand. The overall cost of optimization up to a given point in the optimization process is therefore assumed to be the cumulative sum of the values of r of all evaluations of $c(\boldsymbol{\lambda}; r)$ up to that point. We can also interpret r as the fraction of the budget of a single full fidelity model evaluation that must be spent for evaluating $c(\boldsymbol{\lambda}; r)$.

Given the definition of the HPO problem, we present an (MF-)HPO algorithm for a single, synchronous worker in its most generic form in Algorithm 1. Until a pre-determined budget is exhausted, such an algorithm decides in every iteration (a) which configuration(s) $\boldsymbol{\lambda}_S$ to evaluate next and (b) which fidelity r to use for evaluation; non-multi-fidelity algorithms set this to $r = 1$ as default. The algorithm makes use of an *archive* $\mathcal{A}$, a database recording previously proposed hyperparameter configurations and, if available, their evaluation results. This database can be shared among multiple worker processes that optimize concurrently.

Algorithm 1 A generic HPO algorithm

```
1: while budget is not exhausted do
2:     Propose $\left(\boldsymbol{\lambda}_S^{(i)}, r^{(i)}\right)$, $i = 1, ..., k$, based on archive $\mathcal{A}$
3:     Write proposals into a shared archive $\mathcal{A}$
4:     Estimate generalization error(s) $c\left(\boldsymbol{\lambda}_S^{(i)}; r^{(i)}\right)$
5:     Write results into shared archive $\mathcal{A}$
6: end while
7: Wait for workers to synchronize
8: Return best configuration in archive $\mathcal{A}$
```

The optimization process can be accelerated by making efficient use of parallel resources. We distinguish between *synchronous* and *asynchronous* scheduling. The former starts multiple evaluations synchronously at the same time and waits until all of these have finished. To be more precise, a number of $k > 1$ configurations is proposed in line 2 and evaluated in parallel in line 4, all within the inner loop of Algorithm 1. Given K available parallel resources, it should be ensured that the number k of configurations scheduled in parallel is not significantly smaller than K and that the evaluation runtimes amongst these k configurations do not differ significantly in order to avoid unnecessarily idling single parallel resources. In contrast, for asynchronous scheduling, Algorithm 1 is run individually in K separate worker processes. Given a shared archive that is synchronized between the workers, every worker can independently schedule new configurations to evaluate.

C. Algorithm Design and Configuration

Our goal will be to design and configure a new HPO algorithm based on a superset of design choices included in previously published HPO methods. We are interested in finding a configuration (or making design choices) based on a set of training instances that works across a broad set of future problem instances. This problem is called *algorithm configuration* [5], [43]. It is quite similar to HPO; a major difference is that algorithm configuration optimizes the configuration of an arbitrary algorithm over a diverse set of often heterogeneous instances for optimal average performance, while HPO performs a per-instance configuration of an ML inducer for a single data set. We introduce the following notation for consistency with the relevant literature: $\boldsymbol{\gamma}$ denotes configuration parameters controlling our optimizer A, while $\boldsymbol{\lambda}$ denotes hyperparameters optimized by our optimizer, controlling our inducer $\mathcal{I}$. The algorithm configuration problem can be formally stated as follows: Given an algorithm $A : \Omega \times \Gamma \to \Lambda$ parametrized by $\boldsymbol{\gamma} \in \Gamma$ and a distribution $\mathbb{P}_\Omega$ over problem instances Ω together with a cost metric ζ, we must find a parameter setting $\boldsymbol{\gamma}^*$ that minimizes the expected $\zeta(A)$ over $\mathbb{P}_\Omega$:

$$\boldsymbol{\gamma}^* \in \underset{\boldsymbol{\gamma} \in \Gamma}{\mathsf{argmin}}\, \mathbb{E}_{\omega \sim \mathbb{P}_\Omega}\left[\zeta(A(\omega, \boldsymbol{\gamma}))\right]. \tag{5}$$

In our example, Γ corresponds to the space of possible components of our HPO method and Ω to a class of HPO problems (i.e., ML methods and datasets on which they are evaluated) for which their configuration should be optimal. Based on a training set of representative instances $\{\omega_i\}$ drawn from $\mathbb{P}_\Omega$, a configuration $\boldsymbol{\gamma}^*$ that minimizes c across these instances should be chosen through optimization. When necessary, we refer to this process as the "third level" optimization problem to distinguish it from the optimization performed by the HPO algorithm A, i.e., the second level optimization.

IV. Formalizing a broad class of MF-HPO Algorithms

We aim to find an HPO algorithm that performs particularly well in the multi-fidelity setting. To design an algorithm by optimization, we propose a framework and search space of HPO algorithm candidates that covers a large class of possible algorithms and focus on a subclass of algorithms similar to Hyperband because of their favorable properties. This subclass focuses on multi-fidelity algorithms that use a pre-defined schedule of geometrically increasing fidelity evaluations containing algorithms like Hyperband [14] and BOHB [16].

The basis of this framework is presented in Algorithm 2, which can be configured by combining algorithmic building blocks in novel ways. The main difference to Algorithm 1 is that the *Propose* part is specified more explicitly. At its core, Algorithm 2 consists of two parts: (i) sampling new configurations at low fidelities (lines 2–7) and (ii) increasing the fidelity for existing configurations (lines 8–14). In contrast to Algorithm 1, Algorithm 2 makes use of state variables t, b, and r to account for optimization progress. However, these variables are only shown in Algorithm 2 for clarity and can, in principle, be inferred from the archive $\mathcal{A}$. As argued in Section III, every single worker instance of Algorithm 1 can, in principle, be scheduled asynchronously, but we do not consider this in this work.

In its first iteration, Algorithm 2 uses a SAMPLE-subroutine to initialize the initial batch C of μ solution candidates. The fidelity of the evaluation of the proposed configurations is refined iteratively; when all configurations in the batch have been evaluated with given fidelity r, the top $1/\eta_{\text{surv}}$ fraction of configurations is evaluated with a fidelity that is increased by a factor of η_{fid}. When the fidelity cannot be further increased for a batch because all of its configurations were evaluated at full fidelity $r = 1$, they are set aside, and a new batch of configurations is sampled.

The SAMPLE subroutine creates new configurations to be evaluated, possibly using information from the archive to propose points that are likely to perform well. We allow that any inducer $\mathcal{I}_{f_{\text{sur}}}$ that produces a surrogate model f_{sur} can be used for model-assisted sampling. The subroutine works by at first sampling a number of points from a given generating distribution $\mathbb{P}_{\boldsymbol{\lambda}}(\mathcal{A})$. The performance of these points is then predicted using the surrogate model, and points with unfavorable predictions are discarded in a process we refer to as *filtering*. This process is repeated until the requested number μ of non-discarded points is obtained. N_{s} and ρ have the same function as in [16] (see Section IV-A5), with the filter factor N_{s} controlling the number of sampled points needed for each of the μ points returned, and ρ controlling the fraction of points that are not filtered. Thus, the configuration space of sampling methods also includes purely random sampling, as in Hyperband, by setting $\rho = 1$. The influence of the surrogate model on sampled candidates is larger when (i) the number of sampled configurations N_{s} is large, or (ii) the fraction ρ of candidates sampled at random is small. We present two slightly different SAMPLE algorithms: SAMPLETOURNAMENT (Algorithm 3) and SAMPLEPROGRESSIVE (Algorithm 4) based on this principle (see Appendix A). Both allow to use different N_{s} values for different points they sample, parameterized by N_{s}^0 and N_{s}^1.

While hyperparameters $\boldsymbol{\lambda}_S$ are proposed by one of the two SAMPLE methods, the fidelity hyperparameter r follows a fixed schedule similar to Successive Halving [19] and Hyperband [14], with a few extensions. For one, the survivor factor η_{surv} can be a different value from the fidelity scaling factor η_{fid}. Furthermore, the algorithm allows three scheduling modes, controlled by *batch_method*: `SH` does Successive Halving. The `HB` mode evaluates brackets, as performed by Hyperband. While $\mu(b)$ is, in principle, a free configuration parameter for every value of b, we choose to set $\mu(b)$ so that total budget expenditure is approximately equal between all brackets. This follows the principle used in Hyperband, but the dependency on η_{surv} and η_{fid} is more complex and determined dynamically. Finally, `equal` *batch_method* uses equal batch sizes for every evaluation. Individuals that perform badly at low fidelity are removed, as in SH, but new individuals are sampled to fill up batches to the original size. Because new individuals are added to the batches at all fidelity steps, it is not necessary to use brackets with different initial fidelities, and therefore, only a single repeating bracket $b = 1$ is used. The `equal` method is an original contribution of this work and was designed to be similar to `HB` while using parallel resources more efficiently; the two batch scheduling methods are illustrated in Figure 1.

If the exploration-exploitation tradeoff is not balanced properly, the optimization progress can either stagnate or function evaluations are wasted due to too much exploration of uninteresting regions of the search space. However, the relative importance of exploration and exploitation can change throughout the course of optimization, where exploration performed later during the optimization is not as useful as during the beginning. The given configuration space makes it possible to make the exploration-exploitation tradeoff dependent on optimization progress by providing the option to make $\rho(t)$ and $\left(N_{\text{s}}^0(t), N_{\text{s}}^1(t)\right)$ dependent on the proportion of exhausted total budget at every configuration proposal step. It is likely that large values of $\rho(t)$ / small values of $N_{\text{s}}^{\cdot}(t)$ perform better when t is small. Conversely, it is likely that small $\rho(t)$ / large $N_{\text{s}}^{\cdot}(t)$ work well for large t.

A. Common MF-HPO Algorithms Covered by Algorithm 2

The following describes a few common HPO algorithms that can be instantiated within this framework; see Table I for specific configuration parameter settings within Algorithm 2 that correspond to these algorithms.

1) Random search (RS): Configurations $\boldsymbol{\lambda}_S$ are drawn (uniformly) at random, and every configuration is evaluated with full fidelity $r = 1$. Parallelization is straightforward, as configurations are drawn independently.

2) Bayesian Optimization (BO) [8]: The configuration that maximizes an acquisition function $a(\boldsymbol{\lambda})$ (e.g., expected improvement, EI [4]) is proposed and evaluated with the full fidelity $r = 1$. $a(\boldsymbol{\lambda})$ is based on a surrogate model trained on the archive $\mathcal{A}$. BO can be parallelized by either using methods that can propose multiple points at the same time using a single surrogate model or, alternatively, by fitting a surrogate model on the anticipated outcome of configurations that were proposed but not yet evaluated [11]. BO can be represented in Algorithm 2 by using an inducer $\mathcal{I}_{f_{\text{surr}}}$ that produces a function f_{surr} equal to the composition of model prediction and acquisition function. In its basic form, BO is not an MF algorithm and therefore always sets $r = 1$.

3) Successive halving (SH) [19]: Successive halving, also called Sequential Halving [44], is a simple multi-fidelity optimization algorithm that combines random sampling of configurations with a fixed schedule for r. At the beginning, a batch of μ configurations is sampled randomly and evaluated with an initial fidelity $r_{\text{min}} < 1$. This is

(a)

	$b = 1$		$b = 2$		$b = 3$		$b = 4$	
i	$\|C\|$	r	$\|C\|$	r	$\|C\|$	r	$\|C\|$	r
1	8	1/8	6	1/4	4	1/2	4	1
2	4	1/4	3	1/2	2	1		
3	2	1/2	1	1				
4	1	1						

(b)

	$b = 1$	
i	$\|C\|$	r
1	8	1/8
2	8	1/4
3	8	1/2
4	8	1

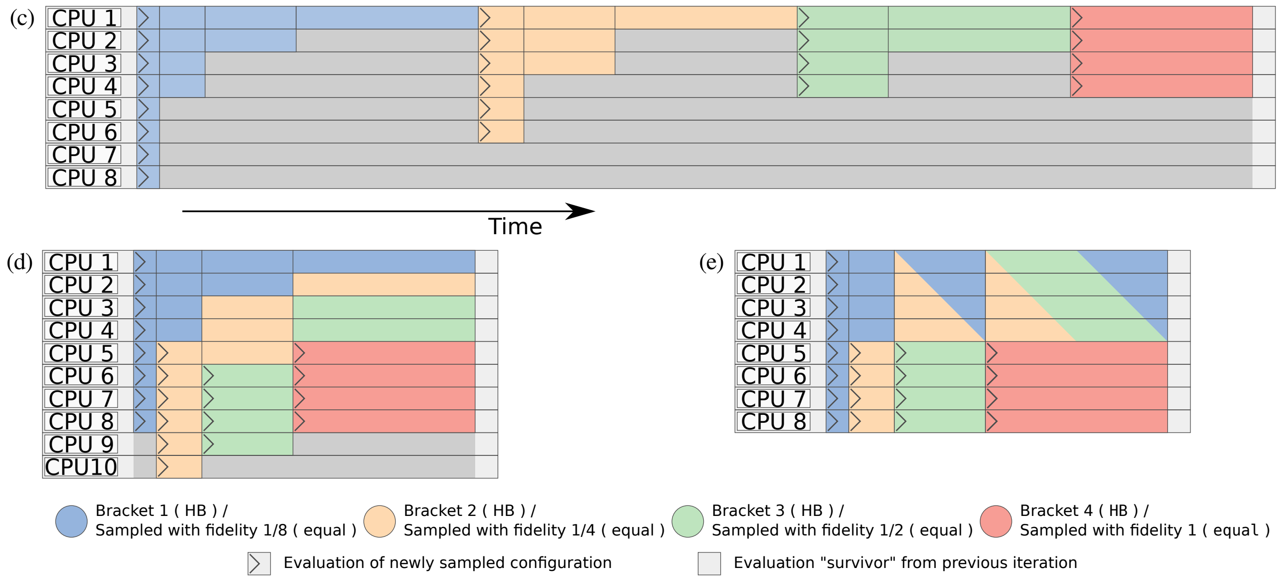

(c)
CPU 1
CPU 2
CPU 3
CPU 4
CPU 5
CPU 6
CPU 7
CPU 8
Time
(d)
CPU 1
CPU 2
CPU 3
CPU 4
CPU 5
CPU 6
CPU 7
CPU 8
CPU 9
CPU10
(e)
CPU 1
CPU 2
CPU 3
CPU 4
CPU 5
CPU 6
CPU 7
CPU 8
Bracket 1 (HB) / Sampled with fidelity 1/8 (equal)
Bracket 2 (HB) / Sampled with fidelity 1/4 (equal)
Bracket 3 (HB) / Sampled with fidelity 1/2 (equal)
Bracket 4 (HB) / Sampled with fidelity 1 (equal)
Evaluation of newly sampled configuration
Evaluation "survivor" from previous iteration

Fig. 1: Illustration of the different *batch_methods* used, corresponding to values of $\eta_{fid} = \eta_{surv} = 2$, $s = 4$, $\mu = 8$.
The tables show (a) the `HB` method and (b) the `equal` method. Shown are the number $|C|$ and fidelity value r of configurations being evaluated in the iterations i of the various brackets counted by b. Except for i, the variables are the same as in Algorithm 2.
Subfigures (c) - (e) illustrate resource utilization by the batch methods, given availability of parallel resources. (c): Naively scheduling the configuration evaluations one batch after another can make use of available parallel resources, but leaves many of them idle. (d): Hypothetical way of scheduling configuration evaluations of different brackets at the same time so that all configurations with the same r-value are scheduled together utilizes resources more efficiently, but the number of evaluations in each batch still varies. (e): The simpler `equal` batch scheduling method always evaluates the same number of configurations within each batch and, therefore, makes optimal use of available parallel resources.

TABLE I: RS, BO, SH, HB, BOHB as instances of Algorithm 2. η, ρ, N_s are configuration parameters of the respective algorithms.
"—" denotes that the value has no influence on the algorithm in this configuration.
*: BO and BOHB use inducers that produce non-standard model functions, which do not aim to predict the actual performance of configurations, and instead calculate the value of an acquisition function such as EI [4] (for BO) or the ratio of two kernel density estimator (KDE) models (for BOHB).
†: In a small departure from BOHB, Algorithm 2 uses the KDE estimate of good points for all sampled points, even when randomly interleaved. BOHB randomly interleaves from a uniform distribution.

Algorithm	$\mu(b)$	s	η_{surv}	η_{budget}	$\mathcal{I}_{f_{\text{sur}}}$	ρ	N_s	*batch_mode*	$\mathbb{P}_{\boldsymbol{\lambda}}(\mathcal{A})$
RS	—	1	—	—	—	1	—	—	uniform
BO	1	1	—	—	e.g. GP+EI*	ρ	N_s	—	uniform
SH	μ	$\lfloor -\log_\eta(r_{\min}) \rfloor + 1$	η	η	—	1	—	SH	uniform
HB	$\lceil s \cdot \frac{\eta^{s-b}}{s-b+1} \rceil$	$\lfloor -\log_\eta(r_{\min}) \rfloor + 1$	η	η	—	1	—	HB	uniform
BOHB	$\lceil s \cdot \frac{\eta^{s-b}}{s-b+1} \rceil$	$\lfloor -\log_\eta(r_{\min}) \rfloor + 1$	η	η	TPE*	ρ	N_s	HB	KDE†

Algorithm 2 SMASHY algorithm

Configuration Parameters: batch size schedule $\mu(b)$, number of fidelity stages s, survival rate η_{surv}, fidelity rate η_{fid}, SAMPLE method (either SAMPLETOURNAMENT or SAMPLEPROGRESSIVE), *batch_method* (one of `equal`, `SH`, or `HB`), total budget B; further configuration parameters of SAMPLE: $\mathcal{I}_{f_{\text{surr}}}$, $\mathbb{P}_{\boldsymbol{\lambda}}(\mathcal{A})$, $\rho(t)$, $\left(N_s^0(t), N_s^1(t)\right)$, n_{trn}.

State Variables: Expended budget fraction $t \leftarrow 0$, bracket counter $b \leftarrow 1$ (remains 1 for *batch_method* $\in$ {`equal, SH`}), current fidelity $r \leftarrow 1$, batch of proposed configurations $C \leftarrow \emptyset$

```
1:  while $t < 1$ do
2:      if $r = 1$ then                                   ▷ Generate new batch of configurations
3:          $r \leftarrow (\eta_{\text{fid}})^{b-s}$
4:          $C \leftarrow$ SAMPLE$\big(\mathcal{A}, \mu(b), r; \mathcal{I}_{f_{\text{sur}}}, \mathbb{P}_{\boldsymbol{\lambda}}(\mathcal{A}),$
                    $\rho(t), \left(N_s^0(t), N_s^1(t)\right), n_{\text{trn}}\big)$
5:          if batch_method = HB then
6:              $b \leftarrow (b \bmod s) + 1$
7:          end if
8:      else                                              ▷ Progress fidelity
9:          $r \leftarrow r \cdot \eta_{\text{fid}}$
10:         $C \leftarrow$ SELECT_TOP $(C, |C|/\eta_{\text{surv}})$
11:         if batch_method = equal then
12:             $\tilde{\mu} \leftarrow \mu(b) - |C|$
13:             $C \leftarrow C \cup$ SAMPLE$\big(\mathcal{A}, \tilde{\mu}, r; \mathcal{I}_{f_{\text{sur}}}, \mathbb{P}_{\boldsymbol{\lambda}}(\mathcal{A}),$
                    $\rho(t), \left(N_s^0(t), N_s^1(t)\right), n_{\text{trn}}\big)$
14:         end if
15:     end if
16:     Evaluate configuration(s) $c\left(\boldsymbol{\lambda}_S; r\right)$ for all $\boldsymbol{\lambda}_S \in C$
17:     Write results into shared archive $\mathcal{A}$
18:     $t \leftarrow t + r \cdot |C|/B$                   ▷ Update budget spent
19: end while
```

followed by repeated "halving" steps, where the top fraction η^{-1} of configurations is kept and evaluated after r is increased by a factor of η, until the maximum fidelity value is reached. The schedule is chosen to keep the total sum of all evaluated r constant in each batch. Both η_{surv} and η_{fid} in Algorithm 2 correspond to SH's η-parameter.

4) Hyperband (HB) [14]: Similar to SH, Hyperband uses a fixed schedule for the fidelity parameter r, but it augments SH by using multiple *brackets* b of SH runs starting at different $r_{\text{min}}(b)$ and with different $\mu(b)$. The number of brackets is set to

$$s = \lfloor \log_\eta (1/r_{\text{min}}) \rfloor + 1, \tag{6}$$

which coincides with the number of fidelity steps that can be performed on a geometric scale on the interval $[r_{\text{min}}, 1]$. In bracket $b \in \{1, 2, \ldots, s\}$, a number of $\mu(b)$ samples are initially sampled and evaluated with initial fidelity $r = \eta^{s-b}$. $\mu(b)$ is chosen such that each bracket needs an approximately similar amount of budget: $\mu(b) = \lceil s \cdot \frac{\eta^{s-b}}{s-b+1} \rceil$.

5) Bayesian Optimization Hyperband (BOHB) [16]: Model-based methods outperform Hyperband when a relatively large amount of budget is available and many objective function evaluations can be performed. BOHB was created to overcome this drawback. This method iterates through successive halving brackets like Hyperband, but, instead of sampling new configurations randomly, it uses information from the archive to propose points that are likely to perform well. A total number of N_s configurations are proposed for evaluation; ρ are sampled at random, and the rest are chosen based on a surrogate model induced on the evaluated configurations in $\mathcal{A}$. The models used by BOHB are a pair of kernel density estimators of the top and bottom configurations in $\mathcal{A}$, similar to the process in [45]. To implement BOHB in Algorithm 2, one therefore needs to use an inducer $\mathcal{I}_{f_{\text{surr}}}$ that produces a function that calculates the ratio of kernel densities, an unusual kind of regression model.

B. Limitations and Further MF-HPO Algorithms

The following lists notable HPO algorithms not currently covered by the optimization space of Algorithm 1. They were excluded because they differ in too substantial ways from the other algorithms considered here.

1) FABOLAS [46]: Fabolas is a continuous multi-fidelity BO method, where the conditional validation error is modelled as a Gaussian process using a complex kernel-capturing covariance with the training set fraction $r \in (0, 1]$ to allow for adaptive evaluation at different resource levels.

2) Asynchronous successive halving (ASHA) [15] and asynchronous Hyperband: Hyperband, as well as SH, have the drawback that batch sizes decrease throughout the stages of an SH run, preventing efficient utilization of parallel resources. ASHA is an effective method to parallelize SH by an asynchronous parallelization scheme. A shared archive across a number of different workers is maintained. Instead of waiting until all n configurations of a batch have been evaluated for fidelity r, every free worker queries the shared archive $\mathcal{A}$ for "promotable" configurations (i.e., configurations that belong to the fraction of top η^{-1} configurations evaluated with the same fidelity). Asynchronous Hyperband works similarly.

3) Asynchronous BOHB (A-BOHB) [17]: A-BOHB, an asynchronous extension of BOHB where configurations are sampled from a joint Gaussian Process, explicitly capturing correlations across fidelities. In contrast to ASHA and asynchronous versions of BOHB in the original BOHB publication [16], A-BOHB does not perform synchronization after each stage but instead uses a stopping rule [47] to asynchronously determine whether a configuration should continue to run or be terminated.

V. EXPERIMENTAL ANALYSIS

Given the formalization of the framework in Section IV, our goal is to find the best representative (out of this class of algorithms) by solving the third-level optimization problem in Equation (5), and explain the role of specific algorithmic components in a benchmark-driven approach. We aim to answer the following research questions:

RQ1: How does the optimal configuration of our MF-HPO framework differ between problem scenarios, i.e., do different problem scenarios benefit from different HPO algorithms?

RQ2: How does our optimized MF-HPO algorithm compare to other established HPO implementations?

RQ3: Does the successive-halving fidelity schedule have an advantage over the simpler equal-batch-size schedule?

RQ4: What is the effect of using multi-fidelity methods in general?

RQ5a: Does changing SAMPLE configuration parameters throughout the optimization process offer an advantage?

TABLE II: Three benchmark collections of YAHPO Gym used in our benchmark.

			Hyperparameter Types					
Scenario	Target Metric	d	Cont.	Integer	Categ.	Hierarchical	# Instances	# Training Set
lcbench: HPO of a neural network	cross entropy loss	7	6	1	0	✗	35	8
rbv2_super: AutoML pipeline configuration	log loss	38	20	11	7	✓	89	30
nb301: Neural architecture search	validation accuracy	34	0	0	34	✓	1	—

RQ5b: Does (more complicated) surrogate-assisted sampling in SAMPLE provide an advantage over using simple random sampling with surrogate filtering?

RQ6: What effect do different surrogate models (or using no model at all) have on performance?

RQ7: Does the equal-batch-size schedule give an advantage over established methods when parallel resources are available?

We rely on benchmark scenarios of the YAHPO Gym benchmark suite [48], each of which provides a number of related *instances* of optimization problems. The benchmark scenarios we have chosen cover three important application areas of AutoML: Hyperparameter optimization of a neural network (*lcbench*), AutoML pipeline configuration (*rbv2_super*), and neural architecture search (*nb301*). These classes of problems do not only represent common and relevant tasks for researchers and practitioners in the field; as presented in Table II, they are also quite different with regards to: (1) the dimensionality of the search space, (2) hyperparameter types (categorical, integer, continuous), and (3) whether there are hierarchical dependencies between hyperparameters. More details on the characteristics of the problem classes are given in Appendix B. To avoid an optimistic bias in the analysis caused by over-adaption to the random peculiarities of the particular instances used during configuration, we are using meta-holdout splits on the level of HPO problem instances (see Appendix IV). This means that for analysing the performance of a configured candidate of Algorithm 2, we are evaluating this candidate by running it on instances that were not seen during configuration. Algorithm 2 is always run with a budget limit corresponding to $30 \cdot d$ full fidelity evaluations (where d is the dimension of the problem instance).

A. Algorithm Design via Configuration

First, we describe the experiments we conducted to configure Algorithm 2 via optimization.

We follow the PBO principle, and configure Algorithm 2 by optimizing separately for different HPO scenarios, namely for *lcbench* and *rbv2_super*, resulting in two optimized configurations $\gamma^{*lcbench}$ and γ^{*rbv2_super}, respectively. The *nb301* scenario is not used for configuration, but exclusively for subsequent analysis.

For the algorithm configuration of our framework (third level), the performance objective $\mathbb{E}_{\omega \sim \mathbb{P}_\Omega}\left[\zeta(A(\omega, \boldsymbol{\gamma}))\right]$ for a configuration $\boldsymbol{\gamma}$ in Equation (5) is estimated by running Algorithm 2 (i.e., second level optimization) configured by $\boldsymbol{\gamma}$ on a set of problem instances and taking the average of observed performances. For this, all problem instances included in the respective benchmark scenario that have not been held out for subsequent analysis are used. As configurator for our framework we use BO with the lower confidence bound acquisition function [49] with interleaved random configurations every three evaluations. Configuration is repeated three times for each scenario with different random seeds. To get the overall best configuration, the set of all evaluated configurations $\boldsymbol{\gamma}$ (i.e., the third level optimization archive) is combined into a single data set for each scenario. To estimate the actual best configuration, a common *identification criterion* [50] is used: a surrogate model is fitted on the combined datasets and the optimum among the in-sample predictions of this model is used ($\boldsymbol{\gamma}^{*lcbench}$ and $\boldsymbol{\gamma}^{*rbv2\_super}$, respectively). We also store the (surrogate-smoothed) optima of all three individual optimization runs and record the range of configuration parameter values to obtain an estimate of the uncertainty of the overall optimal configurations.

The search space used for the optimization of Algorithm 2 is shown in Appendix C, Table V. While the batch size μ is constant in the `equal` *batch_method*, it changes for every bracket when *batch_method* is `HB`. The batch sizes $\mu(2), \mu(3), \ldots$ are constructed from $\mu(1)$ dynamically as described in Section IV. The search space contains several surrogate learners: Random forests [51] (`RF`), K-nearest-neighbors with k set to 1 (`KNN1`), kernelized K-nearest-neighbors with "optimal" weighting [52] (`KKNN7`), and the ratio of density predictions of good and bad points, similar to tree parzen estimators [45] without a hierarchical structure as in BOHB [16] (`TPE`). For the pre-filtering sample distribution $\mathbb{P}_{\boldsymbol{\lambda}}(\mathcal{A})$, we evaluate both uniform sampling (`uniform`), and sampling from the estimated density of good points as done in BOHB [16] (`KDE`). *filter_mb* determines whether the surrogate model makes predictions assuming the highest fidelity value r observed (`TRUE`), as opposed to assuming the fidelity of the points being sampled; in the framework of the SAMPLE Algorithms 3 and 4 in Appendix A, this influences the behavior of $\mathcal{I}_{f_{\text{surr}}}$. Note that the maximum number of fidelity steps per batch s is not part of the search space and instead inferred automatically from η_{fid} and the lower bound for r that is given as part of the optimization problem instance. As in Hyperband, it is set to the largest number of stages that is possible given η_{fid} and the lower bound on r according to Equation (6).

B. Algorithm Analysis

Our goal in this work is not only to determine configurations of Algorithm 2 that perform well on the respective benchmarking scenarios, but also to determine what effect individual components have on performance. However, performing a complete sensitivity analysis would be prohibitively computationally expensive, as it would require evaluation of the objective (i.e., running Algorithm 2) in an experimental design of different configurations. Instead, we evaluate the performance of the candidate configurations found in Section V-A and alternative configurations – which are chosen in a way to allow for answering our research questions – on the benchmark test instances which were held out during configuration. A simple method to answer many of these questions is to take the optimized configuration of Algorithm 2 and swap components of it for simpler components (or removing them completely), thereby performing a one-factor-at-a-time analysis or an ablation study. However, the optimal values of some components may interact strongly with other components. We therefore auto-configure the framework several times under certain constraints dictated by our particular research question at hand. For example, to investigate the effect of varying n_{trn} and N_{s} over t, we run the optimization of Algorithm 2 with the constraint $n(0)$ to be equal to $n(1)$ and compare the resulting configuration to the overall optimum $\boldsymbol{\gamma}$. Table III lists the different values of $\boldsymbol{\gamma}$ we generate under different constraints. For each value of $\boldsymbol{\gamma}$, we run the respectively configured HPO algorithm on both the *lcbench* and the *rbv2_super* scenario, and (unless stated otherwise) once each for *batch_method* set to `equal` and `HB`. We refer to an optimized configuration that was obtained on the *lcbench* scenario with *batch_method* set to `equal` as $\boldsymbol{\gamma}^{*lcbench}$[`equal`], and to the overall optimum (i.e., the better of $\boldsymbol{\gamma}^{*lcbench}$[`equal`] and $\boldsymbol{\gamma}^{*lcbench}$[`HB`]) as $\boldsymbol{\gamma}^{*lcbench}$; similar for *rbv2_super*.

Every evaluation of a framework configuration, i.e., a complete HPO run on a problem instance, is repeated 30 times (with different random seeds) to allow for statistical analysis.

The analysis of our research questions is based on the following tables and visualizations. Table VI in Appendix D shows the configuration parameters that were selected for each benchmark scenario with various search space restrictions. We perform all optimization runs constrained to the fidelity scheduling `equal` and `HB`, respectively, and denote the resulting optimal configurations γ^*[`equal`] and γ^*[`HB`]. Figure 2 shows the configuration values of the top 80 evaluated points according to their surrogate-predicted performance. The ranges covered by the bee swarms are again an indicator of approximate ranges of configuration values that can be expected to work well. Figure 5 shows the final performance at $30 \cdot d$ full-budget evaluations for all optimization runs that were performed. The standard error shown is the estimated standard deviation of the mean of benchmark-instance-wise performance, representing uncertainty about the "true" performance mean if an infinite number of benchmark instances of the given class of problems were available.

We now describe in more detail how we operationalize each of the research question RQ1-RQ7 and report results.

RQ1: How does the optimal configuration differ between problem scenarios, i.e., do different problem scenarios benefit from different HPO algorithms?

Setup: We investigate the difference in the values that $\boldsymbol{\gamma}^{*lcbench}$ and $\boldsymbol{\gamma}^{*rbv2\_super}$ take, and put this difference in perspective by comparing it to the uncertainty of these values. To evaluate how well $\boldsymbol{\gamma}^{*lcbench}$ and $\boldsymbol{\gamma}^{*rbv2\_super}$ generalize to other problem scenarios, we evaluate them on the respective instances of scenarios that they were not configured on.

Results: As can be seen in Table VI and in Figure 2, many of the selected components of the $\boldsymbol{\gamma}^*$ are relatively close to each other across the two scenarios on which they were optimized, relative to their uncertainty ranges. $\mathcal{I}_{f_{\text{surr}}}$ is chosen as `KNN1` on *rbv2_super*, but can also use `KKNN7` on *lcbench*, which in fact seems to be slightly preferred. This is interesting as KNN-based models are rarely considered in surrogate-based HPO; the typically preferred random forest model was not selected. $\mathbb{P}_{\boldsymbol{\lambda}}(\mathcal{A})$ takes any of the two values for *rbv2_super*, but is chosen to be `KDE` in *lcbench*. Finally, $\rho(0)$ is close to 1 in the beginning on *rbv2_super*, and closer to 0 (although still greater than $\rho(1)$) for *lcbench*.

The degree to which the differences in $\boldsymbol{\gamma}^*$ influence the outcome can be observed in Figure 5. The optimized results generalize well to test instances from the same scenario as they were configured on. Figure 3 shows the optimization progress (on unseen test instances) of configurations if configured on the same scenario vs. configurations that were configured on a different scenario. We see, for example, a clear advantage of the configurations that we obtained by optimizing directly on *lcbench* when we evaluate them on their respective held out test instances. We suspect that this difference in performance is mainly due to the different choices of surrogate model classes $\mathcal{I}_{f_{\text{surr}}}$ as well as the random interleave fraction ρ (cf. Figure 2), and that specific settings for these two algorithmic components are needed for *lcbench* to reach optimal performance.

This is not the case for the *rbv2_super* scenario, where none of the different algorithms seem to clearly exploit the problem structure of *rbv2_super* better than others.

RQ2: How does the optimized algorithm compare to other established HPO implementations?

Setup: We evaluate several well-known HPO algorithms in their default configuration on the same benchmark instances: for BOHB [16], we use the implementation found in `HpBandSter`[2] (version 0.7.4); for HB [14], we use `mlr3hyperband`[3] (version 0.1.2); for SMAC [5], we use the SMACv3 package[4] (version 1.0.1). We also construct a traditional Gaussian process-based BO (GPBO) [4] with mlrMBO[5] (version 1.1.5). As GPBO works best with numerical search spaces, we only evaluate it on *lcbench*. Note that GPBO, SMAC, and RS are not multi-fidelity algorithms and therefore always evaluate points with maximum fidelity 1.

Results: The performance curves for the mean normalized regret are shown in Figure 3, and the final performance values at $30 \cdot d$ full fidelity evaluations are shown in Figure 5. A critical difference plot and test can be seen in Figure 4b. The behavior of RS, HB, BOHB, and SMAC is not surprising; initially, RS and SMAC perform the same, as SMAC evaluates an initial random design. After this, the performance of SMAC improves quickly. HB and BOHB initially both perform better than RS or SMAC because of their multi-fidelity evaluations, but there is little difference between them. After a while, BOHB starts to outperform HB because of its surrogate-based sampling, which aligns with the observations in [16]. Therefore, BOHB performs well for most budgets, often being the best optimizer for a budget of one as well as for 100 full fidelity evaluations. Given its multi-fidelity characteristics, HB is a good choice for low budgets, while SMAC is well suited for larger optimization budgets. Our framework is very competitive on both *lcbench* and *rbv2_super*, but is outperformed by SMAC on *nb301*. We assume that this is because Algorithm 2 was not explicitly optimized for the *nb301* scenario.

Although our framework was only optimized for performance at $30 \cdot d$ evaluations, it is also competitive with BOHB after fewer evaluations, as seen in Figure 4b.

RQ3: Does the successive-halving fidelity schedule have an advantage over the (simpler) equal-batch-size schedule?

Setup: It is likely that the type of fidelity scheduling used interacts with other configuration parameters. Therefore, we investigate

[2]https://github.com/automl/HpBandSter
[3]https://cran.r-project.org/package=mlr3hyperband
[4]https://github.com/automl/SMAC3
[5]https://cran.r-project.org/package=mlrMBO

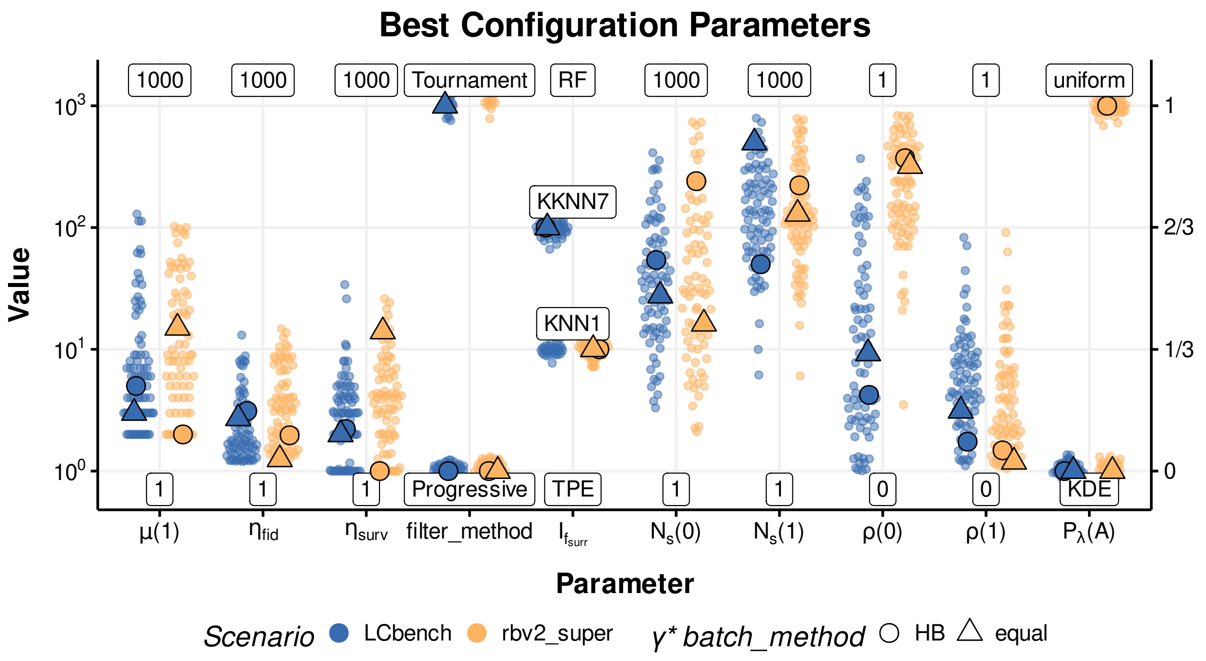


Fig. 2: Beeswarm plot of the best configurations according to the surrogate model over the meta-optimization archive of γ^*. Shown are the top 80 configuration points (according to the surrogate-model-predicted performance) that were evaluated during optimization. Levels of discrete parameters are shown. Most numeric parameters are on a log-scale (left axis), except for $\rho(0)$, $\rho(1)$, which are on a linear scale (right axis). Instead of showing both $N_s^0(t)$ and $N_s^1(t)$, their geometric mean $N_s(t)$ is shown. The highlighted large points are γ^*[HB] and γ^*[EQUAL], which were found on both benchmark scenarios.

the difference of resulting optimal configurations γ^*[equal] and γ^*[HB].

Results: In both scenarios, the batch method HB is ultimately selected for the optimum γ^*, although Figures 4a and 4b show that the difference to batch size equal is not statistically significant at $\alpha = 1\%$. We observe that the equal fidelity scheduling mode has several advantages: it is much simpler than HB as it does not need to keep track of SH brackets, and does not need to adapt $\mu(b)$ to make the expended budget at each bracket approximately equal. As another benefit, it allows for easy parallel scheduling of evaluations (see also Figure 1). This is because it always schedules the same number of function evaluations at a time, which can therefore be run synchronously.

RQ4: What is the effect of using multi-fidelity methods in general?

Setup: We evaluate the performance of a modified γ^* where the number of fidelity stages s is set to to 1, thus ensuring that configurations are only evaluated with maximum fidelity 1.[6]

Results: Our results show the superiority of MF-HPO methods compared to HPO methods that do not make use of lower-fidelity approximations. Figure 4a suggests that multi-fidelity methods are significantly better than their non-multi-fidelity counterparts if optimization is stopped at an intermediate overall budget corresponding to 100 full fidelity evaluations. To be more precise, we see that BOHB as well as both optimized variants $\boldsymbol{\gamma}^*$[equal] and $\boldsymbol{\gamma}^*$[HB] (optimized for the respective scenario, respectively) significantly outperform SMAC under this strict budget constraint. In line with [14], HB significantly outperforms RS for this budget. On the other hand, Figure 4b provides evidence that multi-fidelity methods can achieve performance on the same level as state-of-the-art methods that do not make use of low fidelity approximations (e.g., SMAC) for larger budgets. We conclude that a properly designed multi-fidelity mechanism provides substantial improvements of anytime performance without affecting performance for larger budgets negatively. In our opinion, the gain in anytime performance justifies the additional algorithmic complexity that is introduced by multi-fidelity methods.

RQ5a and RQ5b: Does changing SAMPLE configuration parameters throughout the optimization process offer an advantage? Does (more complicated) surrogate-assisted sampling in SAMPLE provide an advantage over using simple random sampling with surrogate filtering?

Setup: To investigate RQ5a (i.e., the effect of the dependence of ρ, n_{trn} and the N_s configuration parameters on t), we performed an optimization where this t-dependence was removed. As these parameters are interpolated between the values at $t = 0$ and $t = 1$, this corresponds to restricting the search space to where these values are equal, as shown for γ_2 in Table III. In addition to this, we ran another optimization where we further restricted N_s^0 and N_s^1 to be equal, n_{trn} to be 1, and only the tournament *filter_method* be used for RQ5b. The performance of the resulting configurations gives an indication of the performance that is lost for the gain in simplicity.

Results: The observations made for γ_2 (forbidding change over

[6]Because s is not part of the search space Γ and is instead given by Equation 6, this is achieved by setting η_{fid} to ∞.

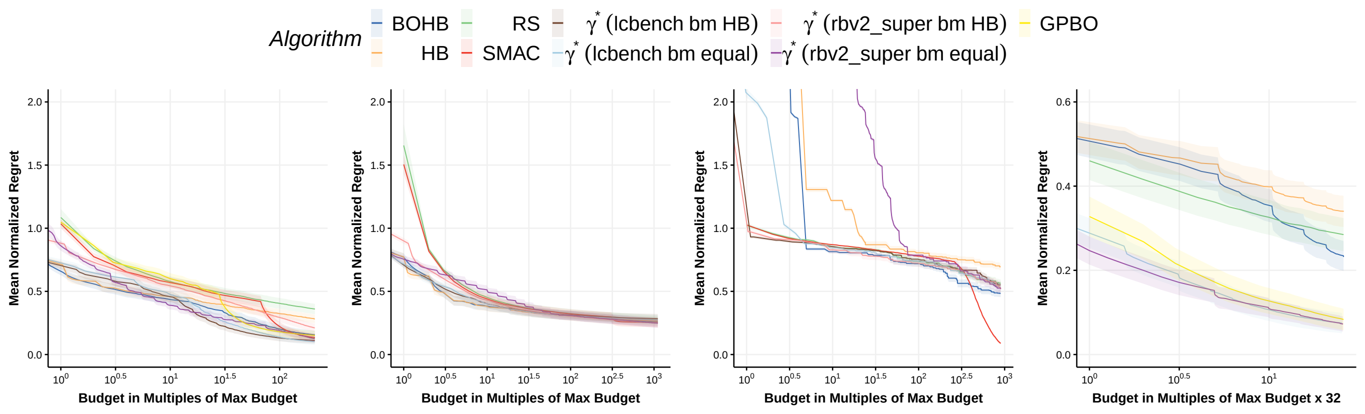
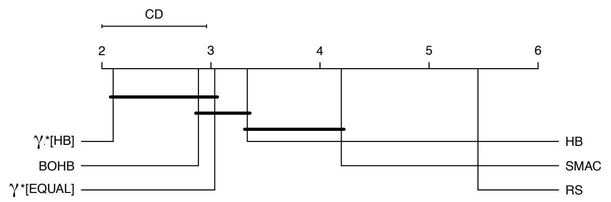
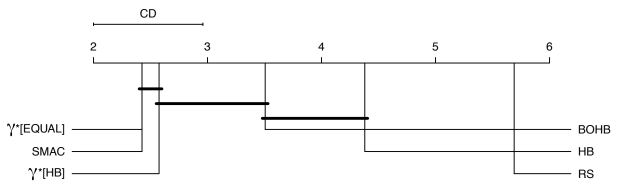


Fig. 3: Optimization progress (mean normalized regret) of serial evaluation on each benchmark scenario as well as 32x parallel evaluation on *lcbench*. Different configurations of Algorithm 2 are executed on benchmark functions that have not been used for the meta-optimization itself, and the progress of these algorithm runs is shown. "γ^*(lcbench bm equal)" is the configuration obtained from optimizing on *lcbench* with *batch_method* `equal`, other labels are contructed similarly. Shown is the mean over 30 evaluations, averaged over all available test benchmark instances for each of the three scenarios. The uncertainty bands show the standard error over the test instances. Note the log-scale on the x-axis. Regret is calculated as the difference between the best evaluation performance so far and the overall best value found on each benchmark instance over all experiments; normalized such that 1 corresponds to the median of the performance of all randomly sampled full fidelity evaluations. We plot performance values observed by the HPO algorithm which depend on evaluation fidelity. This is the reason for the initially "slow" convergence of algorithms that make their first full-fidelity evaluation late. Note that μ of γ^*[`equal`] was set to 32 for the parallel evaluations, and HB and BOHB were only naïvely parallelized to simulate a synchronous "single optimizer, multiple workers" environment. See Figure 6 in Appendix E for a larger version.



(a) Intermediate optimization budget of 100 full evaluations



(b) Full evaluation budget (final performance)

Fig. 4: Critical difference plot [53] comparing the performance of different algorithms across all instances and scenarios. For each of the three scenarios, the mean performance (across replications) for each of the six algorithms is computed ($\boldsymbol{\gamma}^*$[`HB`] is equal to $\boldsymbol{\gamma}^{*lcbench}$[`HB`] for instances of the lcbench scenario, and to $\boldsymbol{\gamma}^{*rbv}$[`HB`] for the *rbv2_super* scenario; same for $\boldsymbol{\gamma}^*$[`EQUAL`]). The critical difference test is based on the ranks of the algorithms computed per scenario and instance. Lower ranks are better. Horizontal bold bars indicate that there is no significant difference between algorithms ($\alpha = 1\%$). GPBO, which was not evaluated on all scenarios, is not included.

time) and γ_3 (forbidding change over time and within each batch) are slightly contradictory. In particular, the *nb301* performance of $\gamma_2^{lcbench}$[`HB`] is a visible outlier with regards to optimization performance. There is no obvious explanation from inspecting the configuration parameters of $\gamma_2^{lcbench}$[`HB`], but it is possible that it is an accidental "good fit" of configuration parameters to the specific landscape of *nb301*.

On *lcbench* and *rbv2_super*, the impact of restricting the search space is smaller and within the uncertainty of the performance of a single configuration. However, we note that both changing configuration parameters over time and within each batch sample introduces significant complexity to the algorithm; thus we prefer the restricted optimization results over γ^*.

RQ6: What effect do different surrogate models (or using no model at all) have on performance?

Setup: We evaluate the overall result $\boldsymbol{\gamma}^*$[`equal`] with $\mathcal{I}_{f_{\text{sur}}}$ set to each of the inducers in the original search space (see Table V). Furthermore, $\boldsymbol{\gamma}^*$[`equal`] is evaluated with ρ set to 1 (i.e., all points are sampled randomly from a distribution that may be non-uniform), and finally, with $\rho = 1$ and $\mathbb{P}_{\boldsymbol{\lambda}}(\mathcal{A}) =$ `uniform` (i.e., all points are sampled completely uniformly at random).

Results: Surprisingly, the simple k-nearest-neighbors algorithm seems to be chosen consistently by the algorithm configuration for both *lcbench* and *rbv2_super* (see Figure 2), either with a value of $k = 1$ or $k = 7$. This result is in line with what we already speculated for RQ1. Our ablation experiments suggest that the performance of the optimizer is on average best when using this surrogate learner, even though the differences do not seem to be significant. `KNN1` is therefore a reasonable, and simpler, alternative to more complex surrogate learners like the `TPE`-based method proposed for the original BOHB algorithm.

RQ7: Does the equal-batch-size schedule give an advantage over established methods when parallel resources are available?

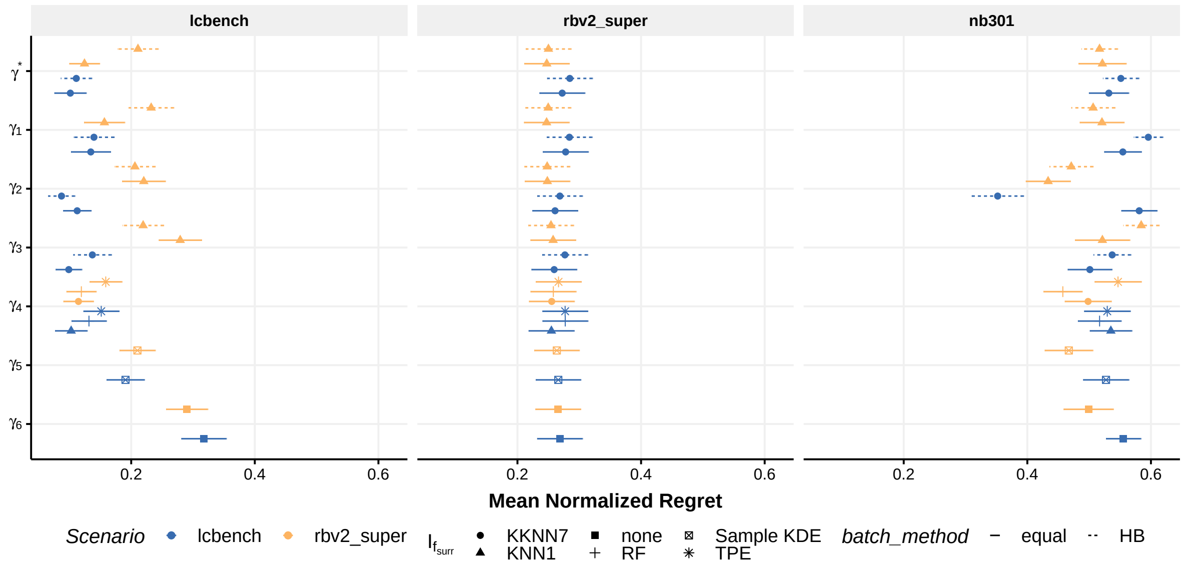


Fig. 5: Mean normalized regret of final performance on "test" benchmark instances for the configuration, shown in Table III. Shown is the mean over 30 evaluations, averaged over all available test benchmark instances for each of the three scenarios. The uncertainty bands show standard error over instance means. Regret is calculated as the difference between the best evaluation performance so far and the overall best value on each benchmark instance over all experiments; normalized such that 1 corresponds to the median of the performance of all randomly sampled full fidelity evaluations.

Setup: Optimization of ML methods that are expensive to evaluate is often done in parallel; we evaluate the performance of our method and other methods in a (simulated) parallel setting. We evaluate γ^*[equal] with μ set to 32 and with an optimization budget of $30 \cdot 4 \cdot d$, where d is the dimensionality of the optimization problem. We compare it to GPBO with qLCB [10] for 32 parallel evaluations, and simulate parallel execution of RS by running $30 \cdot 4 \cdot d$ random evaluations. Both BOHB and SMAC offer parallelized versions, but the YAHPO Gym benchmark package does not yet provide support for asynchronous parallel evaluations [48]. However, since HB and BOHB propose evaluations in batches, we compared HB and BOHB by accounting for submitted batches in increments of 32, essentially simulating a single HB/BOHB optimizer sending evaluations to 32 parallel workers and waiting for their completion synchronously.

Results: Figure 3 shows that our algorithm is competitive with GPBO – a state-of-the-art synchronously parallel optimization algorithm – when evaluated with 32 parallel resources. This result also shows the main advantage that the `equal` fidelity schedule has over scheduling like HB, as synchronously parallelizing HB or BOHB puts them at a great disadvantage over even RS. For HB and BOHB, it is necessary to use asynchronously parallelized methods [15], [17] or use an archive shared between multiple workers [16] to obtain competitive results. However, synchronous objective evaluations are much easier to implement in many environments than asynchronous communication between workers, making the advantage of the simplicity of the `equal` schedule even more pronounced.

C. Reproducibility and Open Science

The implementation of the framework in Algorithm 2 and reproducible scripts for the algorithm configuration and analysis are available in public repositories.[7] All data that were generated by our analyses are available as well.

VI. Conclusion

We presented a principled approach and framework to benchmark-driven algorithm design and applied it to generic multi-fidelity HPO. We formalized the search space of multi-fidelity hyperparameter optimizers and created a rich and configurable optimization framework. Given the search space, we used BO for meta-optimization of our framework on two different problem scenarios within the field of AutoML, and evaluated the result on held out test problems and an entirely held out test scenario. We evaluated the configured optimizers and compared to BOHB, HB, SMAC, and a simple RS as reference. We performed an extensive analysis of the effect of different algorithmic components on performance, while also considering the additional algorithmic complexity they introduce. Our configured framework showed equal and in some cases superior performance to widely-used HPO algorithms.

The additional algorithmic complexity introduced by multi-fidelity evaluations provides substantial benefits. However, based on our experiments, we argue that design choices made by established multi-fidelity optimizers like BOHB can be replaced by simpler choices: For example, the (more complex) SH schedule is not significantly better than a schedule using equal batch sizes, which allows for more efficient parallelization.

KDE-based sampling of points to propose, whether filtered by a surrogate model or not, was consistently chosen by our framework. This detail, which is not usually presented as the main feature of

[7]https://github.com/mlr-org/smashy, https://github.com/compstat-lmu/paper_2021_benchmarking_special_issue

TABLE III: Summary of Experiment. Shown are the various optimizer configurations γ that were obtained from optimizations with different constraints. "Name": The name by which we refer to the configuration in the text. "RQ": The research question that mainly relates to the configuration. "Optimize": Whether the given configuration was obtained by conducting a (possibly constrained) optimization (✓), or by substituting values into the global optimum γ^*.

Name	RQ	Optimize	Design Modification
γ^*	1, 2, 3	✓	none (global optimization)
γ_1	4	✗	$\eta_{\text{fid}} \to \infty$
γ_2	5a	✓	$n_{\text{trn}}(0) = n_{\text{trn}}(1),\ N_{\text{s}}^0(0) = N_{\text{s}}^0(1),\ N_{\text{s}}^1(0) = N_{\text{s}}^1(1),\ \rho(0) = \rho(1)$
γ_3	5b	✓	*filter_method*$\to$ `tournament`, $n_{\text{trn}} \to 1,\ N_{\text{s}}^0(0) = N_{\text{s}}^0(1) = N_{\text{s}}^1(0) = N_{\text{s}}^1(1),\ \rho(0) = \rho(1)$
γ_4	6	✗	*batch_method*$\to$ `equal`, $\mathcal{I}_{f_{\text{sur}}} \to *$
γ_5	6	✗	*batch_method*$\to$ `equal`, $\rho \to 0$
γ_6	6	✗	*batch_method*$\to$ `equal`, $\rho \to 0,\ \mathbb{P}_{\boldsymbol{\lambda}}(\mathcal{A}) \to$ `uniform`
γ_7	7	✗	*batch_method*$\to$ `equal`, $\mu \to 32$, quadruple budget

BOHB, seems to have an unexpectedly large impact. On the other hand, our optimization results suggest that a surprisingly simple surrogate learner (`knn`, $k = 1$) can perform even better.

Some components of our search space with large algorithmic complexity have not shown much benefit. Optimization on *rbv2_super* did choose time-varying random interleaving, and overall, more aggressive filtering late during an optimization run ($N_{\text{s}}(1) > N_{\text{s}}(0)$) was slightly favored, but the results did not consistently outperform a configuration obtained from a restricted optimization that excluded time-varying configuration parameters.

Our analysis of the set of best observed performances during optimization indicates that there is a large agreement between benchmark scenarios about what the optimal γ^* configuration should be, with parameters that control (model-based) sampling and the surrogate model being the notable exception. This suggests that there may be a set of configuration parameters that are either generally good for many ML problems, or have little impact on performance and can therefore be set to the simplest value. However, some configuration parameters should be adapted to the properties of the particular given optimization problem. The meta-optimization framework presented in this work can be used in future work to investigate the relationship between features of optimization problems and related optimal configurations.

Other fruitful directions for future work include the more in-depth evaluation of asynchronous evaluations; asynchronous methods are important nowadays where parallel resources are plentiful, but current widely-used surrogate-based benchmarks do not allow for easy asynchronous evaluations. Suggested methods – such as waiting with a sleep-timer for an appropriate amount [16] – are impractical for meta-optimization.

Acknowledgments

The authors of this work take full responsibilities for its content. This work was supported by the German Federal Ministry of Education and Research (BMBF) under Grant No. 01IS18036A. Lennart Schneider is supported by the Bavarian Ministry of Economic Affairs, Regional Development and Energy through the Center for Analytics – Data – Applications (ADACenter) within the framework of BAYERN DIGITAL II (20-3410-2-9-8). Lars Kotthoff is supported by NSF grant 1813537.

References

[1] B. Bischl, M. Binder, M. Lang, T. Pielok, J. Richter, S. Coors, J. Thomas, T. Ullmann, M. Becker, A. Boulesteix, D. Deng, and M. Lindauer, "Hyperparameter optimization: Foundations, algorithms, best practices and open challenges," *CoRR*, vol. abs/2107.05847, 2021.

[2] L. Kotthoff, C. Thornton, H. H. Hoos, F. Hutter, and K. Leyton-Brown, "Auto-weka: Automatic model selection and hyperparameter optimization in weka," in *Automated Machine Learning: Methods, Systems, Challenges*, F. Hutter, L. Kotthoff, and J. Vanschoren, Eds. Cham: Springer International Publishing, 2019, pp. 81–95.

[3] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997.

[4] D. R. Jones, M. Schonlau, and W. J. Welch, "Efficient global optimization of expensive black-box functions," *J. Glob. Optim.*, vol. 13, no. 4, pp. 455–492, 1998.

[5] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *Learning and Intelligent Optimization*, C. A. C. Coello, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 507–523.

[6] K. Swersky, J. Snoek, and R. P. Adams, "Freeze-thaw bayesian optimization," *CoRR*, vol. abs/1406.3896, 2014.

[7] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, pp. 281–305, 2012.

[8] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'12. Red Hook, NY, USA: Curran Associates Inc., 2012, p. 2951–2959.

[9] R. Turner, D. Eriksson, M. McCourt, J. Kiili, E. Laaksonen, Z. Xu, and I. Guyon, "Bayesian optimization is superior to random search for machine learning hyperparameter tuning: Analysis of the black-box optimization challenge 2020," in *NeurIPS 2020 Competition and Demonstration Track, 6-12 December 2020, Virtual Event / Vancouver, BC, Canada*, ser. Proceedings of Machine Learning Research, H. J. Escalante and K. Hofmann, Eds., vol. 133. PMLR, 2020, pp. 3–26.

[10] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Parallel algorithm configuration," in *Learning and Intelligent Optimization*, ser. Lecture Notes in Computer Science, Y. Hamadi and M. Schoenauer, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, no. 7219, pp. 55–70.

[11] B. Bischl, S. Wessing, N. Bauer, K. Friedrichs, and C. Weihs, "MOI-MBO: multiobjective infill for parallel model-based optimization," in *Learning and Intelligent Optimization - 8th International Conference, Lion 8, Gainesville, FL, USA, February 16-21, 2014. Revised Selected Papers*, ser. Lecture Notes in Computer Science, P. M. Pardalos, M. G. C. Resende, C. Vogiatzis, and J. L. Walteros, Eds., vol. 8426. Springer, 2014, pp. 173–186.

[12] J. González, Z. Dai, P. Hennig, and N. D. Lawrence, "Batch bayesian optimization via local penalization," in *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, AISTATS 2016, Cadiz, Spain, May 9-11, 2016*, ser. JMLR Workshop and Conference Proceedings, A. Gretton and C. C. Robert, Eds., vol. 51. JMLR.org, 2016, pp. 648–657.

[13] C. Chevalier and D. Ginsbourger, "Fast computation of the multi-points expected improvement with applications in batch selection," in *Learning and Intelligent Optimization*. Springer Berlin Heidelberg, 2013, pp. 59–69.

[14] L. Li, K. G. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization," *J. Mach. Learn. Res.*, vol. 18, pp. 185:1–185:52, 2017.

[15] L. Li, K. G. Jamieson, A. Rostamizadeh, E. Gonina, J. Ben-tzur, M. Hardt, B. Recht, and A. Talwalkar, "A system for massively parallel hyperparameter tuning," in *Proceedings of Machine Learning and Systems 2020, MLSys 2020, Austin, TX, USA, March 2-4, 2020*, I. S. Dhillon, D. S. Papailiopoulos, and V. Sze, Eds. mlsys.org, 2020.

[16] S. Falkner, A. Klein, and F. Hutter, "BOHB: robust and efficient hyperparameter optimization at scale," in *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, ser. Proceedings of Machine Learning Research, J. G. Dy and A. Krause, Eds., vol. 80. PMLR, 2018, pp. 1436–1445.

[17] L. C. Tiao, A. Klein, C. Archambeau, and M. W. Seeger, "Model-based asynchronous hyperparameter optimization," *CoRR*, vol. abs/2003.10865, 2020.

[18] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J. Crespo, and D. Dennison, "Hidden technical debt in machine learning systems," in *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds., 2015, pp. 2503–2511.

[19] K. G. Jamieson and A. Talwalkar, "Non-stochastic best arm identification and hyperparameter optimization," in *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, AISTATS 2016, Cadiz, Spain, May 9-11, 2016*, ser. JMLR Workshop and Conference Proceedings, A. Gretton and C. C. Robert, Eds., vol. 51. JMLR.org, 2016, pp. 240–248.

[20] H. H. Hoos, "Programming by Optimization," *Communications of the Association for Computing Machinery (CACM)*, vol. 55, no. 2, pp. 70–80, Feb. 2012.

[21] S. Minton, "Automatically Configuring Constraint Satisfaction Programs: A Case Study," *Constraints*, vol. 1, pp. 7–43, 1996.

[22] S. J. Westfold and D. R. Smith, "Synthesis of Efficient Constraint Satisfaction Programs," *Knowl. Eng. Rev.*, vol. 16, no. 1, pp. 69–84, 2001.

[23] D. Balasubramaniam, L. de Silva, C. A. Jefferson, L. Kotthoff, I. Miguel, and P. Nightingale, "Dominion: An Architecture-driven Approach to Generating Efficient Constraint Solvers," in *9th Working IEEE/IFIP Conference on Software Architecture*, Jun. 2011, pp. 228–231.

[24] A. R. KhudaBukhsh, L. Xu, H. H. Hoos, and K. Leyton-Brown, "SATenstein: automatically building local search SAT solvers from components," in *Proceedings of the 21st International Joint Conference on Artifical Intelligence*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2009, pp. 517–524.

[25] J.-N. Monette, Y. Deville, and P. van Hentenryck, "Aeon: Synthesizing Scheduling Algorithms from High-Level Models," in *Operations Research and Cyber-Infrastructure*, 2009, pp. 43–59.

[26] M. Birattari, Z. Yuan, P. Balaprakash, and T. Stützle, "F-race and iterated f-race: An overview," *Experimental methods for the analysis of optimization algorithms*, pp. 311–336, 2010.

[27] M. López-Ibáñez and T. Stützle, "The automatic design of multi-objective ant colony optimization algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 16, no. 6, pp. 861–875, 2012.

[28] M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, and T. Stützle, "The irace package: Iterated racing for automatic algorithm configuration," *Operations Research Perspectives*, vol. 3, pp. 43–58, 2016.

[29] L. C. T. Bezerra, M. López-Ibáñez, and T. Stützle, "Automatic Component-Wise Design of Multiobjective Evolutionary Algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 20, no. 3, pp. 403–417, 2016.

[30] N. Dang, L. P. Cáceres, P. D. Causmaecker, and T. Stützle, "Configuring irace using surrogate configuration benchmarks," in *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2017, Berlin, Germany, July 15-19, 2017*, P. A. N. Bosman, Ed. ACM, 2017, pp. 243–250.

[31] S. van Rijn, H. Wang, M. van Leeuwen, and T. Bäck, "Evolving the structure of Evolution Strategies," in *IEEE Symposium Series on Computational Intelligence (SSCI)*, 2016, pp. 1–8.

[32] G. Malkomes and R. Garnett, "Automating bayesian optimization with bayesian optimization," *Advances in Neural Information Processing Systems*, vol. 31, pp. 5984–5994, 2018.

[33] M. Lindauer, M. Feurer, K. Eggensperger, A. Biedenkapp, and F. Hutter, "Towards assessing the impact of Bayesian optimization's own hyperparameters," 2019.

[34] M. Lindauer, K. Eggensperger, M. Feurer, A. Biedenkapp, D. Deng, C. Benjamins, T. Ruhkopf, R. Sass, and F. Hutter, "Smac3: A versatile bayesian optimization package for hyperparameter optimization," 2021.

[35] A. Saltelli, "Sensitivity analysis for importance assessment," *Risk Analysis*, vol. 22, no. 3, pp. 579–590, 2002.

[36] W. Hoeffding, "A Class of Statistics with Asymptotically Normal Distribution," *The Annals of Mathematical Statistics*, vol. 19, no. 3, pp. 293 – 325, 1948.

[37] F. Hutter, H. Hoos, and K. Leyton-Brown, "An efficient approach for assessing hyperparameter importance," in *Proceedings of the 31st International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, E. P. Xing and T. Jebara, Eds., vol. 32, no. 1. Bejing, China: PMLR, 22–24 Jun 2014, pp. 754–762.

[38] C. Fawcett and H. H. Hoos, "Analysing differences between algorithm configurations through ablation," *J. Heuristics*, vol. 22, no. 4, pp. 431–458, 2016.

[39] S. Sheikholeslami, M. Meister, T. Wang, A. H. Payberah, V. Vlassov, and J. Dowling, "Autoablation: Automated parallel ablation studies for deep learning," in *EuroMLSys@EuroSys 2021, Proceedings of the 1st Workshop on Machine Learning and Systemsg Virtual Event, Edinburgh, Scotland, UK, 26 April, 2021*, E. Yoneki and P. Patras, Eds. ACM, 2021, pp. 55–61.

[40] M. López-Ibáñez and T. Stützle, "An experimental analysis of design choices of multi-objective ant colony optimization algorithms," *Swarm Intelligence*, vol. 6, pp. 207–232, 2012.

[41] J. de Nobel, D. Vermetten, H. Wang, C. Doerr, and T. Bäck, "Tuning as a Means of Assessing the Benefits of New Ideas in Interplay with Existing Algorithmic Modules," in *Genetic and Evolutionary Computation Conference Companion*. Association for Computing Machinery, 2021, pp. 1375–1384.

[42] A. Klein, L. C. Tiao, T. Lienart, C. Archambeau, and M. Seeger, "Model-based asynchronous hyperparameter and neural architecture search," *arXiv preprint arXiv:2003.10865*, 2020.

[43] M. Birattari, *Tuning Metaheuristics - A Machine Learning Perspective*, ser. Studies in Computational Intelligence. Springer, 2009, vol. 197.

[44] Z. Karnin, T. Koren, and O. Somekh, "Almost optimal exploration in multi-armed bandits," in *International Conference on Machine Learning*. PMLR, 2013, pp. 1238–1246.

[45] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," *Advances in neural information processing systems*, vol. 24, 2011.

[46] A. Klein, S. Falkner, S. Bartels, P. Hennig, and F. Hutter, "Fast bayesian optimization of machine learning hyperparameters on large datasets," in *Artificial Intelligence and Statistics*. PMLR, 2017, pp. 528–536.

[47] D. Golovin, B. Solnik, S. Moitra, G. Kochanski, J. Karro, and D. Sculley, "Google vizier: A service for black-box optimization," in *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, 2017, pp. 1487–1495.

[48] F. Pfisterer, L. Schneider, J. Moosbauer, M. Binder, and B. Bischl, "YAHPO Gym – Design Criteria and a new Multifidelity Benchmark for Hyperparameter Optimization," *arXiv:2109.03670 [cs, stat]*, 2021, arXiv: 2109.03670.

[49] D. R. Jones, "A taxonomy of global optimization methods based on response surfaces," *Journal of Global Optimization*, vol. 21, no. 4, pp. 345–383, 2001.

[50] H. Jalali, I. Van Nieuwenhuyse, and V. Picheny, "Comparison of kriging-based algorithms for simulation optimization with heterogeneous noise," *European Journal of Operational Research*, vol. 261, no. 1, pp. 279–301, 2017.

[51] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.

[52] R. J. Samworth, "Optimal weighted nearest neighbour classifiers," *The Annals of Statistics*, vol. 40, no. 5, Oct. 2012.

[53] J. Demsar, "Statistical comparisons of classifiers over multiple data sets," *J. Mach. Learn. Res.*, vol. 7, pp. 1–30, 2006.

[54] L. Zimmer, M. Lindauer, and F. Hutter, "Auto-pytorch tabular: Multi-fidelity metalearning for efficient and robust autodl," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 9, pp. 3079 – 3090, 2021.

[55] J. Siems, L. Zimmer, A. Zela, J. Lukasik, M. Keuper, and F. Hutter, "Nas-bench-301 and the case for surrogate benchmarks for neural architecture search," 2020.

[56] H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir *et al.*, "Wide & deep learning for recommender systems," in *Proceedings of the 1st workshop on deep learning for recommender systems*, 2016, pp. 7–10.

[57] J. Vanschoren, J. N. van Rijn, B. Bischl, and L. Torgo, "Openml: networked science in machine learning," *SIGKDD Explor.*, vol. 15, no. 2, pp. 49–60, 2013.

[58] M. Binder, F. Pfisterer, and B. Bischl, "Collecting empirical data about hyperparameters for data driven automl," in *Proceedings of the 7th ICML Workshop on Automated Machine Learning (AutoML 2020)*, 2020.

[59] Y. A. Malkov and D. A. Yashunin, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," *IEEE transactions on pattern analysis and machine intelligence*, vol. 42, no. 4, pp. 824–836, 2018.

[60] J. Friedman, T. Hastie, and R. Tibshirani, "Regularization paths for generalized linear models via coordinate descent," *Journal of Statistical Software*, vol. 33, no. 1, pp. 1–22, 2010.

[61] M. N. Wright and A. Ziegler, "ranger: A fast implementation of random forests for high dimensional data in C++ and R," *Journal of Statistical Software*, vol. 77, no. 1, pp. 1–17, 2017.

[62] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification And Regression Trees*. Routledge, Oct. 2017.

[63] B. E. Boser, I. Guyon, and V. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the Fifth Annual ACM Conference on Computational Learning Theory, COLT 1992, Pittsburgh, PA, USA, July 27-29, 1992*, D. Haussler, Ed. ACM, 1992, pp. 144–152.

[64] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. New York, NY, USA: ACM, 2016, pp. 785–794.

[65] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable Architecture Search," in *Proceedings of the International Conference on Learning Representations*, 2019.

[66] A. Krizhevsky, "Learning multiple layers of features from tiny images," University of Toronto, Tech. Rep., 2009.

APPENDIX A
SAMPLE ALGORITHMS

The pseudocode of both SAMPLE algorithms is presented here for clarity.

Algorithm Algorithm 3, SAMPLETOURNAMENT, diversifies the set of points proposed through an extension that draws different points $\{\boldsymbol{\lambda}_S\}$ in n *tournaments* at each invocation of SAMPLE. Each tournament yields the top n_{trn} points out of $n_{\text{trn}} \cdot N_{\text{s}}^{(i)}$ samples according to the surrogate model, where $i \in \{1, \ldots, n\}$. We parameterize the number of points sampled for each tournament using the configuration parameters N_{s}^{0} and N_{s}^{1}; the effective value of N_{s} for each point is interpolated geometrically[8]: $N_{\text{s}}^{(i)} = \lfloor (N_{\text{s}}^{0})^{(n-i)/(n-1)} \cdot (N_{\text{s}}^{1})^{(i-1)/(n-1)} \rceil$. The special case of $n_{\text{trn}} = 1$ corresponds to a basic SAMPLE subroutine where points $\boldsymbol{\lambda}_S^{(i)}$ are each independently filtered with different effective filter factors $N_{\text{s}}^{(i)}$.

Besides the sampling method described above, we propose an alternative method, Algorithm 4, which we name SAMPLEPROGRESSIVE: instead of sampling $N_{\text{s}}^{(i)}$ points independently for each configuration $\boldsymbol{\lambda}_S^{(i)}$ with $i \in \{1, \ldots, \mu\}$, we sample a single ordered pool $\mathcal{P}$ of $\mu \cdot \max(N_{\text{s}}^{0}, N_{\text{s}}^{1})$ random points once at the beginning of SAMPLE. Each $\boldsymbol{\lambda}_S^{(i)}$ is then selected as the point with the best surrogate-predicted performance from the first $\mu \cdot N_{\text{s}}^{(i)}$ points in $\mathcal{P}$ that was not already selected before.

Algorithm 3 SAMPLETOURNAMENT algorithm

Input: Archive $\mathcal{A}$, number of points to generate μ, current fidelity r

Configuration Parameters: Surrogate learner $\mathcal{I}_{f_{\text{sur}}}$, generating distribution $\mathbb{P}_{\boldsymbol{\lambda}}(\mathcal{A})$, random interleave fraction ρ, sample filtering rates $(N_{\text{s}}^{0}, N_{\text{s}}^{1})$, points to sample per tournament round n_{trn}.

State Variables: Batch of proposed configurations $C \leftarrow \emptyset$

```
1:  Use $\rho$ to decide how many points $n_{\text{random\_interleave}}$ to sample without filter
2:  $C \leftarrow$ Sample $n_{\text{random\_interleave}}$ points from $\mathbb{P}_{\boldsymbol{\lambda}}(\mathcal{A})$
3:  $n \leftarrow \lceil (\mu - n_{\text{random\_interleave}})/n_{\text{trn}} \rceil$    ▷ Numter of tournament rounds
4:  $f_{\text{sur}} \leftarrow \mathcal{I}_{f_{\text{sur}}}(\mathcal{A})$    ▷ Surrogate model
5:  for $i \leftarrow 1$ to $n$ do
6:      $n_{\text{sample}} \leftarrow \left\lfloor (N_{\text{s}}^{0})^{\frac{n-i}{n-1}} \cdot (N_{\text{s}}^{1})^{\frac{i-1}{n-1}} \right\rceil$
7:      $C_0 \leftarrow$ Sample $n_{\text{sample}}$ configurations from $\mathbb{P}_{\boldsymbol{\lambda}}(\mathcal{A})$
8:      Predict performances of points in $C_0$ using $f_{\text{sur}}$
9:      $C \leftarrow C \cup \text{SELECT\_TOP}\,(C_0, \min(n_{\text{trn}}, \mu - |C|))$
10: end for
11: return C
```

[8]Here, $\lfloor \cdot \rceil$ is the operation that rounds to the next integer.

Algorithm 4 SAMPLEPROGRESSIVE algorithm

Input: Surrogate learner $\mathcal{I}_{f_{\text{sur}}}$, Archive $\mathcal{A}$, number of points to generate μ, current fidelity r, random interleave fraction ρ, sample filtering rates $(N_{\text{s}}^{0}, N_{\text{s}}^{1})$, generating distribution $\mathbb{P}_{\boldsymbol{\lambda}}(\mathcal{A})$

State Variables: Batch of proposed configurations $C \leftarrow \emptyset$, (ordered) pool of sampled points to select from $\mathcal{P}$

```
1:  Use $\rho$ to decide how many points $n_{\text{random\_interleave}}$ to sample without filter
2:  $C \leftarrow$ Sample $n_{\text{random\_interleave}}$ configurations from $\mathbb{P}_{\boldsymbol{\lambda}}(\mathcal{A})$
3:  $\mu \leftarrow \mu - n_{\text{random\_interleave}}$
4:  $n_{\text{pool}} \leftarrow \mu \cdot \max(N_{\text{s}}^{0}, N_{\text{s}}^{1})$
5:  $\mathcal{P} \leftarrow$ Sample $n_{\text{pool}}$ configurations from $\mathbb{P}_{\boldsymbol{\lambda}}(\mathcal{A})$
6:  $f_{\text{sur}} \leftarrow \mathcal{I}_{f_{\text{sur}}}(\mathcal{A})$    ▷ Surrogate model
7:  Predict performances of points in $\mathcal{P}$ using $f_{\text{sur}}$
8:  for $i \leftarrow 1$ to $\mu$ do
9:      $n_{\text{options}} \leftarrow \left\lfloor (N_{\text{s}}^{0})^{\frac{\mu-i}{\mu-1}} \cdot (N_{\text{s}}^{1})^{\frac{\mu-1}{\mu-1}} \right\rceil$
10:     $\mathcal{P}_{\text{options}} \leftarrow$ first $n_{\text{options}}$ elements of $\mathcal{P}$
11:     $S \leftarrow \text{SELECT\_TOP}\,(\mathcal{P}_{\text{options}}, 1)$
12:     $C \leftarrow C \cup S$
13:     $\mathcal{P} \leftarrow \mathcal{P} - S$
14: end for
15: return C
```

APPENDIX B
BENCHMARK COLLECTIONS

While the underlying data for lcbench and nb301 have been previously used in publications ([54], [55]), rbv2_super is a novel task that has not been investigated previously in literature.

Benchmarks in the YAHPO Gym are implemented as surrogate model benchmarks, where a Wide & Deep [56] neural network was fitted to a set of pre-evaluated performance values of hyperparameter configurations.

HPO on a neural network (*lcbench* [54]): The first set of problems covers HPO on a relatively small and numeric search space. The neural network (more precisely, a funnel-shaped multilayer perceptron) that is tuned has a total of 7 numerical hyperparameters. The fidelity of an evaluation can be controlled by setting the number of epochs over which the neural network is trained. The instances belonging to this scenario represent HPO performed on 35 different classification tasks taken from OpenML [57]. As a target metric, we choose the cross entropy loss on the validation set.

AutoML pipeline configuration (*rbv2_super* [58]): Second, we investigate the problem of configuring an AutoML pipeline. Here, a learning algorithm must be selected first from the following candidates: approximate k nearest neighbors [59], elastic net linear models [60], random forests [61], decision trees [62], support vector machines [63], and gradient boosting [64]. The hyperparameters of each learner are chosen conditioned on this learner being active, i.e., there are hierarchical hyperparameter dependencies. The fidelity of a single evaluation can be controlled by choosing the size of the training data set that is used to train the respective learner. The automated optimization of the pipeline is performed for 89 different classification tasks [58], again taken from OpenML. As a target metric, we opt for the log loss.

Neural architecture search (*nb301* [55]): As third problem scenario, we consider neural architecture search. The search space of architectures is given by the darts search space [65], and architectures were trained and evaluated on CIFAR-10 [66]. A convolutional neural network is constructed by stacking so-called normal and reduction cells that each can be represented as a directed acyclic graph consisting of an ordered sequence of vertices (nodes) resembling feature maps, with each directed edge associated with an operation that transforms the input node. A tabular representation can be derived using 34 categorical parameters with 24 dependencies. Each architecture can be trained for 1 to 98 epochs, allowing again for lower fidelity evaluations. The target metric is validation accuracy.

Appendix C
Meta-Optimization Search Space

The full optimization space used for optimization of γ^* is presented here in Table V. Other γ results have the restrictions applied to them, as shown in Table III in Section V.

TABLE IV: Instances within the benchmarking scenarios *lcbench*, *rbv2_super*, and *nb301* within the YAHPO Gym test suite that have been used for the experimental analysis (Section V). We show the instances that have been used for optimization only (Section V-A), and the instances that have been held out from optimization and exclusively used for analysis (Section V-B).

Scenario	Instances used for configuration	Instances held out for analysis
lcbench	3945, 7593, 126026, 167201, 168329, 168868, 168908, 189354	34539 126025, 126029, 146212, 167083, 167104, 167149, 167152, 167161, 167168, 167181, 167184, 167185, 167190, 167200, 168330, 168331, 168335, 168910, 189862, 189865, 189866, 189873, 189905, 189906, 189908, 189909
rbv2_super	1050, 1053, 1056, 1068, 12, 1461, 1464, 1489, 1510, 1515, 188, 3, 307, 32, 37, 375, 38, 40496, 40498, 40701, 40978 40979, 40983, 41142, 41146, 41156, 41157, 458, 46, 6332,	42, 44, 4534, 4538, 469, 470, 50, 54, 60, 1040, 1049, 1063, 1067, 11, 1111, 14, 1462, 1468, 1475, 1476, 1478, 1479, 1480, 1485, 1486, 1487, 1494, 1497, 15, 1501, 16, 18, 181, 182, 22, 23, 23381, 24, 28, 29, 31, 312, 334, 377, 40499, 40536, 40670, 40900, 40966, 40975, 40981, 40982, 40984, 40994, 41138, 41143, 41212, 4134, 4154
nb301	–	1

TABLE V: Meta-optimization search space used to configure Algorithm 2. Some configuration parameters are optimized on a non-linear *scale*, meaning e.g. the optimizer optimizes a value of $\log \mu(1)$ ranging from $\log 2$ to $\log 200$.

Parameter	Meaning	Range	Scale
$\mu(1)$	(first bracket) batch size	$\{2, \ldots, 200\}$	$\log \mu(1)$
batch_method	batch method	{`equal, HB`}	
η_{fid}	fidelity rate	$[2^{1/4}, 2^4]$	$\log \log \eta_{\text{fid}}$
η_{surv}	survival rate	$[1, \infty)$	$1/\eta_{\text{surv}}$
filter_method	SAMPLE method	{SAMPLETOURNAMENT, SAMPLEPROGRESSIVE}	
$\mathbb{P}_{\boldsymbol{\lambda}}(\mathcal{A})$	SAMPLE generating distribution	{`uniform, KDE`}	
$\mathcal{I}_{f_{\text{surr}}}$	surrogate learner	{`KNN1, KKNN7, TPE, RF`}	
$n_{\text{trn}}(0)$	filter sample per tournament round at $t = 0$	$\{1, \ldots, 10\}$	$\log n_{\text{trn}}(0)$
$n_{\text{trn}}(1)$	filter sample per tournament round at $t = 1$	$\{1, \ldots, 10\}$	$\log n_{\text{trn}}(1)$
$N_{\text{s}}^0(0)$	filtering rate of first point in batch at $t = 0$	$[1, 1000]$	$\log N_{\text{s}}^0(1)$
$N_{\text{s}}^0(1)$	filtering rate of first point in batch at $t = 1$	$[1, 1000]$	$\log N_{\text{s}}^0(1)$
$N_{\text{s}}^1(0)$	filtering rate of last point in batch at $t = 0$	$[1, 1000]$	$\log N_{\text{s}}^1(1)$
$N_{\text{s}}^1(1)$	filtering rate of last point in batch at $t = 1$	$[1, 1000]$	$\log N_{\text{s}}^1(1)$
$\rho(0)$	random interleave fraction at $t = 0$	$[0, 1]$	
$\rho(1)$	random interleave fraction at $t = 1$	$[0, 1]$	
filter_mb	surrogate prediction always with maximum r	{`TRUE, FALSE`}	
ρ_{random}	random interleave the same number in every batch	{`TRUE, FALSE`}	

APPENDIX D
ALL γ-VALUES

A table of all optimization γ-values is given in Table VI.

TABLE VI: Optimized configuration parameters, under some constraints. Top: restricted to *batch_method* `HB`, bottom: `equal`. γ_2, γ_3 are further restricted, as described in Table III in Section V. Shown is the overall result. Square brackets show range (for numeric parameters) or list (for discrete parameters) of values found in individual optimization runs when not aggregated as a rough indicator of uncertainty. "(!)" indicates the parameter was forced to the value by a restriction.

Parameter Scenario	γ^* lcbench	γ^* rbv2_super	γ_2 lcbench	γ_2 rbv2_super	γ_3 lcbench	γ_3 rbv2_super
Optimized with *batch_method* `HB`:						
$\mu(1)$	5 [5, 23]	2 [2, 52]	126 [10, 126]	8 [8, 114]	5 [3, 68]	2 [2, 52]
η_{fid}	3.11 [1.25, 3.11]	1.97 [1.97, 6.73]	2.19 [1.68, 10.2]	4.4 [2.04, 4.4]	14.6 [1.45, 14.6]	5.19 [2.24, 5.19]
η_{surv}	2.22 [2.22, 6.1]	6.09 [1.65, 6.09]	3.42 [2.58, 9.19]	3.26 [3.26, 5]	1.15 [1.15, 3.07]	1.20 [1.03, 1.62]
filter_method	PROG [TRN]	PROG [TRN]	PROG [TRN]	PROG [TRN]	TRN (!)	TRN (!)
$\mathbb{P}_{\boldsymbol{\lambda}}(\mathcal{A})$	`KDE`	`uniform [KDE]`	`KDE`	`uniform [KDE]`	`KDE`	`uniform`
$\mathcal{I}_{f_{\text{surr}}}$	`KKNN7 [KNN1]`	`KNN1`	`KKNN7 [KNN1]`	`KNN1`	`KKNN7 [KNN1]`	`KNN1`
$n_{\text{trn}}(0)$	2 [2, 8]	2 [1, 5]	5 [1, 8]	5 [1, 6]	1 (!)	1 (!)
$n_{\text{trn}}(1)$	1 [1, 5]	5 [1, 5]				
$N_s^0(0)$	101 [9.19, 124]	226 [2.03, 226]	39.6 [10.5, 76.7]	125 [125, 163]	570 [73, 570]	155 [155, 561]
$N_s^0(1)$	312 [56.3, 817]	495 [57.1, 533]				
$N_s^1(0)$	28.9 [4.84, 144]	256 [7.19, 256]	31.4 [18.2, 74.6]	481 [480, 563]		
$N_s^1(1)$	8 [8, 654]	99.7 [46.4, 890]				
$\rho(0)$	0.21 [0.12, 0.85]	0.86 [0.68, 0.86]	0.37 [0.2, 0.49]	0.71 [0.49, 0.71]	0.38 [0.12, 0.54]	0.34 [0.34, 0.45]
$\rho(1)$	0.08 [0.08, 0.55]	0.06 [0.01, 0.25]				
filter_mb	TRUE [FALSE]	FALSE [TRUE]	TRUE	TRUE	TRUE [FALSE]	FALSE
ρ_{random}	FALSE [TRUE]	TRUE [FALSE]	FALSE [TRUE]	TRUE [FALSE]	TRUE	TRUE [FALSE]
Optimized with *batch_method* `equal`:						
$\mu(1)$	3 [2, 4]	15 [11, 15]	5 [2, 7]	5 [2, 5]	2 [2, 6]	85 [3, 93]
η_{fid}	2.71 [1.77, 12.2]	1.25 [1.22, 1.43]	2.63 [2.27, 6.01]	8 [1.28, 8]	2.59 [1.46, 2.59]	2.3 [1.36, 12.7]
η_{surv}	2.5 [1.23, 3.36]	18.8 [8.74, 18.8]	1.87 [1.84, 5.49]	3.45 [3.45, 5.53]	3.53 [1.29, 4.86]	6.5 [5.34, 11.3]
filter_method	TRN [PROG]	PROG	TRN [PROG]	PROG [TRN]	TRN (!)	TRN (!)
$\mathbb{P}_{\boldsymbol{\lambda}}(\mathcal{A})$	`KDE`	`KDE [uniform]`	`KDE`	`uniform [KDE]`	`KDE`	`uniform [KDE]`
$\mathcal{I}_{f_{\text{surr}}}$	`KKNN7 [KNN1]`	`KNN1`	`KNN1 [KNN7]`	`KNN1`	`KNN1 [KNN7]`	`KNN1`
$n_{\text{trn}}(0)$	1 [1, 4]	2 [1, 3]	5 [1, 5]	3 [1, 3]	1 (!)	1 (!)
$n_{\text{trn}}(1)$	2 [1, 2]	9 [1, 9]				
$N_s^0(0)$	21.5 [1.63, 309]	39.5 [2.42, 39.5]	169 [43.4, 191]	212 [49.5, 212]	81.3 [24.8, 111]	583 [295, 777]
$N_s^0(1)$	941 [58.2, 991]	18.1 [11.5, 408]				
$N_s^1(0)$	35.4 [7.8, 280]	6.65 [5.43, 391]	4.76 [2.34, 273]	1.71 [1.71, 4.21]		
$N_s^1(1)$	264 [5, 474]	925 [25.4, 925]				
$\rho(0)$	0.32 [0.09, 0.68]	0.83 [0.49, 0.83]	0.34 [0.09, 0.37]	0.34 [0.34, 0.53]	0.27 [0.03, 0.27]	0.96 [0.38, 0.96]
$\rho(1)$	0.16 [0.06, 0.29]	0.03 [0.03, 0.5]				
filter_mb	TRUE	TRUE	TRUE	TRUE	TRUE [FALSE]	TRUE
ρ_{random}	TRUE [FALSE]	FALSE	TRUE [FALSE]	FALSE [TRUE]	TRUE	TRUE [FALSE]

APPENDIX E
ENLARGED OPTIMIZATION CURVE PLOTS

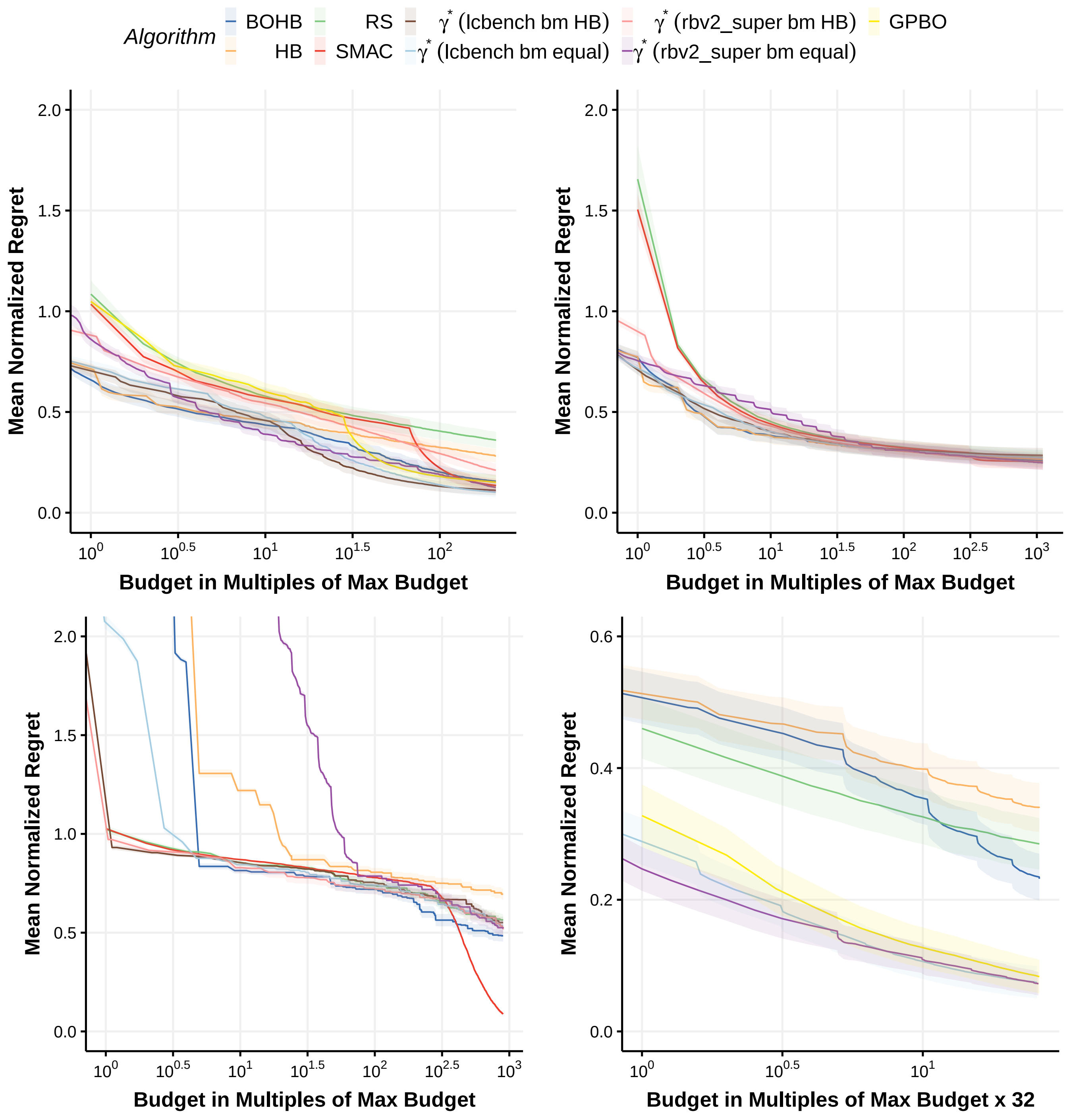


Fig. 6: Enlarged version of Figure 3.

4.7 YAHPO Gym - Design Criteria and a new Multifidelity Benchmark for Hyperparameter Optimization

Contributed Article:
F. Pfisterer, L. Schneider, J. Moosbauer, M. Binder, and B. Bischl. Yahpo gym - an efficient multi-objective multi-fidelity benchmark for hyperparameter optimization. In I. Guyon, M. Lindauer, M. van der Schaar, F. Hutter, and R. Garnett, editors, *Proceedings of the First International Conference on Automated Machine Learning*, volume 188 of *Proceedings of Machine Learning Research*, pages 3/1–39. PMLR, 25–27 Jul 2022

Declaration of contributions FP and LS contributed equally. The core idea for the system originated from FP who also developed initial code and a first version of the system. FP, with help by LS then re-implemented the underlying software. FP, LS, MB collected samples from relevant benchmarks and performance datasets. LS contributed several improvements to software, stability and functionality as well as automated tuning. MB, JM and BB advised throughout this process. LS and FP jointly developed the experiments, which were executed by LS who also contributed implementations of relevant baselines algorithms. FP and LS jointly authored the resulting manuscript with input and improvements by MB, JM and BB.

YAHPO Gym - An Efficient Multi-Objective Multi-Fidelity Benchmark for Hyperparameter Optimization

Florian Pfisterer[1] **Lennart Schneider**[1] **Julia Moosbauer**[1] **Martin Binder**[1] **Bernd Bischl**[1]

[1]LMU Munich

Abstract When developing and analyzing new hyperparameter optimization methods, it is vital to empirically evaluate and compare them on well-curated benchmark suites. In this work, we propose a new set of challenging and relevant benchmark problems motivated by desirable properties and requirements for such benchmarks. Our new surrogate-based benchmark collection consists of 14 scenarios that in total constitute over 700 multi-fidelity hyperparameter optimization problems, which all enable multi-objective hyperparameter optimization. Furthermore, we empirically compare surrogate-based benchmarks to the more widely-used tabular benchmarks, and demonstrate that the latter may produce unfaithful results regarding the performance ranking of HPO methods. We examine and compare our benchmark collection with respect to defined requirements and propose a single-objective as well as a multi-objective benchmark suite on which we compare 7 single-objective and 7 multi-objective optimizers in a benchmark experiment. Our software is available at [`https://github.com/slds-lmu/yahpo_gym`].

1 Introduction

Hyperparameter optimization (HPO) of machine learning (ML) models is a crucial step for achieving good predictive performance [44]. Over the last ten years, a large and still growing set of HPO tuning methods based on different principles has been developed [32, 66, 38]. A particularly interesting development are multi-fidelity methods, which make use of relatively cheap approximations of a given true objective, thereby achieving good performance relatively quickly [46, 22, 36], as well as multi-objective methods, which allow for simultaneous optimization of multiple objectives [41]. While different HPO methods found considerable adoption in practice, it is by no means clear which method performs best under which circumstances. In order to investigate this, it is necessary to evaluate these methods on testbeds that are ideally *i*) highly efficient, *ii*) include a sufficient amount of representative and diverse benchmark instances and *iii*) are easy to set up and integrate with different optimizer APIs. Furthermore, benchmarks have found use in *meta-learning* [70, 75, 60] and *meta-optimization* [50, 54]. In those settings, a larger number of potentially relevant optimization problems is required in order to obtain results that generalize beyond the set of (meta-)training instances. Simultaneously, those applications require a large number of evaluations that make obtaining real evaluations prohibitively expensive, indicating a need for benchmarks that are cheap to query.

Several benchmarks that aim to address this, each of which are collections of multiple benchmark instances, have been proposed [69, 16, 3, 20]. Benchmark instances can be classified into four categories: (i) synthetic functions, (ii) benchmarks incorporating *real* evaluations, (iii) *tabular* benchmarks based on pre-evaluated grid points, and (iv) *surrogate* benchmarks making use of meta-models that approximate the relationship between configurations and performance metrics. These categories have various advantages and drawbacks. Synthetic functions can be evaluated quickly but are often not representative for the type of problems encountered in practice; real evaluations on the other hand are often prohibitively expensive, especially in the context of larger benchmarks and neural architecture search. Tabular benchmarks, while cheap to evaluate, rely on a pre-defined grid which changes the optimization problem and can potentially lead to biases. Surrogate benchmarks are also cheap to query but require high quality surrogates in order to avoid introducing bias. While benchmark suites have found some use in scientific publications, they are not used ubiquitously. This lack of permeation – and consequently the lack of a standard test bed – can result in researchers choosing benchmark problems that favor their own method, leading to the publication of biased results. The problem of *cherry picking*, also termed *rigging the lottery* [15], can be ameliorated through the use of standardized testing infrastructure along with a detailed definition of evaluation criteria that are widely adapted.

We therefore observe a clear need for benchmark libraries that provide unified interfaces to a variety of cheap to evaluate, realistic, and practically relevant benchmarking problems that are defined across diverse search spaces. In this work, we propose *YAHPO Gym*, a *surrogate-based* benchmark library including a collection of over 700 benchmark instances defined across 14 *scenarios*. Scenarios are comprised of evaluations of one given machine learning algorithm on different datasets (= instances) and therefore share the same search space and performance metrics. It contains a versioned set of surrogate models that allow for *multi-fidelity* evaluations of *multiple objectives*. Our library is licensed under the *Apache 2.0* license and can be freely used and extended by the community. Usage and available functionality is extensively documented[1].

Contributions: We introduce YAHPO Gym, a surrogate-based benchmark for machine-learning HPO. We conceptually demonstrate that tabular benchmarks may induce bias in performance estimation and ranking of HPO methods, and that this happens to a lesser degree with surrogate benchmarks. We argue that our surrogate benchmark YAHPO Gym meets all desiderata for a good benchmark, providing faithful results, fast evaluation, relevant problems and realistic objective landscapes both on local as well as global scales. In order to demonstrate this, we conduct an extensive evaluation of the proposed surrogates indicating that our surrogate models indeed provide high quality approximations. We propose two benchmark suites for *single-objective* and *multi-objective* evaluation comprised of a subset of our instances and demonstrate how they can be used with YAHPO Gym in a *multi-fidelity* and a *multi-objective* optimization benchmark.

2 Related Work

Several efforts to provide unified testbeds for black-box optimization exist. For general purpose black-box optimization, COCO [30] provides a collection of various synthetic black-box benchmark functions, while *kurobako* [57] is a collection of various general black-box optimizers and benchmark problems. Similarly, *Bayesmark* [69] includes several benchmarks for Bayesian Optimization on real problems and *LassoBench* [64] provides a benchmark for high-dimensional optimization problems. *HPOlib* [16] was one of the first to propose a common test bed for empirically assessing the performance of HPO methods. It provides a common API to access synthetic test functions, real-world HPO problems, tabular benchmarks as well as some surrogate benchmarks and found use in empirical benchmark studies [7]. Its successor *HPOBench* [20] offers similar capabilities, focussing on reproducible containerized benchmarks. It offers 12 benchmark scenarios and more than 100 test instances. Recently, [3] introduced *HPO-B*, a large-scale reproducible (tabular) benchmark for black-box HPO based on OpenML [71]. *HPO-B*[2] relies on 16 search spaces that were evaluated sparsely on 101 datasets. *PROFET* [39] in contrast is not based on real datasets but uses a generative meta-model to generate synthetic but realistic benchmark instances. In the past, tabular benchmarks have been used frequently to speed up experiments in the context of HPO [66, 23, 73, 24] and Neural Architecture Search (NAS) (c.f. [51]). Eggensperger et al. [18] compared different instance surrogate models for 9 different HPO problems and

[1]Documentation and data are available at `https://github.com/slds-lmu/yahpo_gym`

[2]We consider the published v2 version for comparison. Surrogates are only available in the v3 version.

Table 1: Comparison of HPO Benchmark Suites.

Suite	Types	#Collections	#HPs	MF	MO	TF	Async	H	Time†	Memory†
YAHPO Gym	S	14	2-38	✓	✓	✓	(-)	✓	$0.4^{*}s$	0.1 GB
HPOBench	R/T/S	12	4-26	✓	✓	(-)	–	(-)	12.2s	0.2 GB
HPO-B (v2)	T/(S)	16	2-18	–	–	✓	–	–	18.8s	3.7 GB

MF: Multi-fidelity; MO: Multi-objective, TF: Transfer-HPO, Async: Asynchronous evaluation; H: Hierarchical search spaces.
✓: fully supported; (-): partially supported; -: not supported; R/T/S:real/tabular/surrogate.
†: Runtime and memory footprint for 300 iterations of random search on an SVM instance. ∗: allowing for batched evaluation, YAHPO Gym takes only 0.13s).

concluded that the results of benchmarks run on surrogate models generally closely mimic those of benchmarks using the actual evaluations that they are derived from, if performance measures of the surrogate models indicate that they predict the underlying objective values sufficiently well (cross-validated Spearman's ρ between 0.9 and 1 [18]). Similar observations have been made in the context of algorithm configuration [19] and NAS [65].

We compare YAHPO Gym with the recently published benchmarks HPOBench [20] and HPO-B [3] in Table 1. Our library relies on high quality surrogates that allow for *multi-fidelity* as well as *multi-objective* evaluation. While existing benchmark suites could in principle be used to construct multi-objective benchmarks, they do not offer full support: HPOBench contains only few instances that allow evaluating multiple metrics and offers no unified API to query those, while HPO-B does not support multiple objectives at all. Furthermore, neither propose a concrete evaluation protocol, opening up a multiplicity of (benchmark) design choices which can lead to inconclusive results (c.f. [56]). Instead of relying on *containerization* to allow for portability, our library relies on neural network surrogates compressed using *ONNX* [4], allowing for reproducibility and portability while simultaneously being extremely fast and efficient due to minimal overhead. This is demonstrated in a small experiment where we measure runtime and memory consumption for evaluating 300 random configurations on SVM search spaces also shown in Table 1, demonstrating that our software is more time and memory efficient. While YAHPO Gym provides the flexibility to design and execute any subset of the provided benchmarks, we also propose two fully specified testbeds for single- and multi-objective optimization that were specifically selected to cover a diverse set of relevant instances while being less extensive. Seed details in Supplement B.2.

3 Background

3.1 Hyperparameter Optimization

An ML *learner* or *inducer* $\mathcal{I}$ configured by hyperparameters $\boldsymbol{\lambda} \in \Lambda$ maps a data set $\mathcal{D} \in \mathbb{D}$ to a model $\hat{f}$, i.e., $\mathcal{I} : \mathbb{D} \times \Lambda \to \mathcal{H}, (\mathcal{D}, \boldsymbol{\lambda}) \mapsto \hat{f}$. Hyperparameter optimization (HPO) methods for ML aim to identify a well-performing hyperparameter configuration (HPC) $\boldsymbol{\lambda} \in \tilde{\Lambda}$ for $\mathcal{I}_{\boldsymbol{\lambda}}$ [11]. Typically, the considered search space $\tilde{\Lambda} \subset \Lambda$ is a subspace of the set of all possible HPCs: $\tilde{\Lambda} = \tilde{\Lambda}_1 \times \tilde{\Lambda}_2 \times \cdots \times \tilde{\Lambda}_d$, where $\tilde{\Lambda}_i$ is a bounded subset of the domain of the i-th hyperparameter Λ_i. This $\tilde{\Lambda}_i$ can be either real, integer, or category valued, and the search space can contain dependent hyperparameters, leading to a possibly hierarchical search space. We formally define the (potentially multi-objective) HPO problem as:

$$\boldsymbol{\lambda}^* \in \operatorname*{arg\,min}_{\boldsymbol{\lambda} \in \tilde{\Lambda}} c(\boldsymbol{\lambda}), \qquad \text{with} \qquad c : \tilde{\Lambda} \to \mathbb{R}^m, \tag{1}$$

where $\boldsymbol{\lambda}^*$ denotes the theoretical optimum and c maps an arbitrary HPC to (possibly multiple) target metrics. The classical HPO problem is defined as $\boldsymbol{\lambda}^* \in \arg\min_{\boldsymbol{\lambda} \in \tilde{\Lambda}} \widehat{\mathrm{GE}}(\boldsymbol{\lambda})$, i.e., the goal is to minimize the estimated generalization error, see [11] for further details. Instead of optimizing only for predictive performance, other metrics such as model sparsity or computational efficiency of prediction (e.g., MACs and FLOPs or model size and memory usage) could be included, resulting in a multi-objective HPO problem [62, 31, 8, 58, 28]. $c(\boldsymbol{\lambda})$ is a black-box function, as it usually has no closed-form mathematical representation, and analytic gradient information is generally not available. Furthermore, the evaluation of $c(\boldsymbol{\lambda})$ can take a significant amount of time. Therefore, the minimization of $c(\boldsymbol{\lambda})$ forms an *expensive black-box* optimization problem.

Many HPO problems allow for approximations of the objective to a varying fidelity, making *multi-fidelity optimization* a viable option [46, 62, 36]. E.g., in the context of fitting neural networks, it is possible to stop or pause training runs early when performance does not indicate a promising final result [67]. Another possibility is given by reducing the fraction of the dataset $\mathcal{D}_{\text{train}}$ used for training [38], since the complexity of evaluating $c(\boldsymbol{\lambda})$ is often at least linear in $|\mathcal{D}_{\text{train}}|$. Formally, the possibility of multi-fidelity evaluation can be represented in the form of a "budget" hyperparameter which we denote by λ_{budget} as a component of $\boldsymbol{\lambda}$.

3.2 Hyperparameter Optimization Benchmarks

Benchmark suites are comprised of a set of benchmark *instances* that each define an optimization problem to be solved. We formally define benchmark instances adapted from [20] as:

Definition 1 (Benchmark Instance) *A benchmark instance consists of a function $g : \Lambda \longrightarrow \mathbb{R}^m, m \in \mathbb{N}^+$, and a bounded hyperparameter space $\tilde{\Lambda}$ which is the Cartesian product of hyperparameters $\tilde{\Lambda}_1, \ldots, \tilde{\Lambda}_d$. Multi-fidelity benchmarks can be queried at lower fidelities by varying the budget parameter $\tilde{\Lambda}_{budget} \in \tilde{\Lambda}$.While hyperparameters $\tilde{\Lambda}_i$ can be continuous, integer, ordinal or categorical, we require at least ordinal scales for the fidelity parameter(s) Λ_{budget}. We call a benchmark instance multi-objective if the number of objectives $m > 1$ and single-objective otherwise.*

We consider HPO benchmark instances estimating the generalization error $g(\boldsymbol{\lambda}) = \widehat{\text{GE}}(\mathcal{I}, \mathcal{J}, \rho, \boldsymbol{\lambda})$ given an inducer $\mathcal{I}$, resampling $\mathcal{J}$, and performance metric(s) ρ, along with other possibly relevant metrics (computational cost, memory, ...). *Real* instances are based on actually performing these evaluations during the benchmark, while *tabular* instances are based on a fixed set of prerecorded evaluations. Instances based on *surrogates* in turn approximate the functional relationship between $\boldsymbol{\lambda}$ and $g(\boldsymbol{\lambda})$. For clarity, we provide more precise definitions of *synthetic, tabular* and *surrogate* instances in Section B.3 of the supplement. *Real* instances rely on live evaluations of the generalization error and are therefore often prohibitively computationally expensive, especially when considering larger benchmarks or meta-learning scenarios across many tasks [70, 60, 25]. Practitioners therefore often rely on *tabular* or *surrogate* benchmarks for large benchmark studies because they are often cheaper to evaluate by orders of magnitude. For *tabular* benchmarks, a large collection of pre-computed hyperparameter performance mappings is provided, which serves as a look-up table during runs of HPO methods. This has the downside of constraining the search space to precomputed evaluations, essentially turning the optimization problem from a *continuous/mixed space* to a *discrete* optimization problem. *Surrogate* benchmarks can strike a balance between the efficiency and faithful approximation to the real problem by learning the functional relationship between hyperparameters and performance values yielding an approximation $\hat{g}(\boldsymbol{\lambda})$ of $g(\boldsymbol{\lambda})$. This allows evaluations across the full search space Λ while being considerably cheaper to evaluate. The usefulness of surrogates in turn relies on the approximation quality of the surrogate model. We present an in-depth analysis of approximation qualities of the surrogates employed in YAHPO Gym in Supplement E.1.

Definition 2 (Benchmark Scenario) *A benchmark scenario consists of a set of K functions $g_k : \Lambda \longrightarrow \mathcal{Y} \subseteq \mathbb{R}^m, m \in \mathbb{N}^+, k \in \{1, ..., K\}$ corresponding to a set of* Benchmark Instances. *Each instance within a scenario shares the same bounded hyperparameter space $\tilde{\Lambda}$ (and therefore fidelity parameters) as well as the same co-domain $\mathcal{Y}$.*

Table 2: YAHPO Gym Benchmarks.

Scenario	Search Space	# Instances	Target Metrics	Fidelity	H
rbv2_super	38D: Mixed	103	9: perf(6) + rt(2) + mem	fraction	✓
rbv2_svm	6D: Mixed	106	9: perf(6) + rt(2) + mem	fraction	✓
rbv2_rpart	5D: Mixed	117	9: perf(6) + rt(2) + mem	fraction	
rbv2_aknn	6D: Mixed	118	9: perf(6) + rt(2) + mem	fraction	
rbv2_glmnet	3D: Mixed	115	9: perf(6) + rt(2) + mem	fraction	
rbv2_ranger	8D: Mixed	119	9: perf(6) + rt(2) + mem	fraction	✓
rbv2_xgboost	14D: Mixed	119	9: perf(6) + rt(2) + mem	fraction	✓
nb301	34D: Categ.	1	2: perf(1) + rt(1)	epoch	✓
lcbench	7D: Cont.	34	6: perf(5) + rt(1)	epoch	
iaml_super	28D: Mixed	4	12: perf(4) + inp(3) + rt(2) + mem(3)	fraction	✓
iaml_rpart	4D: Cont.	4	12: perf(4) + inp(3) + rt(2) + mem(3)	fraction	
iaml_glmnet	2D: Cont.	4	12: perf(4) + inp(3) + rt(2) + mem(3)	fraction	
iaml_ranger	8D: Mixed	4	12: perf(4) + inp(3) + rt(2) + mem(3)	fraction	✓
iaml_xgboost	13D: Mixed	4	12: perf(4) + inp(3) + rt(2) + mem(3)	fraction	✓

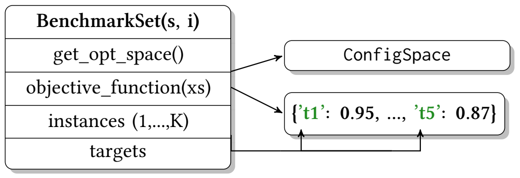
Mixed = numeric and categorical hyperparameters; perf = performance measures; rt = train/predict time; mem = memory consumption; inp = interpretability measures; H = Hierarchical search space. We do not include the fidelity parameter in the search space dimensionality.

A scenario is therefore a collection of instances sharing the same search space and objective(s), e.g., allowing for hyperparameter transfer learning between instances of the scenario. *Benchmark Suites* in turn are sets of instances that do not need to share the same objectives, but instead can consist of instances stemming from different scenarios.

4 YAHPO Gym

Motivated by the need for efficient and faithful benchmarks for HPO, we develop YAHPO Gym based on a set of *Criteria for HPO Benchmarks* discussed in Supplement B.1. YAHPO Gym is explicitly designed to use surrogate-based benchmarks only. It consists of a collection of 14 *scenarios* that can be evaluated across a total of $\sim$ 700 instances. Each benchmark instance consists of an objective function that is parameterized in the form of a `ConfigSpace` Python object [48], making the search space computer-readable and readily usable with a range of existing HPO implementations. The objective function generates a prediction using the instance surrogate model, which is a compressed neural network. Table 2 provides an overview of all benchmark scenarios available in YAHPO Gym. We describe data sources as well as the full search spaces in Supplement F. We want to highlight the *rbv2_super* collection, which reflects an AutoML pipeline: It is, to our knowledge, the first available benchmark simulating a combined algorithm and hyperparameter selection problem [68] in the form of a high dimensional hierarchical search space by introducing the algorithm as an additional tunable hyperparameter.

In YAHPO Gym, every scenario allows for querying objective values at lower fidelities, enabling efficient benchmarking of multi-fidelity HPO methods. Analogously, every benchmark allows for returning multiple target metrics as criteria, enabling benchmarking of multi-objective HPO methods. Finally, almost all benchmark scenarios provide problems on a large number of instances (ranging from 34 to 119), allowing for benchmarking of transfer-learning HPO methods. Predictions as well as sampling can be made reproducible through seeding. In order to achieve *portability* while still being *efficient*, YAHPO Gym uses fitted neural networks compressed via `ONNX` [4] as surrogate models. Our neural networks are `ResNets` for tabular data [27] consisting of up to 8 layers with a width of up to 512 and hyperparameters individually tuned for each scenario. We refer the reader to Supplement D for details regarding architecture and fitting procedure. Surrogate models have very small memory and inference time overhead and are compatible across platforms



(a) YAHPO Gym's core functionality (**s**: scenario, **i**: instance, **xs**: configuration). Evaluating `objective_function` for a given configuration **xs** returns a dictionary of predicted metrics for a given scenario and instance.

```
from yahpo_gym import *
b = BenchmarkSet('lcbench', instance='3945')
# Sample a point from the ConfigSpace
xs = b.get_opt_space().sample_configuration(1)
# Evaluate the configuration
b.objective_function(xs)
```

(b) Python code for instantiating a benchmark instance, sampling a new configuration and evaluating the objective function.

and operating systems. In contrast to other benchmarks, evaluating $c(\boldsymbol{\lambda})$ requires only $10 - 100$ ms and only 100 MB of memory. In fact, YAHPO Gym's current infrastructure is so lightweight, it can easily be integrated in any existing toolbox or benchmark suite.

4.1 Suites: YAHPO-SO & YAHPO-MO

Together with YAHPO Gym, we propose two carefully selected *benchmark suites*. They constitute a proposal for surrogate-based benchmarks of HPO problems. We call those YAHPO-SO (single-objective, 20 instances) and YAHPO-MO (multi-objective, 25 instances). Together with the set of instances, we provide specific evaluation criteria, such as the budget available for optimization and number of stochastic replications as well as metrics to be used and fully specified search spaces which can be obtained from our software. Instances were selected across all scenarios taking into account approximation quality of the underlying surrogate and diversity. We consider those benchmarks a first draft for such a benchmark set (version **v1**) and explicitly invite the community to jointly work on a larger, more comprehensively evaluated set of benchmark instances. Details with respect to how instances were selected, and a full list of included instances, can be found in Section C.2 in the Supplement. We conduct a benchmark providing anytime performance for a large variety of baselines on the proposed benchmark suites.

5 Tabular or Surrogate Benchmarks?

Consider the true objective $c(\boldsymbol{\lambda})$ of a *real* benchmark instance with $c : \tilde{\Lambda} \to \mathbb{R}$ in the single-objective setting. In a *tabular* benchmark, the domain of the objective function is implicitly discretized into a finite grid $\tilde{\Lambda}_{\text{discrete}}$ of the original domain and pre-evaluated at these points and the benchmark objective $\hat{c}_{\text{tabular}}(\boldsymbol{\lambda})$ is thus the original $c(\boldsymbol{\lambda})$ restricted to $\tilde{\Lambda}_{\text{discrete}}$. The extent to which discretization affects the faithfulness of tabular benchmarks depends on the nature and dimensionality of the search space: It disregards local structure in the response function and might even impose fixed fidelity schedules, should evaluations not be available at all budget levels. In order to assess the magnitude of this effect, we investigate the practical effects of discretization in the following experiment by comparing 8 black-box optimizers on *tabular*, *surrogate* and *real* versions of 5 synthethic multi-fidelity functions of varying dimensionality (Branin2D, Currin2D, Hartmann3D/6D, and Borehole8D [36]). The tabular benchmark is constructed by drawing and evaluating 10^6 points from a grid. Surrogates are then fitted using those points. We compare Random search (`RS`), several versions of Bayesian optimization (`BO`) and Hyperband (`HB`, [46]) across all settings. `BO` is configured with algorithm surrogate model either a Gaussian process (`BO_GP`), ensemble of feed-forward neural networks (`NN`, [74]) or random forest (`BO_RF`, [13]) and acquisition function optimizer either Nelder-Mead/exhaustive search[3] (`*_DF` [55]) or random search (`*_RS`). We describe additional details regarding the benchmark setup in Supplement E.1 and briefly present results: Figure 2 shows the anytime performance and mean rank of each HPO method split for the

[3]for tabular benchmarks

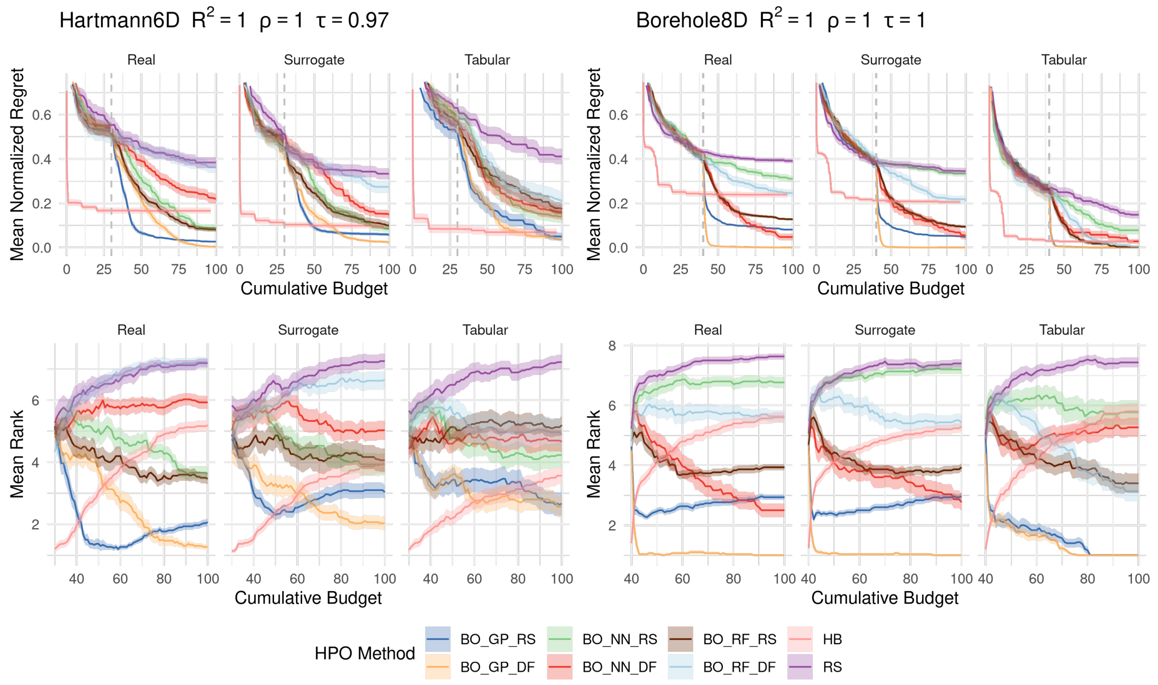


Figure 2: Mean normalized regret (top) and mean ranks (bottom) of different HPO methods on different benchmarks. Ribbons represent standard errors. The gray vertical line indicates the cumulative budget used for the initial design of BO methods. Performance measures of the surrogate benchmarks are stated after the benchmark function. 30 replications.

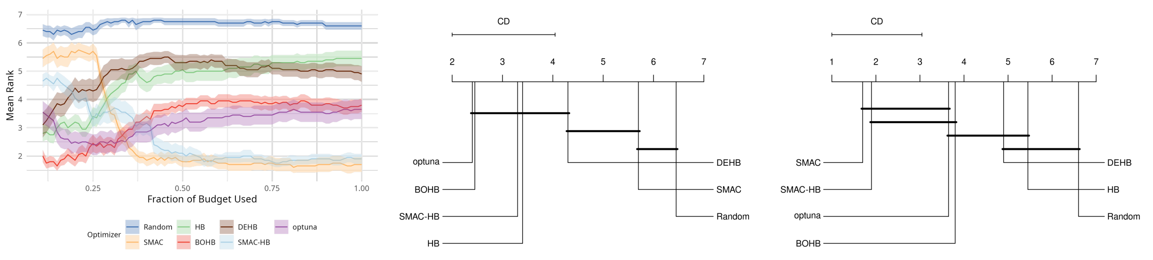
real, surrogate, and tabular benchmark on the Hartmann6D and Borehole8D test functions. We observe very similar performance traces of HPO methods on surrogate versions of benchmarks compared to real versions (Figure 2, top). However, in tabular benchmarks, we notice that for some problems, the BO methods converge substantially faster to a lower mean normalized regret (especially for `BO_GP_*`), which can possibly be explained by the much simpler infill optimization problem solved in the tabular case. Moreover, Hyperband appears to consistently perform better on tabular benchmarks. We further investigate average rankings over all replications (Figure 2, bottom). Each benchmark function yields an average ranking of HPO methods (e.g., with respect to final performance). Using consensus rankings, we can arrive at a single ranking over all benchmark functions [52] for a given benchmark type. We use the optimization based symmetric difference (SD) [37] minimizing rank reversals to compare both the surrogate and tabular inferred consensus rankings with the "ground truth" real function consensus ranking. We observe that consensus rankings obtained using surrogate benchmarks (permutation order 2) match more closely than tabular benchmarks (permutation order 5). We again provide additional details in Supplement E.1.

6 A Benchmark of HPO Methods on YAHPO Gym

We now demonstrate how YAHPO Gym can be used in practice to benchmark different HPO methods. We benchmark 7 single-objective HPO methods on YAHPO-SO and 7 multi-objective HPO methods on YAHPO-MO and want to answer the following research questions: (**RQ1**) *Do multi-fidelity (single-objective) HPO methods improve over full-fidelity methods?* (**RQ2**) *Do advanced multi-objective HPO methods improve over Random Search?*

6.1 RQ1: Do multi-fidelity (single-objective) HPO methods improve over full-fidelity methods?

We compare Random Search and SMAC (SMAC4HPO facade; [49]) to the multi-fidelity methods Hyperband [46], BOHB [22], DEHB [5], SMAC-HB (SMAC4MF facade; [49]) and optuna ([2]; TPE sampler and median pruner following successive halving steps). More details on the experimental setup and HPO methods is given in Supplement E.2. All optimizers are run for a total budget of $\lceil 20 + 40 \cdot \sqrt{\text{SEARCH\_SPACE\_DIM}} \rceil$ full-fidelity evaluations with 30 replications. Figure 3a shows the average rank of HPO methods with respect to their anytime performance. Figure 3b and Figure 3c show critical difference plots ($\alpha = 0.05$) of mean ranks after 25% and 100% of the optimization budget. The corresponding Friedman tests indicate significant differences ($p < 0.001$) in both cases. We observe that all multi-fidelity optimizers outperform Random Search with respect to intermediate performance (25% of optimization budget) and optuna, BOHB, SMAC-HB and Hyperband also outperform SMAC. With respect to final performance, SMAC takes the lead closely followed by SMAC-HB with other multi-fidelity optimizers slightly falling behind. We conclude that multi-fidelity HPO methods indeed improve over full-fidelity methods, but only with respect to intermediate performance. Our results are in line with what has been reported in other benchmarks [20] with the exception that optuna seems more competitive in our benchmark, while DEHB is less competitive. One reason for this difference might be that we include hierarchical search spaces in contrast to previous work.



(a) Mean ranks of HPO methods. x-axis starts after 10%.

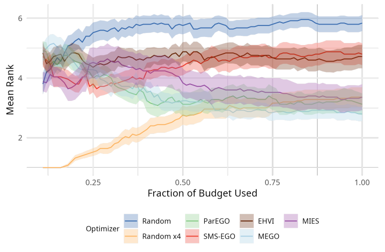
(b) Critical differences plot for mean ranks of HPO methods after 25% of the optimization budget.

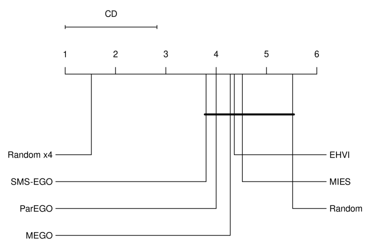
(c) Critical differences plot for mean ranks of of HPO methods after 100% of the optimization budget.

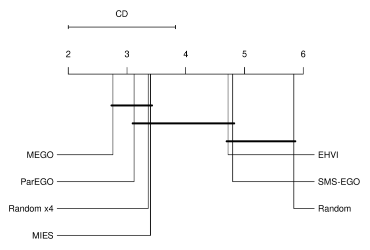
Figure 3: Results of YAHPO-SO single-objective benchmark across 7 optimizers (20 Instances).

6.2 RQ2: Do advanced multi-objective HPO methods improve over Random Search?

We compare Random search, Random search x4 (random search with quadrupled budget as a strong baseline), ParEGO [41], SMS-EGO [61], EHVI [21], MEGO [34] and MIES [47] on multi-objective HPO problems with $2 - 4$ objectives. More details on the experimental setup and HPO methods is given in Supplement E.3. All optimizers are run for a total budget of $\lceil 20 + 40 \cdot \sqrt{\text{SEARCH\_SPACE\_DIM}} \rceil$ full-fidelity evaluations for 30 replications. Figure 4a shows the average rank of HPO methods with respect to their anytime performance (determined based on the normalized Hypervolume Indicator). Figure 4b and Figure 4c show critical difference plots ($\alpha = 0.05$) of these ranks after 25% and 100% of the optimization budget. The corresponding Friedman tests indicate significant differences ($p < 0.001$) in both cases. We observe that not all methods significantly improve over Random Search with respect to final performance, i.e., EHVI and SMS-EGO fail to do so. Especially with respect to intermediate performance (25% of optimization budget), Random x4 outperforms all competitors. However, with respect to final performance, MEGO, ParEGO and MIES yield similar performance catching up to Random x4. We conclude that, in general, advanced multi-objective HPO methods improve over Random Search but also want to highlight that optimizer performance strongly varies with respect to the different benchmark instances.







(a) Mean ranks of HPO methods. x-axis starts after 10% of the optimization budget has been used.

(b) Critical differences plot for differences in ranks of HPO methods after 25% of optimization budget.

(c) Critical differences plot for differences in ranks of HPO methods after 100% of optimization budget.

Figure 4: Results of the YAHPO-MO multi-objective benchmark across 7 optimizers (25 Instances).

In total, both benchmarks described in this section took the equivalent of 138.7 CPU days using YAHPO Gym. We estimate that the YAHPO-SO benchmark, would take 15.34 CPU **years** when running real benchmarks, while our benchmark using YAHPO Gym took only 388 CPU **hours**, essentially speeding up evaluation by a factor of ~ 350.

7 Conclusions, Limitations and Broader Impact

We present YAHPO Gym, a multi-fidelity, multi-objective benchmark for HPO. Our benchmark is based on surrogates, which strike a favorable trade-off between faithfulness and efficiency, which we demonstrate in various experiments throughout our paper before conducting a large scale benchmark of modern single- and multi-objective optimizers. An as of yet under-explored domain are asynchronous optimization algorithms, which have recently gained popularity [45]. This has been studied in surrogate-based benchmarks by predicting runtimes and pausing the objective function for the predicted runtime, lowering computational demand for benchmarks but leading to a large waiting time [22]. In future work we plan on introducing faster-than-real time asynchronous benchmarking based on predicted runtimes.

Limitations. YAHPO Gym is based on surrogate models and therefore heavily relies on the faithfulness of those models in order to allow for valid conclusions. We have comprehensively evaluated surrogate models and provide a detailed report of performance metrics, hoping to demonstrate the faithfulness of our surrogates, but can only do so to a certain degree. We are furthermore aware that the real HPO problems modeled in our surrogates are in fact stochastic, and results can vary depending on randomness of the fitting procedure, data splits or initialization. We therefore provide a set of *noisy* surrogate models that intend to model the stochasticity of the problems using an ensemble of neural networks, but simultaneously allow for full control of the stochastic process by using random seeds.

Broader Impact. This manuscript presents a set of surrogate-based benchmarks for HPO. As such, our work does not have direct implications on society or individuals, but can lead to such indirectly if new methods are developed based on it. We would like to emphasize the possible **societal & environmental benefits**. First, we hope our benchmarks can improve the state of benchmarking in hyperparameter optimization contexts, leading to better tracking of progress in the discipline. Second, and more important, we hope that experiments based on YAHPO Gym can drastically reduce **computational cost** of hyperparameter optimization experiments. This type of experiments is usually extremely expensive, if real experiments are run for the evaluation of each HPC, which can be sped up by large factors if cheap approximations through surrogates are available.

Acknowledgements. The authors of this work take full responsibilities for its content. This work was supported by the German Federal Ministry of Education and Research (BMBF) under Grant No. 01IS18036A. Lennart Schneider is supported by the Bavarian Ministry of Economic Affairs, Regional Development and Energy through the Center for Analytics Data Applications (ADACenter) within the framework of BAYERN DIGITAL II (20-3410-2-9-8). This work has been carried out by making use of AI infrastructure hosted and operated by the Leibniz-Rechenzentrum (LRZ) der Bayerischen Akademie der Wissenschaften and funded by the German Federal Ministry of Education and Research (BMBF) under Grant No. 01IS18036A. The authors gratefully acknowledge the computational and data resources provided by the Leibniz Supercomputing Centre (`www.lrz.de`). The authors gratefully acknowledge the computational and data resources provided by the *ARCC Teton HPC* [1].

8 Reproducibility Checklist

f

1. For all authors...

   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]

   (b) Did you describe the limitations of your work? [Yes] See Section 7

   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See Section 7

   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]

2. If you are including theoretical results...

   (a) Did you state the full set of assumptions of all theoretical results? [N/A]

   (b) Did you include complete proofs of all theoretical results? [N/A]

3. If you ran experiments...

   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results, including all requirements (e.g., `requirements.txt` with explicit version), an instructive `README` with installation, and execution commands (either in the supplemental material or as a URL)? [Yes] The full code for experiments, figures and table can be obtained from the following GitHub repositories:

      i. Software: `https://github.com/slds-lmu/yahpo_gym`
      ii. Documentation: `https://slds-lmu.github.io/yahpo_gym/`
      iii. Surrogates & Search Spaces: `https://github.com/slds-lmu/yahpo_data`
      iv. Code for Results: `https://github.com/slds-lmu/yahpo_exps`

   (b) Did you include the raw results of running the given instructions on the given code and data? [Yes] We make the full data used to train our surrogates available at `https://syncandshare.lrz.de/getlink/fiCMkzqj1bv1LfCUyvZKmLvd/`

   (c) Did you include scripts and commands that can be used to generate the figures and tables in your paper based on the raw results of the code, data, and instructions given? [Yes] See `https://github.com/slds-lmu/yahpo_exps`

(d) Did you ensure sufficient code quality such that your code can be safely executed and the code is properly documented? [Yes]

(e) Did you specify all the training details (e.g., data splits, pre-processing, search spaces, fixed hyperparameter settings, and how they were chosen)? [Yes] See Supplement F for search spaces, the code repository as well as the software repository for further fixed hyperparameters

(f) Did you ensure that you compared different methods (including your own) exactly on the same benchmarks, including the same datasets, search space, code for training and hyperparameters for that code? [Yes] Yes, this is explicitly guaranteed by our software.

(g) Did you run ablation studies to assess the impact of different components of your approach? [Yes] Partially, see sections throughout the supplementary material.

(h) Did you use the same evaluation protocol for the methods being compared? [Yes] Yes

(i) Did you compare performance over time? [Yes] Anytime performances are reported in all relevant figures throughout the paper.

(j) Did you perform multiple runs of your experiments and report random seeds? [Yes] We perform 30 replications for each experiments. Random seeds can be obtained from the acompanying code.

(k) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] All figures reporting experimental results include error bars.

(l) Did you use tabular or surrogate benchmarks for in-depth evaluations? [Yes] Surrogate benchmarks

(m) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] We state the total computation as well as CO_2 equivalent in the respective section and briefly summarize here: Tuning and fitting surrogates required a total of 45 GPU-days (116 kg CO_2-equivalent on NVIDIA DGX-A100 instances) while the main experiments require 138.7 CPU days across all replications (262 kg CO_2 equivalent). The tabular vs surrogate benchmark required 22 CPU-hours (2kg CO_2) equivalent.

(n) Did you report how you tuned hyperparameters, and what time and resources this required (if they were not automatically tuned by your AutoML method, e.g. in a NAS approach; and also hyperparameters of your own method)? [Yes] We report tuning of surrogates in the supplementary material.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes] Yes, throughout the paper and explicitly in Supplement F for datasets we base our surrogates on.

(b) Did you mention the license of the assets? [Yes] Yes, see Supplement F.

(c) Did you include any new assets either in the supplemental material or as a URL? [Yes] Yes, trained surrogates are available at `https://github.com/slds-lmu/yahpo_data`.

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] Data is meta-data about ML experiments and we do not consider any personal data. All used data is available via OSS Licenses and no consent was required.

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [Yes] Data is only metadata about ML experiments.

5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A] No crowd sourcing.
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A] No IRB was required.
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

References

[1] Advanced Research Computing Center. Teton computing environment. `https://doi.org/10.15786/M2FY47`, 2018. University of Wyoming.

[2] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[3] Sebastian Pineda Arango, Hadi S. Jomaa, Martin Wistuba, and Josif Grabocka. HPO-B: A Large-Scale Reproducible Benchmark for Black-Box HPO based on OpenML. *arXiv:2106.06257 [cs]*, 2021.

[4] ONNX authors. ONNX. `https://github.com/onnx/onnx`, 2022.

[5] Noor Awad, Neeratyoy Mallik, and Frank Hutter. DEHB: Evolutionary Hyberband for Scalable, Robust and Efficient Hyperparameter Optimization. *arXiv:2105.09821 [cs]*, 2021.

[6] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, pages 2546–2554, 2011.

[7] James Bergstra, Brent Komer, Chris Eliasmith, and David Warde-Farley. Preliminary Evaluation of Hyperopt Algorithms on HPOLib. In *ICML Workshop on Automatic Machine Learning*, 2014.

[8] Martin Binder, Julia Moosbauer, Janek Thomas, and Bernd Bischl. Multi-objective Hyper-parameter Tuning and Feature Selection Using Filter Ensembles. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, GECCO '20, page 471479, New York, NY, USA, 2020. Association for Computing Machinery.

[9] Martin Binder, Florian Pfisterer, Michel Lang, Lennart Schneider, Lars Kotthoff, and Bernd Bischl. mlr3pipelines - Flexible machine learning pipelines in R. *Journal of Machine Learning Research*, 22(184):1–7, 2021.

[10] Martin Binder, Florian Psterer, and Bernd Bischl. Collecting Empirical Data About Hyper-parameters for Data Driven AutoML. In *ICML Workshop on Automatic Machine Learning*, 2020.

[11] Bernd Bischl, Martin Binder, Michel Lang, Tobias Pielok, Jakob Richter, Stefan Coors, Janek Thomas, Theresa Ullmann, Marc Becker, Anne-Laure Boulesteix, Difan Deng, and Marius Lindauer. Hyperparameter Optimization: Foundations, Algorithms, Best Practices and Open Challenges. *arXiv:2107.05847 [cs, stat]*, 2021.

[12] Bernd Bischl, Pascal Kerschke, Lars Kotthoff, Marius Lindauer, Yuri Malitsky, Alexandre Fréchette, Holger Hoos, Frank Hutter, Kevin Leyton-Brown, Kevin Tierney, et al. Aslib: A benchmark library for algorithm selection. *Artificial Intelligence*, 237:41–58, 2016.

[13] L. Breiman. Random Forests. *Machine Learning*, 45(1):5–32, 2001.

[14] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. Wide & Deep Learning for Recommender Systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, DLRS 2016, pages 7–10, New York, NY, USA, 2016. Association for Computing Machinery.

[15] Mostafa Dehghani, Yi Tay, Alexey A. Gritsenko, Zhe Zhao, Neil Houlsby, Fernando Diaz, Donald Metzler, and Oriol Vinyals. The Benchmark Lottery. *arXiv:2107.07002 [cs]*, 2021.

[16] K. Eggensperger, M. Feurer, F. Hutter, J. Bergstra, J. Snoek, H. Hoos, and K. Leyton-Brown. Towards an Empirical Foundation for Assessing Bayesian Optimization of Hyperparameters. In *NIPS Workshop on Bayesian Optimization in Theory and Practice*, 2013.

[17] Katharina Eggensperger, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Surrogate Benchmarks for Hyperparameter Optimization. In *MetaSel@ ECAI*, pages 24–31, 2014.

[18] Katharina Eggensperger, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Efficient Benchmarking of Hyperparameter Optimizers via Surrogates. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, pages 1114–1120, Austin, Texas, 2015. AAAI Press.

[19] Katharina Eggensperger, Marius Lindauer, Holger H. Hoos, Frank Hutter, and Kevin Leyton-Brown. Efficient Benchmarking of Algorithm Configurators via Model-based Surrogates. *Machine Learning*, 107(1):15–41, 2018.

[20] Katharina Eggensperger, Philipp Müller, Neeratyoy Mallik, Matthias Feurer, René Sass, Aaron Klein, Noor Awad, Marius Lindauer, and Frank Hutter. Hpobench: A collection of reproducible multi-fidelity benchmark problems for hpo, 2021.

[21] Michael T. M. Emmerich. Single- and multi-objective evolutionary design optimization assisted by Gaussian random field metamodels. *PhD Dissertation*, 2005.

[22] Stefan Falkner, Aaron Klein, and Frank Hutter. BOHB: Robust and Efficient Hyperparameter Optimization at Scale. In *International Conference on Machine Learning*, pages 1437–1446. PMLR, July 2018.

[23] M. Feurer, T. Springenberg, and F. Hutter. Initializing Bayesian Hyperparameter Optimization via Meta-learning. In B. Bonet and S. Koenig, editors, *Proceedings of the Twenty-Ninth National Conference on Artificial Intelligence (AAAI15)*, volume 15, pages 1128–1135. AAAI Press, 2015.

[24] Matthias Feurer, Benjamin Letham, Frank Hutter, and Eytan Bakshy. Practical Transfer Learning for Bayesian Optimization. *arXiv:1802.02219 [cs, stat]*, 2021.

[25] Pieter Gijsbers, Florian Pfisterer, Jan N. van Rijn, Bernd Bischl, and Joaquin Vanschoren. Meta-Learning for Symbolic Hyperparameter Defaults. *arXiv:2106.05767 [cs, stat]*, 2021.

[26] Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and D. Sculley. Google Vizier: A service for black-box optimization. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, page 14871495, 2017.

[27] Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. Revisiting deep learning models for tabular data. *Advances in Neural Information Processing Systems*, 34, 2021.

[28] J. Guerrero-Viu, S. Hauns, S. Izquierdo, G. Miotto, S. Schrodi, A. Biedenkapp, T. Elsken, D. Deng, M Lindauer, and F. Hutter. Bag of Baselines for Multi-objective Joint Neural Architecture Search and Hyperparameter Optimization. arXiv:2105.01015 [cs.LG], 2021.

[29] Cheng Guo and Felix Berkhahn. Entity Embeddings of Categorical Variables. *arXiv:1604.06737 [cs]*, 2016.

[30] Nikolaus Hansen, Anne Auger, Raymond Ros, Olaf Mersmann, Tea Tušar, and Dimo Brockhoff. Coco: A platform for comparing continuous optimizers in a black-box setting. *Optimization Methods and Software*, 36(1):114–144, 2021.

[31] Daniel Horn and Bernd Bischl. Multi-objective Parameter Configuration of Machine Learning Algorithms using Model-based Optimization. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8, 2016.

[32] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential Model-based Optimization for General Algorithm Configuration. In Carlos A. Coello Coello, editor, *Learning and Intelligent Optimization*, Lecture Notes in Computer Science, pages 507–523, Berlin, Heidelberg, 2011. Springer.

[33] Kevin Jamieson and Ameet Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2015.

[34] S. Jeong and S. Obayashi. Efficient global optimization (EGO) for multi-objective problem and data mining. In *2005 IEEE Congress on Evolutionary Computation*, volume 3, pages 2138–2145, 2005.

[35] D. R. Jones, M. Schonlau, and W. J. Welch. Efficient Global Optimization of Expensive Black-Box Functions. *Journal of Global Optimization*, 13(4):455–492, 1998.

[36] Kirthevasan Kandasamy, Gautam Dasarathy, Jeff Schneider, and Barnabas Poczos. Multi-fidelity Bayesian Optimisation with Continuous Approximations. *arXiv:1703.06240 [stat]*, 2017.

[37] John G. Kemeny and James Laurie Snell. *Mathematical Models in the Social Sciences.* MIT Press, Cambridge, MA, USA, 1972.

[38] A. Klein, S. Falkner, S. Bartels, P. Hennig, and F. Hutter. Fast Bayesian Optimization of Machine Learning Hyperparameters on Large Datasets. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 54 of *Proceedings of Machine Learning Research*, pages 528–536. PMLR, 2017.

[39] Aaron Klein, Zhenwen Dai, Frank Hutter, Neil Lawrence, and Javier Gonzalez. Meta-surrogate benchmarking for hyperparameter optimization. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[40] Aaron Klein, Louis C. Tiao, Thibaut Lienart, Cedric Archambeau, and Matthias Seeger. Model-based Asynchronous Hyperparameter and Neural Architecture Search. *arXiv:2003.10865 [cs, stat]*, 2020.

[41] J. Knowles. ParEGO: A hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. *IEEE Transactions on Evolutionary Computation*, 10(1):50–66, 2006.

[42] Michel Lang, Martin Binder, Jakob Richter, Patrick Schratz, Florian Pfisterer, Stefan Coors, Quay Au, Giuseppe Casalicchio, Lars Kotthoff, and Bernd Bischl. mlr3: A modern object-oriented machine learning framework in R. *Journal of Open Source Software*, 4(44):1903, 2019.

[43] Michel Lang, Bernd Bischl, and Dirk Surmann. batchtools: Tools for R to work on batch systems. *The Journal of Open Source Software*, 2017.

[44] N. Lavesson and P. Davidsson. Quantifying the impact of learning algorithm parameter tuning. In *Proc. of AAAI*, volume 6, pages 395–400, 2006.

[45] Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Jonathan Bentzur, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. A System for Massively Parallel Hyperparameter Tuning. In I. Dhillon, D. Papailiopoulos, and V. Sze, editors, *Proceedings of Machine Learning and Systems*, volume 2, pages 230–246, 2020.

[46] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. *Journal of Machine Learning Research*, 18(185):1–52, 2018.

[47] Rui Li, Michael T.M. Emmerich, Jeroen Eggermont, Thomas Bäck, M. Schütz, J. Dijkstra, and J.H.C. Reiber. Mixed integer evolution strategies for parameter optimization. *Evolutionary Computation*, 21(1):2964, 2013.

[48] M. Lindauer, K. Eggensperger, M. Feurer, A. Biedenkapp, J. Marben, P. Müller, and F. Hutter. BOAH: A Tool Suite for Multi-fidelity Bayesian Optimization & Analysis of Hyperparameters. arXiv:1908.06756 [cs.LG], 2019.

[49] Marius Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Difan Deng, Carolin Benjamins, Tim Ruhopf, René Sass, and Frank Hutter. SMAC3: A versatile Bayesian optimization package for hyperparameter optimization. *arXiv:2109.09831 [cs, stat]*, 2022. arXiv: 2109.09831.

[50] Marius Lindauer, Matthias Feurer, Katharina Eggensperger, André Biedenkapp, and Frank Hutter. Towards assessing the impact of bayesian optimization's own hyperparameters. *arXiv preprint arXiv:1908.06674*, 2019.

[51] H. Liu, K. Simonyan, and Y. Yang. DARTS: Differentiable Architecture Search. In *Proceedings of the International Conference on Learning Representations*, 2019.

[52] Olaf Mersmann, Heike Trautmann, Boris Naujoks, and Claus Weihs. Benchmarking Evolutionary Multiobjective Optimization Algorithms. In *IEEE Congress on Evolutionary Computation*, pages 1–8, July 2010.

[53] Christoph Molnar, Giuseppe Casalicchio, and Bernd Bischl. Quantifying model complexity via functional decomposition for better post-hoc interpretability. In Peggy Cellier and Kurt Driessens, editors, *Machine Learning and Knowledge Discovery in Databases*, page 193204. Springer International Publishing, 2020.

[54] Julia Moosbauer, Martin Binder, Lennart Schneider, Florian Pfisterer, Marc Becker, Michel Lang, Lars Kotthoff, and Bernd Bischl. Automated benchmark-driven design and explanation of hyperparameter optimizers, 2021.

[55] J. A. Nelder and R. Mead. A Simplex Method for Function Minimization. *The Computer Journal*, 7(4):308–313, 1965.

[56] Christina Nießl, Moritz Herrmann, Chiara Wiedemann, Giuseppe Casalicchio, and Anne-Laure Boulesteix. Over-optimism in benchmark studies and the multiplicity of design and analysis options when interpreting their results. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, page e1441, 2021.

[57] Takeru Ohta and Hiroyuki Vincent Yamazaki. Kurobako. `https://github.com/optuna/kurobako`, 2022.

[58] Maryam Parsa, John P. Mitchell, Catherine D. Schuman, Robert M. Patton, Thomas E. Potok, and Kaushik Roy. Bayesian Multi-objective Hyperparameter Optimization for Accurate, Fast, and Efficient Neural Network Accelerator Design. *Frontiers in Neuroscience*, 14:667, 2020.

[59] Valerio Perrone, Rodolphe Jenatton, Matthias W Seeger, and Cedric Archambeau. Scalable Hyperparameter Transfer Learning. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[60] Florian Pfisterer, Jan N. van Rijn, Philipp Probst, Andreas C. Müller, and Bernd Bischl. Learning Multiple Defaults for Machine Learning Algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '21, pages 241–242, New York, NY, USA, 2021. Association for Computing Machinery.

[61] Wolfgang Ponweiser, Tobias Wagner, Dirk Biermann, and Markus Vincze. Multiobjective optimization on a limited budget of evaluations using model-assisted $\mathcal{S}$-metric selection. In Günter Rudolph, Thomas Jansen, Nicola Beume, Simon Lucas, and Carlo Poloni, editors, *Parallel Problem Solving from Nature PPSN X*, Lecture Notes in Computer Science, pages 784–794. Springer, 2008.

[62] R. Schmucker, M. Donini, V. Perrone, M. B. Zafar, and C. Archambeaut. Multi-objective Multi-fidelity Hyperparameter Optimization with Application to Fairness. In *NeurIPS Workshop on Meta-Learning*, volume 2, 2020.

[63] Roy Schwartz, Jesse Dodge, Noah A Smith, and Oren Etzioni. Green ai. *Communications of the ACM*, 63(12):54–63, 2020.

[64] Kenan Šehić, Alexandre Gramfort, Joseph Salmon, and Luigi Nardi. Lassobench: A high-dimensional hyperparameter optimization benchmark suite for lasso. *arXiv preprint arXiv:2111.02790*, 2021.

[65] J. Siems, L. Zimmer, A. Zela, J. Lukasik, M. Keuper, and F. Hutter. NAS-Bench-301 and the Case for Surrogate Benchmarks for Neural Architecture Search. arXiv:2008.09777 [cs.LG], 2020.

[66] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical Bayesian Optimization of Machine Learning Algorithms. In *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[67] Kevin Swersky, Jasper Snoek, and Ryan Prescott Adams. Freeze-Thaw Bayesian Optimization. *arXiv:1406.3896 [cs, stat]*, 2014.

[68] Chris Thornton, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 847–855, 2013.

[69] Ryan Turner. Uber. bayesopt benchmark. `https://github.com/uber/bayesmark`, 2022.

[70] Joaquin Vanschoren. Meta-Learning. In Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren, editors, *Automated Machine Learning: Methods, Systems, Challenges*, The Springer Series on Challenges in Machine Learning, pages 35–61. Springer International Publishing, Cham, 2019.

[71] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luís Torgo. OpenML: Networked Science in Machine Learning. *SIGKDD Explor.*, 15(2):49–60, 2013.

[72] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. OpenML: Networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 15(2):49–60, 2014.

[73] Michael Volpp, Lukas P Fröhlich, Kirsten Fischer, Andreas Doerr, Stefan Falkner, Frank Hutter, and Christian Daniel. Meta-learning Acquisition Functions for Transfer Learning in Bayesian Optimization. *International Conference on Learning Representations*, 2020.

[74] C. White, W. Neiswanger, and Y. Savani. BANANAS: Bayesian Optimization with Neural Architectures for Neural Architecture Search. arXiv:1910.11858 [cs.LG], 2019.

[75] Martin Wistuba, N. Schilling, and L. Schmidt-Thieme. Learning Hyperparameter Optimization Initializations. *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 1–10, 2015.

[76] Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. Two-Stage Transfer Surrogate Model for Automatic Hyperparameter Optimization. In *European Conference on Machine Learning and Knowledge Discovery in Databases - Volume 9851*, ECML PKDD 2016, pages 199–214, Berlin, Heidelberg, 2016. Springer-Verlag.

[77] C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter. NAS-Bench-101: Towards Reproducible Neural Architecture Search. In *Proceedings of the 36th International Conference on Machine Learning*, pages 7105–7114, 2019.

[78] Lucas Zimmer. data_2k_lw.zip. figshare. Dataset. `https://doi.org/10.6084/m9.figshare.11662422.v1`, Apache License, Version 2.0, 2020.

[79] Lucas Zimmer. nasbench301_full_data. figshare. Dataset. `https://doi.org/10.6084/m9.figshare.13286105.v1`, Apache License, Version 2.0, 2020.

[80] Lucas Zimmer, Marius Lindauer, and Frank Hutter. Auto-pytorch tabular: Multi-fidelity metalearning for efficient and robust autodl. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(9):3079 – 3090, 2021.

[81] E. Zitzler, L. Thiele, M. Laumanns, C.M. Fonseca, and V.G. da Fonseca. Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Transactions on Evolutionary Computation*, 7(2):117132, 2003.

A Maintenance of YAHPO Gym

Following [20], we present a maintenance plan for YAHPO Gym.

- Who is maintaining the benchmarking library?
  YAHPO Gym is developed and maintained by the *Statistical Learning and Data Science Group at the Ludwig-Maximilians University Munich.*

- How can the maintainer of the dataset be contacted (e.g., email address)?
  Questions should be submitted via an issue on the Github repository at `https://github.com/slds-lmu/yahpo_gym`.

- Is there an erratum?
  No

- Will the library be updated?
  We plan on adding new instances as well as continuously updating in existing instances should need occur. Changes will be communicated via Github releases as well as a *CHANGELOG*.

- Will older versions of the benchmarking library continue to be supported/hosted/maintained?
  Old versions are available via GitHub releases in the git repositories. We aim to support old versions on a best-effort basis with limited support for older versions.

- If others want to extend/augment/build on/contribute to the dataset, is there a mechanism for them to do so?
  We have detailed how additional benchmarks can be added in the documentation `https://slds-lmu.github.io/yahpo_gym/extending.html`. We have furthermore made available the full code used to tune, fit and export surrogate models used in YAHPO Gym. The code is easily extendable for future datasets.

- Which dependencies does YAHPO Gym have?
  YAHPO Gym currently relies on the following dependencies
  (versions used throughout experiments in brackets):

  - `onnxruntime (1.10.0)`
  - `pyyaml (5.4.1)`
  - `configspace (0.4.20)`
  - `pandas (1.3.5)`

B Benchmark Suites

B.1 Criteria for Benchmark Suites and Instances

To allow for a more systematic assessment of the quality of benchmarking instances, we define criteria that guided the development of YAHPO Gym and which should be satisfied to make a compelling argument for the use of any HPO benchmark.

I. **Representativity & Diversity of Tasks** The goal of benchmark suites is to allow for a ranking of HPO methods according to their performance on future problems. Instances should therefore cover response surfaces encountered in relevant problem domains.

II. **Difficulty and Structure** Benchmarks must be non-trivial, i.e., they should contain instances of sufficient difficulty to identify rankings between optimizers. Search spaces should reflect search spaces that are encountered frequently in practice including mixed spaces with interactions as well as hierarchical spaces and sufficient dimensionality.

III. **Faithfulness** Rankings based on approximations (e.g., for *tabular* and *surrogate* instances) should reflect true rankings. The performance of surrogate models $\hat{g}$ should be close enough to g based on performance metrics such as Spearman's ρ.

IV. **Efficiency** Benchmark experiments often require repeated evaluation of several optimizers across several datasets leading to considerable computational (and consequentially environmental cost [63]). Benchmarks should therefore strive for computational efficiency.

V. **Ease of use** Benchmark software needs to be accessible and portable across operating systems and programming languages. In practice, systems that do not require complex set up or require establishment of databases might lead to more widespread adoption. Meta-data such as search spaces should be available and machine-readable. As benchmarks allow for embarrassingly parallel execution, parallelization should be supported.

VI. **Reproducibility** While performance estimation in practice often includes stochastic components, it is important that benchmark suites can be made reproducible through the use of random seeds. Furthermore, software dependencies and versions should be clearly communicated and design components should be fixed and versioned to avoid cherry picking.

VII. **Stochasticity** Performance estimates obtained in real instances are realizations of random variables. In order to reflect this in practice, instances should allow for repeated evaluations.

While we consider the above requirements for good benchmarking suites, we furthermore want to highlight other properties that might be relevant for benchmarking suites.

A. **Multi-fidelity** Multi-fidelity methods have been shown to considerably speed up evaluation. Benchmark instances should therefore allow for querying performances at multiple fidelities.

B. **Runtime** In practice, HPO evaluations, especially for complex AutoML scenarios, can have very heterogeneous runtimes [66], which should also be reflected in a realistic benchmark by providing access to (estimated) runtimes which could subsequently be used to more accurately benchmark cost-efficient optimization methods.

C. **Asynchronous Evaluation** Although technically non-trivial, benchmarks should ideally allow the comparison of parallel HPO methods, allowing to compare, e.g., asynchronous HPO procedures [45, 40].

D. **Multi-Objective** In many scenarios, users are not only interested in maximizing a single performance metric such as accuracy, but instead multiple relevant metrics such as calibration, inference time, memory usage, and many others. We therefore consider including multi-objective HPO problems an important characteristic of a benchmark suite.

E. **Meta-Learning** Last but not least, in many cases, data collections are used to test scenarios for *meta-learning* [70, 60, 25] or *transfer learning* [76, 59]. For these scenarios, the availability of data across a large amount of datasets is often useful.

B.2 Comparison to other Benchmark Suites

While a variety of benchmarking suites for optimization such as COCO [30], HPOLib [17], ASLib [12] and others exist, we do not go into detail and instead refer the reader to [20] where those libraries are discussed in more detail. We instead compare YAHPO Gym to the most similar suites: HPOBench [20] and HPO-B [3] and discuss and justify assessments made in Table 1.
Evaluations in Table 1 follow the doctrine *"the documentation is the product"* and we therefore consider only features that are explicitly documented in the accompanying manuscript and doc-

umentation, not considering other features. We note that all three libraries could theoretically be used or extended for additional tasks such as multi-objective evaluations but instead focus on scenarios where the considered property is explicitly included in the documented API. We furthermore note that several important aspects such as ease of use are not easily quantifiable and assessments made are therefore subjective. We derive assessments made in this section based on the criteria defined in Supplement B.1.

I. **Representativity** YAHPO Gym contains 14 across diverse search spaces for widely used ML algorithms trained on representative datasets. Search spaces are often mixed and sometimes include dependent hyperparameters resulting in a hierarchical search space. While theoretically possible, none of the instances in HPOBench currently contain hierarchical search spaces. HPO-B only supports continuous search spaces.

II. **Difficulty** To the best of our knowledge, it is not yet clear how to assess the difficulty of a benchmark instance. We therefore instead focus on showing that benchmark instances in YAHPO Gym are not trivial, e.g., constant across the full search space.

III. **Faithfulness** We evaluate the quality of fitted surrogates in Supplement D.2. To the best of our knowledge, analyses that establish the faithfulness of tabular benchmarks have not been conducted for tabular benchmarks previously.

IV. **Efficiency** We consider efficiency with respect to two aspects: *computational cost* and *memory consumption.* Tabular benchmarks often keep the full data in memory, essentially limiting the amount of parallel optimization runs on a given hardware required, e.g., for replications of stochastic benchmark experiments. Furthermore, surrogate benchmarks are often based on un-optimized models fitted for each single instance. As a result, the required metadata (and memory consumption when multiple models are kept in memory) is often comparatively large. Our surrogates in contrast are highly optimized, compressed neural networks fitted across an entire scenario. Our surrogates are furthermore portable across platforms, alleviating concerns regarding software dependencies. Prediction on a surrogate requires only 10-100 ms and around 100 MB of memory allowing for a high degree of parallelization. In a small experiment, we estimate runtime and memory overhead for 300 iterations of random search on comparable SVM search spaces in Table 1 using the Python memory profiler (`https://pypi.org/project/memory-profiler/`). Since memory profiling is not accurate for HPO-Bench due to external processes, we estimate memory consumption using `htop`. Differences partially stem from more expensive setup in other libraries, but we consider 300 iterations of random search a representative use-case for many scenarios. Benchmarks were conducted on an AMD Ryzen 5 3600 6-Core CPU.

V. **Ease of use** YAHPO Gym does not require setting up containerization or any database and has only 4 dependencies that are both widely used and mature. All metadata required can be downloaded from a single, versioned metadata repository [4]. The modules API is simple to use (see, e.g., Section 4). Other benchmarking suites either require *benchmark instance specific* software dependencies that can differ from benchmark instance to instance. While HPOBench has solved this using *containerization* adding considerable computational overhead, our surrogates only rely on a single fixed version of `ONNX` and can therefore completely ignore the problem.

VI. **Reproducibility** surrogates used in the benchmarking suites proposed along with YAHPO Gym are deterministic. Reproducibility therefore only requires ensuring seeding of any stochastic procedures in the optimization algorithm. Furthermore, we fix several design choices that might lead to differences between benchmarks: *i*) search spaces $\tilde{\Lambda}$ are fixed

[4] `https://github.com/slds-lmu/yahpo_data`

for each scenario and should be used in benchmarks *ii*) target metrics and exact evaluation protocol are fixed within the benchmark suites (see Supplement C.2) to ensure comparability.

Additional properties *A.* − *E.* described in Supplement B.1 are compared in Table 1 and described in more detail below.

A. **Multi-fidelity** Only surrogate based benchmarks allow doing so for the full range of available fidelity steps. This essentially enforces evaluation at fixed fidelities in tabular benchmarks, e.g., disallowing evaluation of differing fidelity schedules. In contrast, surrogates in YAHPO Gym allow for evaluation at all fidelity steps.

B. **Runtime** All surrogates in YAHPO Gym allow for querying the predicted runtime for training a configuration, essentially allowing benchmarking methods that take into account runtimes.

C. **Asynchronous Evaluation** To our knowledge, none of the existing benchmark suites allow for asynchronous evaluation (except for *real* instances in *HPO-Bench*). YAHPO Gym currently allows for asynchronous evaluation, but this is considered an experimental feature. We hope to be able to fully allow asynchronous benchmarking in future versions of our benchmark.

D. **Multi-Objective** YAHPO Gym explicitly includes multiple objective for each scenario and allows the user to subset the returned targets explicitly. In contrast, HPO-Bench contains only few multi-objective benchmarks and does not explicitly document how they are supposed to be used.

E. **Transfer learning** All considered suites allow for transfer learning. In contrast to *HPO-Bench* and *HPO-B*, YAHPO Gym includes the (to our knowledge) largest collection of instances for a given scenario for the *rbv2_** scenarios consisting of up to 119 instances. Only few collections in HPOBench contain enough instances for meta-learning.

We furthermore define a *single objective* as well as a *multi-objective* benchmark task that include a evaluation protocol with respect to instances, search spaces, evaluation budget and target metrics. This allows for reproduction and extension by practicioners without additional design choices and provides a singular point of references.

B.3 A Benchmark Instance

In order to improve differentiation, we formally define four different types of benchmark instances derived from 1. We therefore only consider benchmarks based on *tabular, surrogate* and *real* instances in our manuscript.

Definition 3 (Synthetic Benchmark Instance) *A synthetic benchmark instance is a benchmark instance, where* $g : \boldsymbol{\lambda} \rightarrow \mathbb{R}^m$ *is a mathematically tractable function.*

Synthetic instances, such as the ones, e.g., included in *COCO* [30] rely on mathematically tractable test functions (e.g., Rosenbrock-2D) as response surface. While they provide cheap evaluations, problem structures in such functions are qualitatively distinct from test functions encountered in HPO scenarios, and the resulting optimization problem is therefore often not representative for optimization problems typically encountered in HPO.

Definition 4 (Tabular Benchmark Instance) *A tabular benchmark instance returns function evaluations* $g(\boldsymbol{\lambda})$ *from a table of pre-recorded performance results. Performance results are typically obtained by estimating* $\widehat{\mathrm{GE}}(\mathcal{I}, \mathcal{J}, \rho, \boldsymbol{\lambda})$ *for given* $\mathcal{I}$, $\mathcal{J}$ *and* ρ*. In contrast to synthetic and surrogate instances, the search space* Λ *is discretized and* g *can therefore be only evaluated at discrete points* $\tilde{\Lambda} \in \Lambda$.

Definition 5 (Surrogate Benchmark Instance) *A surrogate benchmark returns predictions $\hat{g}(\boldsymbol{\lambda})$ of machine learning models trained to infer the functional relationship between $\boldsymbol{\lambda}$ and function evaluations $g(\boldsymbol{\lambda})$ based on a set of pre-recorded performance results.*

For clarity, we would like to differentiate in terminology between the *instance surrogate* of a surrogate benchmark, and the algorithm surrogate potentially used by an HPO method, e.g., the Gaussian process as surrogate model in BO explicitly mentioning the algorithm surrogate where required. The instance surrogate model $\hat{g}$ or the tabular data should approximate the true relationship between $\boldsymbol{\lambda}$ and the target metrics reasonably well. We consider a mapping $\hat{g}$ to be *faithful* if:

1. cross-validated performance metrics are sufficiently good with respect to metrics such as R^2 and Spearman's ρ. We typically consider a cutoff $\rho > 0.7$ for including a surrogate.

2. if the induced ranking of optimizers on a given $\hat{g}$ closely resembles the true rankings on the original underlying optimization problem (in general, the *real* setting relying on g).

3. learning curves of HPO methods on $\hat{g}$ closely resemble the true performance curves.

Definition 6 (Real Benchmark Instance) *A real benchmark instance returns function evaluations $g(\boldsymbol{\lambda})$. Performance results are typically obtained by estimating $\widehat{\mathrm{GE}}(\mathcal{I}, \mathcal{J}, \rho, \boldsymbol{\lambda})$ for given $\mathcal{I}$, $\mathcal{J}$ and ρ.*

Since the same benchmark instance can be provided as a *real*, *tabular*, or *surrogate* instance, we speak of different *versions* of that instance where required.

C YAHPO Gym

In the following we will provide additional details on general aspects of YAHPO Gym. A detailed description of included surrogates can be found in Supplement D and a detailed description of used data and included search spaces can be found in Supplement F.

C.1 Usage

The `yahpo_gym` software can be directly installed from *GitHub*[5] and only requires downloading one additional GitHub repository containing metadata[6] in an initial setup step.

HPO Benchmarking

To ensure interoparability with different optimizer API's, YAHPO Gym offers only evaluation of the objective function using the `BenchmarkSet.objective_function(xs)` method (where `xs` is a hyperparameter configuration to be evaluated). This allows for use with many different optimizers (see, e.g., examples provided in the acompanying notebooks). We furthermore allow for querying the search space using `BenchmarkSet.get_opt_space(xs)` in order to ensure that optimizers are ran on comparable search spaces. We provide additional details with respect to exact setups

Transfer HPO

Different forms of Transfer HPO are available in YAHPO Gym and can be setup analogous by querying the objective function across different instances of the scenario. We present examples in the modules documentation.

[5]`https://github.com/slds-lmu/yahpo_gym`
[6]`https://github.com/slds-lmu/yahpo_data`

C.2 Benchmark Suites: YAHPO-SO & YAHPO-MO

This section provides additional details with respect to the two benchmark sets proposed with YAHPO Gym. Both suites can be obtained via `get_suites(<type>, <version>)` specifying the type of the benchmark (currently supporting 'single' for YAHPO-SO and 'multi' for YAHPO-MO) and the version (currently 1.0).

- Optimizers should use the search spaces included in YAHPO Gym in order to establish that differences in performance do not depend on differing search spaces.
- Optimization should be run for $\lceil 20 + 40 \cdot \sqrt{\text{SEARCH\_SPACE\_DIM}}\, \rceil$ steps. Each step is equivalent to a full budget evaluation, essentially allowing multi-fidelity method the same number of full budget equivalents. We report the budgets for each scenario in Table 3 and Table 4.
- Target metrics to be used with the single-objective and multi-objective suite are reported in Table 3 and Table 4.
- We encourage reporting *mean normalized regret* and *mean ranks* for the anytime performance of an optimizer. Reported values are based on the target metric for YAHPO-SO and the normalized Hypervolume Indicator for YAHPO-MO.
- In order to assess variance, we encourage reporting averages and standard errors across 30 replications with differing random seeds.

We will now go on to discuss criteria for inclusion of tasks in the respective benchmarks.

In light of the criteria defined in B.1, we strive for diversity by including instances from all included scenarios. We consider only surrogates that are *faithful* (measured via Spearman's ρ reported for each target below). Our benchmarks are made available through a fully documented API. Inference on a surrogate model is highly efficient taking usually only 10-100 milliseconds per batch. Benchmarks are furthermore reproducible and allow for parallelization and runtime prediction on a continuous range of fidelities. We include search spaces for all problems in Supplement F.
We furthermore briefly want to discuss selecting a budget that depends on the scenario at hand. We consider the search space dimension to be a relevant input for determining the overall optimization budget that should be used for optimization. Our formula ensures, that optimization runs for a minimum of 77 iterations (iaml_glmnet, 2D) and a maximum of 267 (rbv2_super, 38D) iterations, which we consider useful bounds for the respective search space dimensionality, especially given that multi-fidelity allows for evaluations at a fraction of the full budget.

C.3 R package

While we focus on the python module in the manuscript, YAHPO Gym offers a R interface that is equivalent in functionality. We do not present the API in detail here since it follows the same principles and naming conventions as the python module. Further information is available from the package documentation. Listing 1 contains the sample R-code used to first draw a random configuration from the search space and then evaluate the drawn configuration.

D YAHPO Gym Surrogates

On an implementation level, YAHPO Gym consists of a (versioned) Python module / R-package `yahpo_gym` and a (versioned) set of required metadata (including fitted surrogate models) which we will call `yahpo_data` in the following. The core contribution in YAHPO Gym is a set of surrogate models[7] based on neural networks. This section provides additional details with respect to the fitting procedures of surrogate models as well as a rigorous evaluation of the final surrogates.

[7] available at `https://github.com/slds-lmu/yahpo_data`

Table 3: **YAHPO-SO** (v1): Collection of single-objective benchmark instances. We indicate surrogate approximation quality using Spearman's ρ.

	scenario	instance	target(s)	ρ	budget
1	lcbench	167168	val_accuracy	0.94	126
2	lcbench	189873	val_accuracy	0.97	126
3	lcbench	189906	val_accuracy	0.97	126
4	nb301	CIFAR10	val_accuracy	0.98	250
5	rbv2_glmnet	375	acc	0.80	90
6	rbv2_glmnet	458	acc	0.85	90
7	rbv2_ranger	16	acc	0.93	134
8	rbv2_ranger	42	acc	0.98	134
9	rbv2_rpart	14	acc	0.92	110
10	rbv2_rpart	40499	acc	0.97	110
11	rbv2_super	1053	acc	0.31	267
12	rbv2_super	1457	acc	0.70	267
13	rbv2_super	1063	acc	0.57	267
14	rbv2_super	1479	acc	0.36	267
15	rbv2_super	15	acc	0.75	267
16	rbv2_super	1468	acc	0.77	267
17	rbv2_xgboost	12	acc	0.93	170
18	rbv2_xgboost	1501	acc	0.89	170
19	rbv2_xgboost	16	acc	0.91	170
20	rbv2_xgboost	40499	acc	0.96	170

```r
library("yahpogym")
library("paradox")
library("bbotk")
# Instantiate the BenchmarkSet
b = BenchmarkSet$new('lcbench', instance='3945')
# Get the objective
objective = b$get_objective('3945', check_values = FALSE)
# Sample a point from the ConfigSpace
xdt = generate_design_random(b$get_search_space(), 1)$data
xss_trafoed = transform_xdt_to_xss(xdt, b$get_search_space())
# Evaluate the configuration
objective$eval_many(xss_trafoed)
```

Listing 1: R-code to sample and evaluate a configuration using YAHPO Gym.

D.1 Setup and Training

Previous work [18, 19, 65] suggests that tree based regression methods such as random forests [13] are very suited as instance surrogate models for (single-objective) benchmarks. However, in YAHPO Gym we want to predict multiple target metrics for each instance of a benchmark collection efficiently and compactly. As a result, we use neural network surrogates because they 1) can naturally handle multiple outputs and do not require a model for each target metric and 2) should scale better than a random forest (fitted on each target metric) when the dimensionality of the data (especially in the number of features) increases.

Table 4: **YAHPO-MO** (v1): Collection of multi-objective benchmark instances. We indicate surrogate approximation quality using Spearman's ρ (averaged over targets).

	scenario	instance	target(s)	ρ	budget
1	iaml_glmnet	1489	mmce, nf	0.86	77
2	iaml_glmnet	1067	mmce, nf	0.73	77
3	iaml_ranger	1489	mmce, nf, ias	0.93	134
4	iaml_ranger	1067	mmce, nf, ias	0.92	134
5	iaml_super	1489	mmce, nf, ias	0.82	232
6	iaml_super	1067	mmce, nf, ias	0.82	232
7	iaml_xgboost	40981	mmce, nf, ias	0.88	165
8	iaml_xgboost	1489	mmce, nf, ias	0.92	165
9	iaml_xgboost	40981	mmce, nf, ias,rammodel	0.89	165
10	iaml_xgboost	1489	mmce, nf, ias,rammodel	0.92	165
11	lcbench	167152	val_accuracy, val_cross_entropy	0.98	126
12	lcbench	167185	val_accuracy, val_cross_entropy	0.91	126
13	lcbench	189873	val_accuracy, val_cross_entropy	0.93	126
14	rbv2_ranger	6	acc, memory	0.90	134
15	rbv2_ranger	40979	acc, memory	0.73	134
16	rbv2_ranger	1476	acc, memory	0.88	134
17	rbv2_rpart	41163	acc, memory	0.85	110
18	rbv2_rpart	1476	acc, memory	0.80	110
19	rbv2_rpart	40499	acc, memory	0.83	110
20	rbv2_super	1457	acc, memory	0.66	267
21	rbv2_super	6	acc, memory	0.68	267
22	rbv2_super	1053	acc, memory	0.45	267
23	rbv2_xgboost	1478	acc, memory	0.86	170
24	rbv2_xgboost	1476	acc, memory	0.83	170
25	rbv2_xgboost	32	acc, memory	0.82	170

Surrogate models used in YAHPO Gym are based on `ResNet` architectures for tabular data [27]. Instead of relying on a fixed architecture, we tune the neural network for each *Scenario* using `Optuna` [2]. We used the Adam optimizer for a maximum of 100 epochs (early stopping with patience of 10) with L2 loss. Surrogates were trained jointly for each benchmark scenario (for all instances and target metrics). We use a stratified train/validation/test split of 0.6/0.2/0.2, using the validation data to determine the surrogate model architecture and report performances on the test set. The search space as well as the fully reproducible code for fitting can be obtained at YAHPO Gym. Tuning and fitting of a single *Scenario* takes 3 GPU days on average on an NVIDIA DGX-A100 instance, we therefore estimate a one time cost of 45 GPU days for establishing the full benchmark.

We adapt the architecture proposed in [27] in multiple ways:

Feature- and output-scaling Hyperparameters as well as resulting performance metrics (e.g learning rates of log-loss values) often vary across orders of magnitudes. We have practically observed that transforming target metrics to the unit cube prior to training and reverse-transforming afterwards massively improves quality of the resulting surrogates. Available scaling techniques include *Neg-Exp* and *Log* transformation before scaling to $[0, 1]$. We furthermore include clamping to ensure that predictions are in valid ranges. Non-numeric features were transformed via entity embeddings [29].

Ensembles In order to allow for an estimate of variance, we make *noisy* versions of our surrogates available together with the standard *deterministic* set of surrogates. Ensembles consist of replications of the architecture determined during tuning and fitted on different permutations of the data with differing initial weights. The prediction step is the weighted average over predictions from ensemble members with weights α_i sampled from a Dirichlet distribution.

We furthermore consider scenarios that allow simulating **asynchronous evaluation** and therefore predict the time of the training procedure using our surrogates. YAHPO Gym currently supports asynchronous scheduling by estimating the runtime of training a model and then idling the system for the estimated time. This is implemented via `objective_function_timed` in `yahpo_gym` but currently considered in an experimental status.
In future work, we hope to propose and evaluate a surrogate-based benchmark explicitly allowing for benchmarking of asynchronous scheduling strategies based on surrogate predictions. To enable more realistic scheduling, we hope to furthermore include memory constraints using predicted peak memory consumption for a training run.

D.2 Surrogate Quality

We provide an overview over surrogate quality measured on the test set using Spearman's ρ averaged across all instances in Table 5. Metrics are routinely ≥ 0.9 except for few instances / target metrics and even surpasses performances for surrogate models reported, e.g., in [65]. We furthermore depict real and predicted learning curves for 4 randomly drawn configurations in Figure 5. Note that in our work, learning curves are predicted only based on hyperparameters, and not based on initial, low-fidelity observations (as done in learning curve prediction tasks). Our surrogates therefore solve a much harder task. Surrogates in general predict the learning curves with a high degree of precision.

Table 5: Average surrogate performance (Spearman's rho) across all instances per scenario/target. We abbreviate cross_entropy (ce) and balanced_accuracy(bac) for brevity.

Scenario	ρ
iaml_glmnet	mmce:0.97,f1:0.9,auc:0.92,logloss:0.97,rammodel:0.97,timetrain:0.95,mec:0.9,ias:0.91,nf:0.97
iaml_ranger	mmce:0.99,f1:0.98,auc:1,logloss:0.95,rammodel:1,timetrain:0.91,mec:0.88,ias:0.98,nf:1
iaml_rpart	mmce:0.99,f1:0.96,auc:0.99,logloss:0.96,rammodel:1,timetrain:0.96,mec:0.71,ias:0.96,nf:0.96
iaml_super	mmce:0.93,f1:0.95,auc:0.89,logloss:0.93,rammodel:0.71,timetrain:0.61,mec:0.94,ias:0.65,nf:0.92
iaml_xgboost	mmce:0.97,f1:0.98,auc:0.97,logloss:0.93,rammodel:0.86,timetrain:0.71,mec:0.95,ias:0.84,nf:0.99
lcbench	time:0.94,val_accuracy:0.95,val_ce:0.97,val_bac:0.98,test_ce:0.99,test_bac:0.98
nb301	val_accuracy:0.98,runtime:0.94
rbv2_aknn	acc:0.99,bac:0.99,auc:0.98,brier:1,f1:0.91,logloss:0.99,timetrain:0.64,memory:0.83
rbv2_glmnet	acc:0.99,bac:0.95,auc:0.91,brier:1,f1:0.96,logloss:0.99,timetrain:0.79,memory:0.82
rbv2_ranger	acc:0.99,bac:0.98,auc:0.95,brier:1,f1:0.92,logloss:1,timetrain:0.84,memory:0.66
rbv2_rpart	acc:0.98,bac:0.96,auc:0.93,brier:0.99,f1:0.93,logloss:0.98,timetrain:0.72,memory:0.86
rbv2_super	acc:0.82,bac:0.78,auc:0.73,brier:0.91,f1:0.91,logloss:0.89,timetrain:0.69,memory:0.71
rbv2_svm	acc:0.99,bac:0.98,auc:0.94,brier:0.99,f1:0.91,logloss:0.99,timetrain:0.76,memory:0.84
rbv2_xgboost	acc:0.98,bac:0.96,auc:0.94,brier:0.99,f1:0.92,logloss:0.98,timetrain:0.93,memory:0.78

Some of the targets available require further study and we therefore discourage their use in benchmarks. Those are *rampredict & ramtrain* (iaml_* scenarios) as well as *timepredict* (rbv2_* scenarios). Reasons for this assessment are partially poor surrogates, but we also assume that the underlying data is at fault: Prediction times are often very small and heavily influenced by system load, while correct estimation of required memory are relatively difficult to obtain in general.

D.3 Instance Difficulty

We quantify difficulty of instances using the The Empirical Cumulative Distribution Function (ECDF), assuming that difficult instances have only a small probability mass close to the optimum.

ECDFs for all instances in YAHPO-SO are shown in Figure 6. Differences between real evaluations and surrogate predictions can stem from the sampling procedure (random on surrogates vs. unknown sampling for real evaluations), as well as biases in the surrogates. All evaluations are made at maximal fidelity.
We furthermore provide cumulative ECDF plots for all optimizers in Figure 7. This allows for a different perspective on the quality of solutions found by the different optimizers.

E Experiments

E.1 Tabular vs. Surrogate Benchmarks

Resolution of tabular benchmarks. In practice, the resolution of grid points needs to be low for high dimensional spaces to limit the resulting table to a usable size. With purely categorical search spaces, often used in Neural Architecture Search, an exhaustive (i.e., $\tilde{\Lambda}_{\text{discrete}} = \tilde{\Lambda}$) tabular benchmark is often possible, as in, e.g., NAS-Bench-101 [77], which contains "only" 423k unique architectures. Multi-fidelity evaluations essentially add an additional dimension to the optimization problem when considering tabular data, since each evaluation now needs to be stored at multiple fidelity steps. If fidelity steps are not available at all budget levels, optimization benchmarks can be restricted to fixed fidelity progression (e.g., geometric progression as used in Hyperband).

Discrete Search Spaces. The modification of the search space from $\tilde{\Lambda}$ to $\tilde{\Lambda}_{\text{discrete}}$ can be handled in one of two ways: One can let HPO methods operate on the original search space $\tilde{\Lambda}$ and transparently "round" values to the nearest point contained in $\tilde{\Lambda}_{\text{discrete}}$. This effectively presents the optimization algorithm with a locally constant objective function. Alternatively, one can inform the HPO algorithm about the discrete nature of $\tilde{\Lambda}_{\text{discrete}}$, and possibly even modify the optimization procedure. As an example, consider the acquisition function optimization step within the BO framework: In the context of tabular benchmarks, the problem of optimizing the infill criterion becomes trivial because one can perform an exhaustive search over all points not yet evaluated to determine the next candidate(s) for evaluation. Note that we could also proceed to use a 1-Nearest-Neighbor model to evaluate HPCs in tabular benchmarks. This essentially results in a surrogate benchmark because we now rely on a performance model for the evaluation. In contrast to approximation by discretization, in a surrogate benchmark the domain of the objective function is not explicitly altered. Instead, predictions of an instance surrogate regression model $\hat{f}(\cdot)$ are returned as function evaluations, $\hat{c}_{\text{surrogate}} : \tilde{\Lambda} \rightarrow \mathbb{R}^m, \boldsymbol{\lambda} \mapsto \hat{f}(\boldsymbol{\lambda})$. The drawback here is that values returned by the surrogate model may misrepresent the local structure of the problem as well. Beyond the resolution of the surrogate model training data, these structures are interpolated and influenced by the inductive bias implied by the model.

Experimental Setup. As a *real* benchmark, we consider the original synthethic benchmark function, while we generate a grid containing at most 10^6 points for the tabular version, storing these pre-evaluated points in a look-up table together with their function value. The resolution of the grid is the same for all functions along the budget parameter dimension, with 10 grid points ranging from 2^{-9} to 1 on a 2^x scale. For all other parameters of the domain, an equidistant grid was generated by using $\lfloor (10^5)^{\frac{1}{D}} \rfloor$ grid points for each dimension $d = 1, \ldots, D$. With the same data we employ a similar surrogate neural network as used in YAHPO Gym. We compare the following methods on real, surrogate, and tabular benchmarks: All HPO methods were run for a total budget of 100 evaluations reflecting 100 full fidelity evaluations. The synthetic test functions used in the experiments [36] include a multi-fidelity parameter allowing for the use of multi-fidelity methods such as Hyperband. Of the methods investigated, only `HB` makes use of the fidelity parameter, while all other methods perform full budget evaluations. As a surrogate, we train a Wide & Deep Network [14]. More details can be found in `https://github.com/slds-lmu/yahpo_exps`. BO variants used Expected Improvement [35] as acquisition function and an initial design of $5 \cdot D$

Table 6: Consensus Rankings of HPO Methods for Real, Surrogate and Tabular Benchmarks.

Benchmark	Consensus Ranking (CR)	Permutation Order
Real	BO_GP_DF ≻ BO_GP_RS ≻ BO_RF_RS ≻ BO_NN_RS ≻ BO_NN_DF ≻ HB ≻ BO_RF_DF ≻ RS	-
Surrogate	BO_GP_DF ≻ BO_GP_RS ≻ BO_RF_RS ≻ BO_NN_RS ≻ HB ≻ BO_NN_DF ≻ BO_RF_DF ≻ RS	2
Tabular	BO_GP_DF ≻ BO_GP_RS ≻ BO_RF_DF ≻ HB ≻ BO_RF_RS ≻ BO_NN_DF ≻ BO_NN_RS ≻ RS	5

points sampled uniformly at random. The Gaussian process surrogate model used a Matérn 3/2 kernel. Nelder-Mead as acquisition function optimizer was terminated if the relative change in the maximum fell below $1e-4$. Tabular benchmarks used an exhaustive search for optimizing the acquisition function in the scenario of `*_DF`. Random search as acquisition function optimizer was allowed 10^4 evaluations.

Evaluation. For evaluation, we computed the mean normalized regret for each HPO method separately on the real, surrogate and tabular benchmarks (where the normalized regret for an HPO method given a cumulative budget is defined as the difference between the value of the best HPC found by any algorithm and the value of the best HPC found by this method, scaled by the range of objective function values as found by any method, see also [3]). Based on the normalized regret, we also computed the mean rank of each HPO method.
Results for the Branin2D, Currin2D and Hartmann3D benchmark functions are given in Figure 8. Differences between tabular and real/surrogate benchmarks can be explained by the fact that the inner optimization problem of BO methods is much easier to solve when only a finite set of potential candidates must be evaluated (i.e., by exhaustive search). We also observe that for the BO performance on the tabular benchmarks, there is no substantial difference in whether the acquisition function optimization is solved exactly or via a random search. We employ the rank-based symmetric difference (SD) method that aims to find a consensus ranking that minimizes the average number of rank reversals for the individual benchmark function rankings. We limit ourselves to the scenario of considering the set of all linear orders of HPO methods as candidates for a consensus ranking (SD/L).
By comparing the consensus ranking obtained via the surrogate/tabular benchmarks to the consensus ranking obtained using the real benchmarks, we determine the faithfulness of surrogate and tabular benchmarks.
We observe that the consensus ranking obtained using the surrogate benchmarks matches the real one more closely than rankings obtained using tabular benchmarks (Table 6).

E.2 Single-Objective Benchmark on YAHPO-SO

Instances and Evaluation Protocol. We use the set of instances and target variables defined for the YAHPO-SO benchmark suite defined in Supplement C.2 and detailed Table 3. We furthermore follow the described evaluation protocol, using available search spaces and optimization budgets including 30 replications to assess variance in results. As an evaluation criterion, we report mean normalized regret (based on the target metric), see Figure 9. Table 7 provides additional info on all optimizers used in the benchmark. Random Search simply samples configurations uniformly at random. SMAC is a model based full-fidelity optimizer using a random forest as surrogate model and Expected Improvement as acquisition function [35]. We use the SMAC4HPO facade [49]. Hyperband randomly samples new configurations and allocates more fidelity to promising configurations by relying on repeated successive halving (SH; [33]). BOHB combines BO with Hyperband and uses a Tree Parzen Estimator (TPE; [6]) as surrogate model. DEHB is a model-free successor of BOHB which relies on differential evolution instead of BO. We use the software defaults regarding the choice of mutation and crossover. SMAC-HB also combines BO with Hyperband but uses a random forest as surrogate model (SMAC4MF facade; [49]). Optuna uses a TPE as surrogate model and a median pruner [26] that follows a fixed SH schedule. A configuration is stopped by the pruner if its best intermediate result (at a given fidelity level determined by the SH schedule) is worse compared to the median of the other configurations on the same fidelity level.

Table 7: Optimizers used in the single-objective benchmark.

Optimizer	Software	Reference	Version
Random Search	-	-	-
SMAC (SMAC4HPO)	`https://github.com/automl/SMAC3`	[49]	1.1.1
Hyperband	`https://github.com/automl/HpBandSter`	[46]	0.7.4
BOHB	`https://github.com/automl/HpBandSter`	[22]	0.7.4
DEHB	`https://github.com/automl/DEHB`	[5]	67ac239
SMAC-HB (SMAC4MF)	`https://github.com/automl/SMAC3`	[49]	1.1.1
optuna	`https://optuna.org/`	[2]	2.10.0

E.3 Multi-Objective Benchmark on YAHPO-MO

Instances and Evaluation Protocol. We use the set of instances and target variables defined for the YAHPO-MO benchmark suite defined in Supplement C.2 and detailed in Table 4. We furthermore follow the described evaluation protocol, using available search spaces and optimization budgets including 30 replications to assess variance in results. As an evaluation criterion, we report the mean Hypervolume Indicator [81] computed on normalized targets (see Figure 10). Nadir points and reference Pareto fronts were obtained empirically over all replications of all HPO methods on a given benchmark instance. Table 8 provides additional info on all optimizers used in the benchmark. Random Search simply samples configurations uniformly at random. Random Search (x4) at each step samples four configurations uniformly at random (in parallel). ParEGO is a model based optimizer relying on a scalarization of the objectives which we then model using a random forest as surrogate model. As acquisition function we use Expected Improvement (EI; [35]). SMS-EGO is a model based optimizer that uses a surrogate model for each objective (again, we use random forests) and proposes candidates based on the $\mathcal{S}$-metric [61]. EHVI is a model based optimizer using a surrogate model for each objective (again, we use random forests) and proposes candidates based on their Expected Hypervolume Improvement [21]. MEGO is a model based optimizer using a surrogate model for each objective (again, we use random forests) and proposes candidates by considering the EI for each objective which gives rise to a multi-objective optimization problem of the acquisition functions themselves. For the final candidate selection, we sample uniformly at random over the Pareto optimal (with respect to the EIs) candidates. MIES is a mixed integer evolutionary optimizer (plus survival scheme, $\mu = \lfloor \text{budget}/6 \rfloor$, $\lambda = \lfloor \mu/4 \rfloor$[8]). We use Gaussian mutation ($p = 0.2$) for numerical parameters and discrete uniform mutation ($p = 0.2$) for categorical parameters. For recombination, we use uniform crossover ($p = 0.2$). As parent selection we perform a tournament selection of parents using nondominated sorting. For survival, we select the best individuals based on nondominated sorting.

F Scenarios, Search Spaces and Data Sources

Random Bot V2 (rbv2_)

All scenarios prefixed with *rbv2_* use data described in [10]. Data contains results from several ML algorithms trained across up to 117 datasets evaluated for a large amount of random evaluations. Table 9 lists all hyperparameters of the search space of the *rbv2_* scenarios. Targets are given by

[8]where budget is the optimization budget for a given instance, i.e., number of total evaluations

Table 8: Optimizers used in the multi-objective benchmark.

Optimizer	Software	Reference	Version
Random Search	-	-	-
Random Search (x4)	-	-	-
ParEGO	`https://github.com/mlr-org/mlr3mbo`	[41]	`1f59e13`
SMS-EGO	`https://github.com/mlr-org/mlr3mbo`	[61]	`1f59e13`
EHVI	`https://github.com/mlr-org/mlr3mbo`	[21]	`1f59e13`
MEGO	`https://github.com/mlr-org/mlr3mbo`	[34]	`1f59e13`
MIES	`https://github.com/mlr-org/miesmuschel`	[47]	`3483f11`

accuracy (`acc`), balanced accuracy (`bac`), AUC (`auc`), Brier Score (`brier`), F1 (`f1`), log loss (`logloss`), time for training the model (`timetrain`), and memory usage (`memory`).
Surrogates are fitted on subsets of the full data available from [10], such that a minimum of 1500 and a maximum of 200000 (depending on the scenario) evaluations are available for each instance in each scenario. All scenarios consist of a pre-processing step (missing data imputation) and a subsequently fitted ML algorithm. Instance ID's correspond to OpenML [71] dataset ids through which dataset properties can be queried[9]. OpenML tasks corresponding to each dataset can be obtained from [10]. We abbreviate the *num.impute.selected.cpo* hyperparameter with *imputation* throughout the tables. We fix the `repl` parameter to 10 for experiments.

NasBench-301 (nb301)

nb301 uses data of the NAS-Bench-301 benchmark ([79], see also [65]). Table 10 lists all hyperparameters of the search space of the *nb301* scenario. Targets are given by the validation accuracy (`val_accuracy`) and the training time (`runtime`).

LCBench (lcbench)

The *lcbench* collection uses data of the LCBench benchmark [78], as described in [80]. Table 11 lists all hyperparameters of the search space of the *lcbench* scenario. Targets are given by the validation accuracy (`val_accuracy`), validation cross entropy (`val_crossentropy`), validation balanced accuracy (`val_balanced_accuracy`), test cross entropy (`test_crossentropy`), test balanced accuracy (`test_balanced_accuracy`) and the training time (`time`).

Interpretable AutoML (iaml_)

All scenarios prefixed with *iaml_* rely on data that were newly collected by us. Different `mlr3` [42] learners ("classif.glmnet", "classif.rpart", "classif.ranger", "classif.xgboost") were incorporated into an ML pipeline with minimal preprocessing (removing constant features, fixing unseen factor levels during prediction and missing value imputation for factor variables by sampling from mon-missing training levels) via `mlr3pipelines` [9]. Hyperparameters of the learners were sampled uniformly at random (for the search spaces, see Table 12) and the ML pipeline performance (classification error - `mmce`, F1 score - `f1`, AUC - `auc`, logloss - `logloss`) was evaluated via 5-fold cross-validation on the following OpenML [72] classification tasks (data id): 40981, 41146, 1489, 1067. Each pipeline was then refitted and used for prediction on the whole data to estimate training and predict time (`timetrain`, `timepredict`) and RAM usage (during training and prediction, `ramtrain` and `rampredict` as well as model size, `rammodel`). Moreover, interpretability measures as described in [53] were computed for all models: number of features used (`nf`), interaction strength of features (`ias`) and main effect complexity of features (`mec`). To our best knowledge, this is the

[9]`https://www.openml.org/d/<dataset_id>`

Table 9: Search Spaces of YAHPO Gym's *rbv2_* scenarios. ⊦ indicates the parent in case dependencies between hyperparameters exist. The *super* scenario inherits dependencies from previous scenarios, while additional dependencies on the learner_id are introduced, indicated by a prefix.

rbv2_glmnet

Hyperparameter	Type	Range	Info
alpha	continuous	[0, 1]	
s	continuous	[0.001, 1097]	log
trainsize	continuous	[0.03, 1]	budget
imputation	categorical	impute.{mean, median, hist}	

rbv2_rpart

Hyperparameter	Type	Range	Info
cp	continuous	[0.001, 1]	log
maxdepth	integer	[1, 30]	
minbucket	integer	[1, 100]	
minsplit	integer	[1, 100]	
trainsize	continuous	[0.03, 1]	budget
imputation	categorical	impute.{mean, median, hist}	

rbv2_svm

Hyperparameter	Type	Range	Info
kernel	categorical	{linear, polynomial, radial}	
cost	continuous	[4.5e-05, 2.2e4]	log
gamma	continuous	[4.5e-05, 2.2e4]	log, ⊦ kernel
tolerance	continuous	[4.5e-05, 2]	log
degree	integer	[2, 5]	⊦ kernel
trainsize	continuous	[0.03, 1]	budget
imputation	categorical	impute.{mean, median, hist}	

rbv2_aknn

Hyperparameter	Type	Range	Info
k	integer	[1, 50]	
distance	categorical	{l2, cosine, ip}	
M	integer	[18, 50]	
ef	integer	[7, 403]	log
ef_construction	integer	[7, 403]	log
trainsize	continuous	[0.03, 1]	budget
imputation	categorical	impute.{mean, median, hist}	

rbv2_xgboost

Hyperparameter	Type	Range	Info
booster	categorical	{gblinear, gbtree, dart}	
nrounds	integer	[7, 2980]	log
eta	continuous	[0.001, 1]	log, ⊦ booster
gamma	continuous	[4.5e-05, 7.4]	log, ⊦ booster
lambda	continuous	[0.001, 1097]	log
alpha	continuous	[0.001, 1097]	log
subsample	continuous	[0.1, 1]	
max_depth	integer	[1, 15]	⊦ booster
min_child_weight	continuous	[2.72, 148.4]	log, ⊦ booster
colsample_bytree	continuous	[0.01, 1]	⊦ booster
colsample_bylevel	continuous	[0.01, 1]	⊦ booster
rate_drop	continuous	[0, 1]	⊦ booster
skip_drop	continuous	[0, 1]	⊦ booster
trainsize	continuous	[0.03, 1]	budget
imputation	categorical	impute.{mean, median, hist}	

rbv2_ranger

Hyperparameter	Type	Range	Info
num.trees	integer	[1, 2000]	
sample.fraction	continuous	[0.1, 1]	
mtry.power	integer	[0, 1]	
respect.unordered.factors	categorical	{ignore, order, partition}	
min.node.size	integer	[1, 100]	
splitrule	categorical	{gini, extratrees}	
num.random.splits	integer	[1, 100]	⊦ splitrule
trainsize	continuous	[0.03, 1]	budget
imputation	categorical	impute.{mean, median, hist}	

rbv2_super

Hyperparameter	Type	Range	Info
svm.kernel	categorical	{linear, polynomial, radial}	
svm.cost	continuous	[4.5e-05, 2.2e4]	log
svm.gamma	continuous	[4.5e-05, 2.2e4]	log
svm.tolerance	continuous	[4.5e-05, 2]	log
svm.degree	integer	[2, 5]	
glmnet.alpha	continuous	[0, 1]	
glmnet.s	continuous	[0.001, 1097]	log
rpart.cp	continuous	[0.001, 1]	log
rpart.maxdepth	integer	[1, 30]	
rpart.minbucket	integer	[1, 100]	
rpart.minsplit	integer	[1, 100]	
ranger.num.trees	integer	[1, 2000]	
ranger.sample.fraction	continuous	[0.1, 1]	
ranger.mtry.power	integer	[0, 1]	
ranger.respect.unordered.factors	categorical	{ignore, order, partition}	
ranger.min.node.size	integer	[1, 100]	
ranger.splitrule	categorical	{gini, extratrees}	
ranger.num.random.splits	integer	[1, 100]	
aknn.k	integer	[1, 50]	
aknn.distance	categorical	{l2, cosine, ip}	
aknn.M	integer	[18, 50]	
aknn.ef	integer	[7, 403]	log
aknn.ef_construction	integer	[7, 403]	log
xgboost.booster	categorical	{gblinear, gbtree, dart}	
xgboost.nrounds	integer	[7, 2980]	log
xgboost.eta	continuous	[0.001, 1]	log
xgboost.gamma	continuous	[4.5e-05, 7.4]	log
xgboost.lambda	continuous	[0.001, 1097]	log
xgboost.alpha	continuous	[0.001, 1097]	log
xgboost.subsample	continuous	[0.1, 1]	
xgboost.max_depth	integer	[1, 15]	
xgboost.min_child_weight	continuous	[2.72, 148.41]	log
xgboost.colsample_bytree	continuous	[0.01, 1]	
xgboost.colsample_bylevel	continuous	[0.01, 1]	
xgboost.rate_drop	continuous	[0, 1]	
xgboost.skip_drop	continuous	[0, 1]	
trainsize	continuous	[0.03, 1]	budget
imputation	categorical	impute.{mean, median, hist}	
learner_id	categorical	{aknn, glmnet, ranger, rpart, svm, xgboost}	

Table 10: Search space of the *nb301* scenario. We summarize multiple parameters (using, e.g., $\{3-5\}$ if parameters with suffix 3 through 5 are present).

Hyperparameter	Type	Range	Info
NetworkSelectorDatasetInfo_COLON_darts_COLON_edge_normal_**{0-13}**	categorical	{max_pool_3x3, avg_pool_3x3, skip_connect, sep_conv_3x3, sep_conv_5x5, dil_conv_3x3, dil_conv_5x5}	
NetworkSelectorDatasetInfo_COLON_darts_COLON_edge_reduce_**{0-13}**	categorical	{max_pool_3x3, avg_pool_3x3, skip_connect, sep_conv_3x3, sep_conv_5x5, dil_conv_3x3, dil_conv_5x5}	
NetworkSelectorDatasetInfo_COLON_darts_COLON_inputs_node_normal_**{3-5}**	categorical	{0_1, 0_2, 1_2}	
NetworkSelectorDatasetInfo_COLON_darts_COLON_inputs_node_reduce_**{3-5}**	categorical	{0_1, 0_2, 1_2}	
epoch	integer	[1, 98]	budget

Table 11: Search Space of the *lcbench* scenario.

Hyperparameter	Type	Range	Info
epoch	integer	[1, 52]	budget
batch_size	integer	[16, 512]	log
learning_rate	continuous	[1e-04, 0.1]	log
momentum	continuous	[0.1, 0.9]	
weight_decay	continuous	[1e-05, 0.1]	
num_layers	integer	[1, 5]	
max_units	integer	[64, 1024]	log
max_dropout	continuous	[0, 1]	

first publicly available benchmark that combines performance, resource usage and interpretability of models allowing for the construction of interesting multi-objective benchmarks. Hyperparameter configurations were evaluated at different fidelity steps (training sizes of the following fractions: 0.05, 0.1, 0.2, 0.4, 0.6, 0.8, 1) achieved via incorporating resampling in the ML pipeline. The super learner scenario was constructed by using the data of all four base learners introducing conditional hyperparameters in the form of branching. In total, 5451872 different configurations were evaluated. Data collection was performed on the *moran* partition of the *ARCC Teton HPC* cluster of the University of Wyoming using `batchtools` [43] for job scheduling and took around 9.8 CPU years. Surrogate models were then fitted on the available data as described in Supplement D.1. Table 12 lists all hyperparameters of the search spaces of the *iaml_* scenarios. Instance ID's correspond to OpenML [71] dataset ids through which dataset properties can be queried[10]. OpenML tasks corresponding to each dataset can be obtained from [10].

[10] `https://www.openml.org/d/<dataset_id>`

Table 12: Search spaces of YAHPO Gym's *iaml_* scenarios. ⊦ indicates the parent in case dependencies between hyperparameters exist. The *super* scenario inherits dependencies from previous scenarios, while additional dependencies on the learner are introduced, indicated by a prefix.

iaml_glmnet

Hyperparameter	Type	Range	Info
alpha	continuous	[0, 1]	
s	continuous	[1e-04, 1000]	log
trainsize	continuous	[0.03, 1]	budget

iaml_rpart

Hyperparameter	Range	Type	Info
cp	continuous	[1e-04, 1]	log
maxdepth	integer	[1, 30]	
minbucket	integer	[1, 100]	
minsplit	integer	[1, 100]	
trainsize	continuous	[0.03, 1]	budget

iaml_ranger

Hyperparameter	Type	Range	Info
num.trees	integer	[1, 2000]	
replace	boolean	{TRUE, FALSE}	
sample.fraction	continuous	[0.1, 1]	
mtry.ratio	continuous	[0, 1]	
respect.unordered.factors	categorical	{ignore, order, partition}	
min.node.size	integer	[1, 100]	
splitrule	categorical	{gini, extratrees}	
num.random.splits	integer	[1, 100]	⊦ splitrule
trainsize	continuous	[0.03, 1]	budget

iaml_xgboost

Hyperparameter	Type	Range	Info
booster	categorical	{gblinear, gbtree, dart}	
nrounds	integer	[3, 2000]	log
eta	continuous	[1e-04, 1]	log, ⊦ booster
gamma	continuous	[1e-04, 7]	log, ⊦ booster
lambda	continuous	[1e-04, 1000]	log
alpha	continuous	[1e-04, 1000]	log
subsample	continuous	[0.1, 1]	
max_depth	integer	[1, 15]	⊦ booster
min_child_weight	continuous	[$\exp(1)$, 150]	log, ⊦ booster
colsample_bytree	continuous	[0.01, 1]	⊦ booster
colsample_bylevel	continuous	[0.01, 1]	⊦ booster
rate_drop	continuous	[0, 1]	⊦ booster
skip_drop	continuous	[0, 1]	⊦ booster
trainsize	continuous	[0.03, 1]	budget

iaml_super

Hyperparameter	Type	Range	Info
learner	categorical	{ranger, glmnet, xgboost, rpart}	
glmnet.alpha	continuous	[0, 1]	
glmnet.s	continuous	[1e-04, 1000]	log
rpart.cp	continuous	[1e-04, 1]	log
rpart.maxdepth	integer	[1, 30]	
rpart.minbucket	integer	[1, 100]	
rpart.minsplit	integer	[1, 100]	
ranger.num.trees	integer	[1, 2000]	
ranger.replace	boolean	{TRUE, FALSE}	
ranger.sample.fraction	continuous	[0.1, 1]	
ranger.mtry.ratio	continuous	[0, 1]	
ranger.respect.unordered.factors	categorical	{ignore, order, partition}	
ranger.min.node.size	integer	[1, 100]	
ranger.splitrule	categorical	{gini, extratrees}	
ranger.num.random.splits	integer	[1, 100]	
xgboost.booster	categorical	{gblinear, gbtree, dart}	
xgboost.nrounds	integer	[3, 2000]	log
xgboost.eta	continuous	[1e-04, 1]	log
xgboost.gamma	continuous	[1e-04, 7]	log
xgboost.lambda	continuous	[1e-04, 1000]	log
xgboost.alpha	continuous	[1e-04, 1000]	log
xgboost.subsample	continuous	[0.1, 1]	
xgboost.max_depth	integer	[1, 15]	
xgboost.min_child_weight	continuous	[2.71828182845905, 150]	log
xgboost.colsample_bytree	continuous	[0.01, 1]	
xgboost.colsample_bylevel	continuous	[0.01, 1]	
xgboost.rate_drop	continuous	[0, 1]	
xgboost.skip_drop	continuous	[0, 1]	
trainsize	continuous	[0.03, 1]	budget

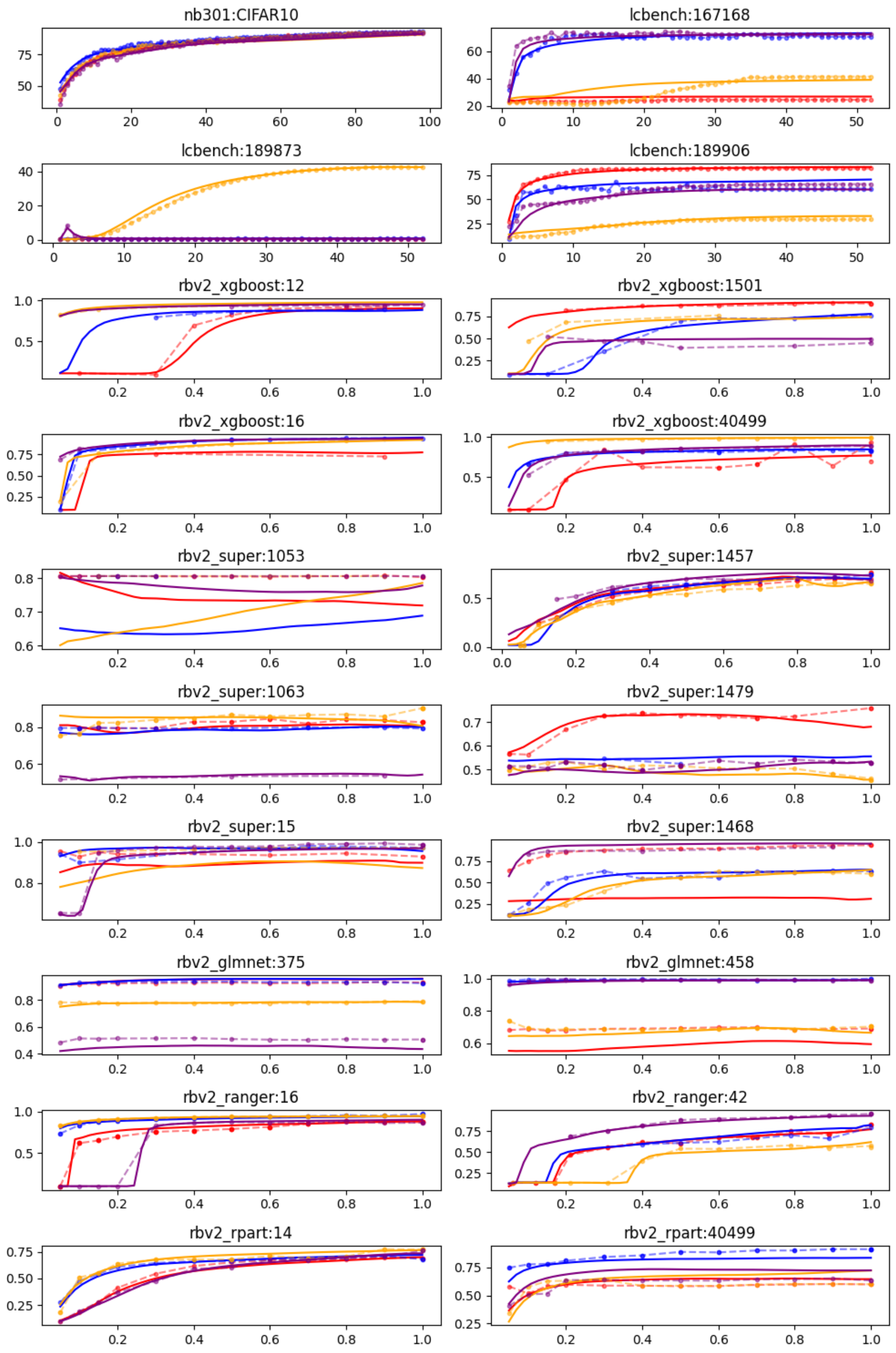


Figure 5: Predicted learning curves (lines) together with true learning curves (dotted) for 4 randomly drawn configurations (differentiated by colour) out of each instance in YAHPO-MO reporting the respective target metric. Best viewed in color.

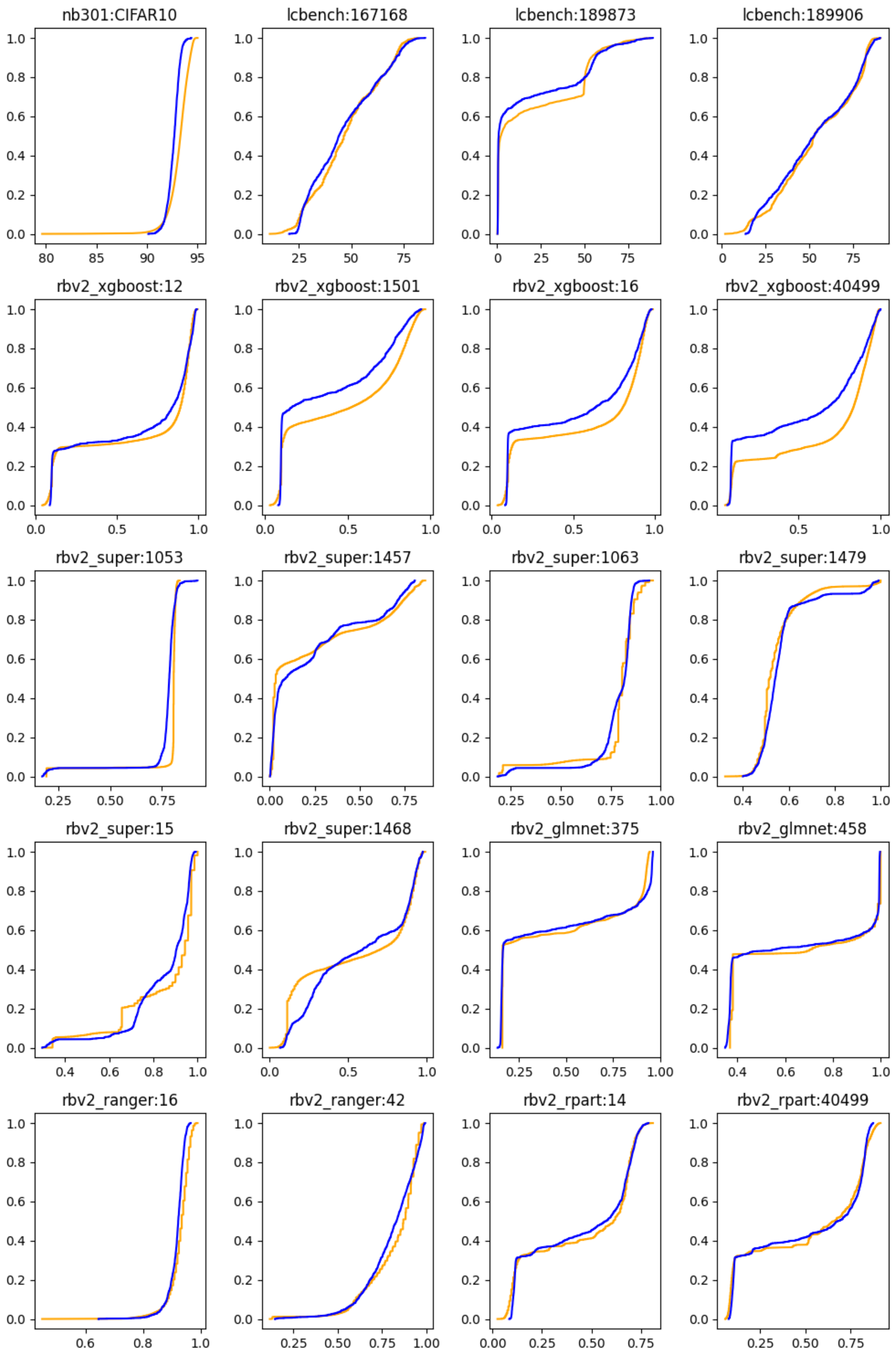


Figure 6: Empirical Cumulative Distribution Function (ECDF) for surrogate predictions (blue) and real evaluations (orange). Best viewed in color.

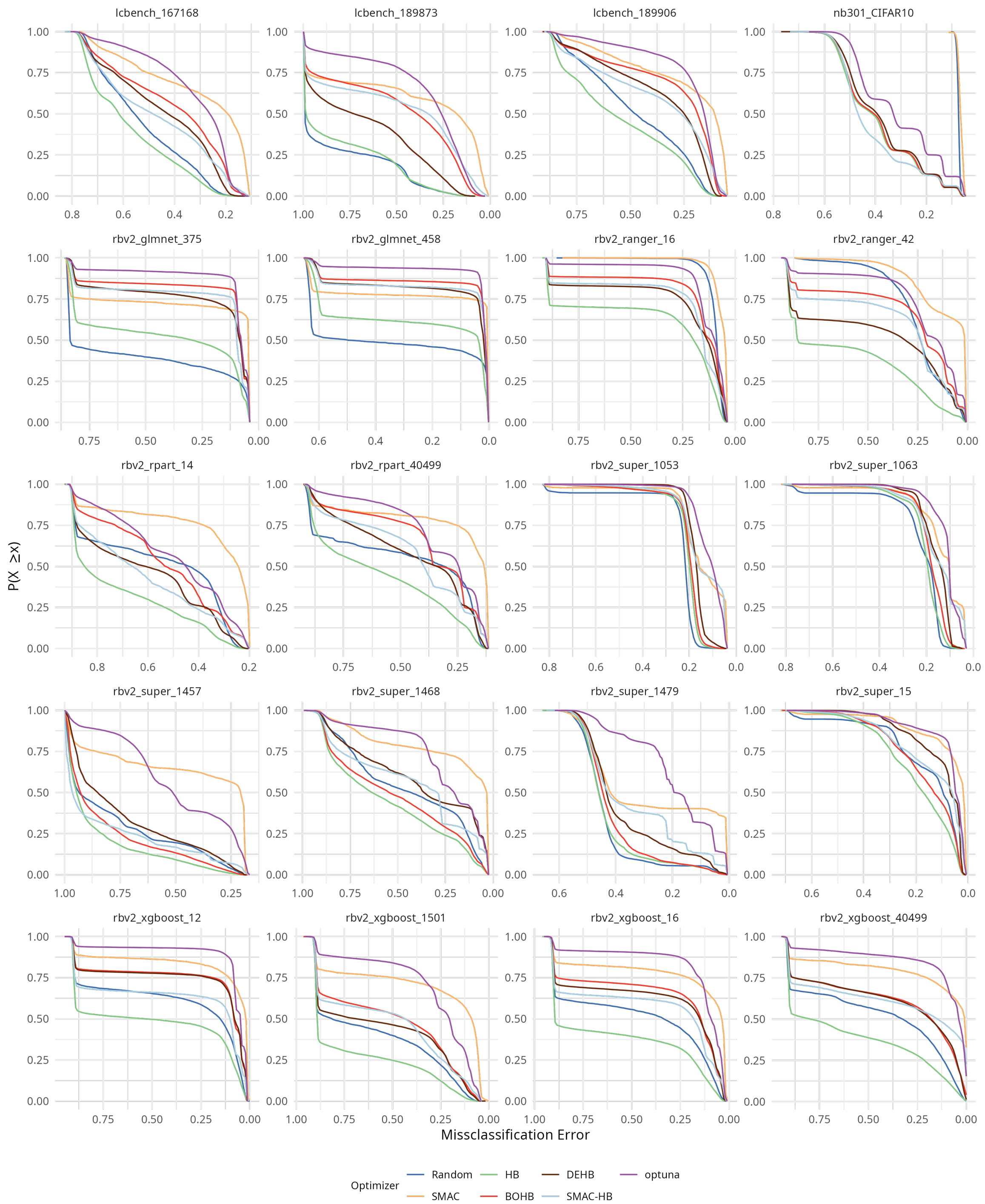


Figure 7: Empirical Cumulative Distribution Function (ECDF) for optimizer traces on YAHPO-SO. Best viewed in color.

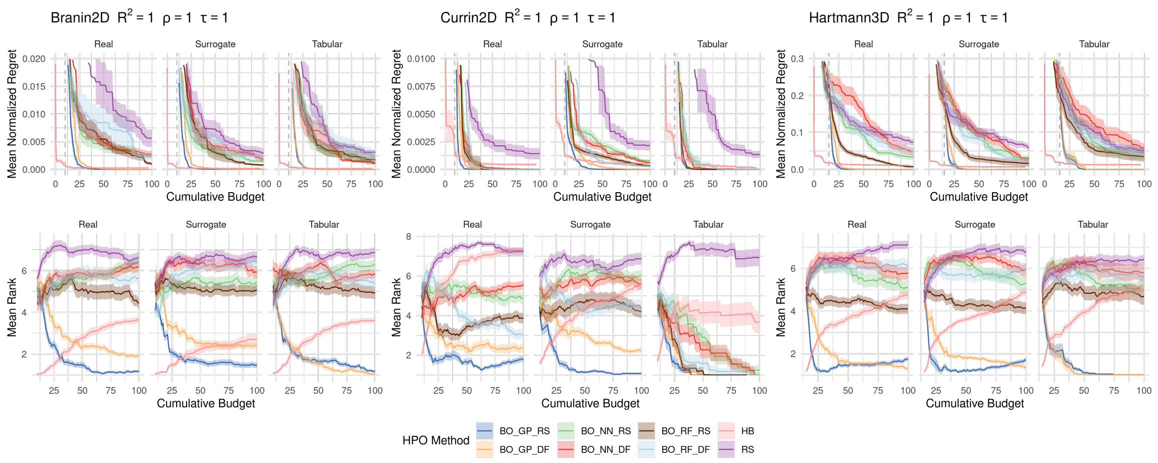


Figure 8: Mean normalized regret (top) and mean ranks (bottom) of different HPO methods on different benchmarks. Ribbons represent standard errors. The gray vertical line indicates the cumulative budget used for the initial design of BO methods. Performance measures of the surrogate benchmarks are stated after the benchmark function. 30 replications.

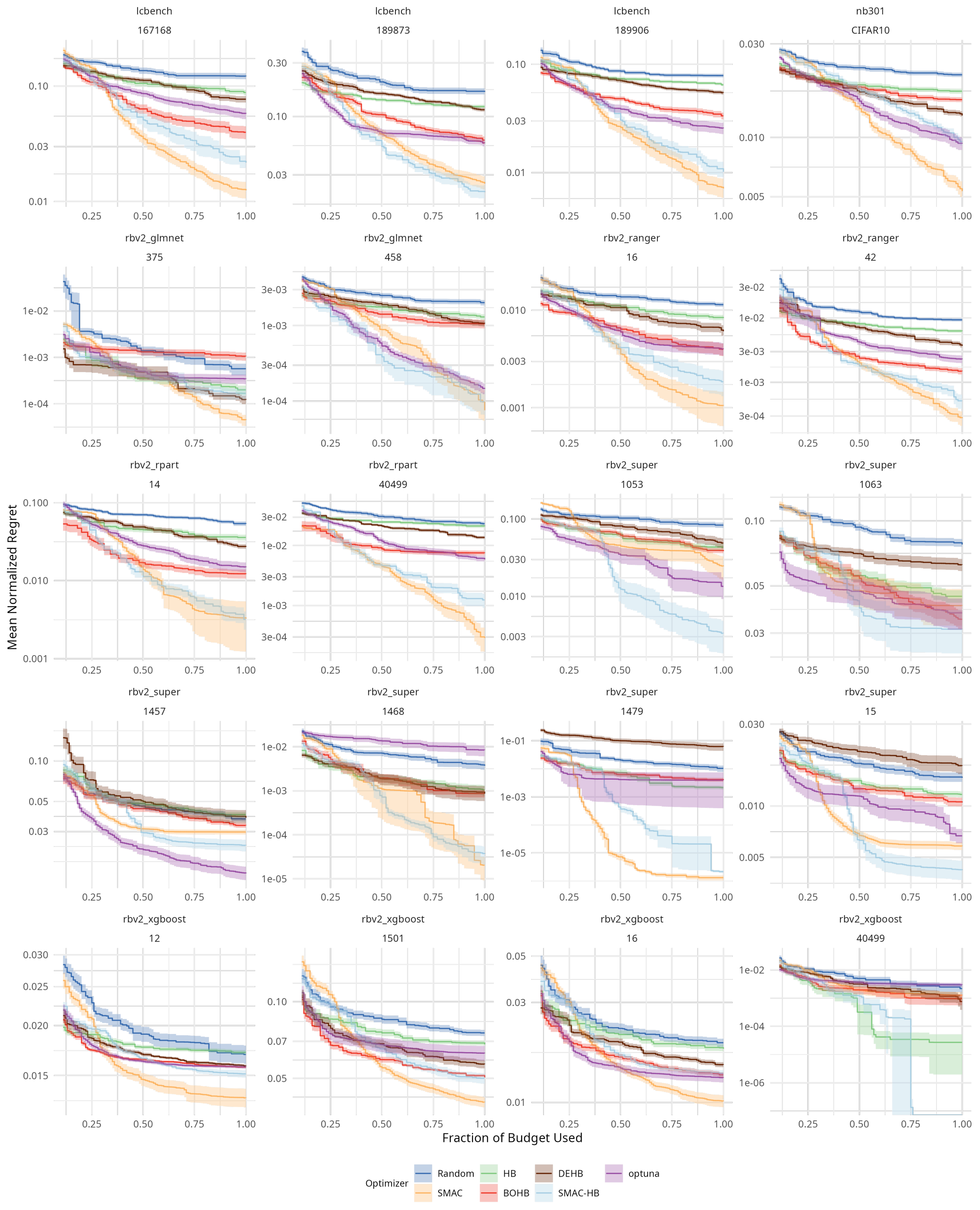


Figure 9: Mean normalized regret of HPO methods separate for each benchmark instance. x-axis starts after 10% of the optimization budget has been used.

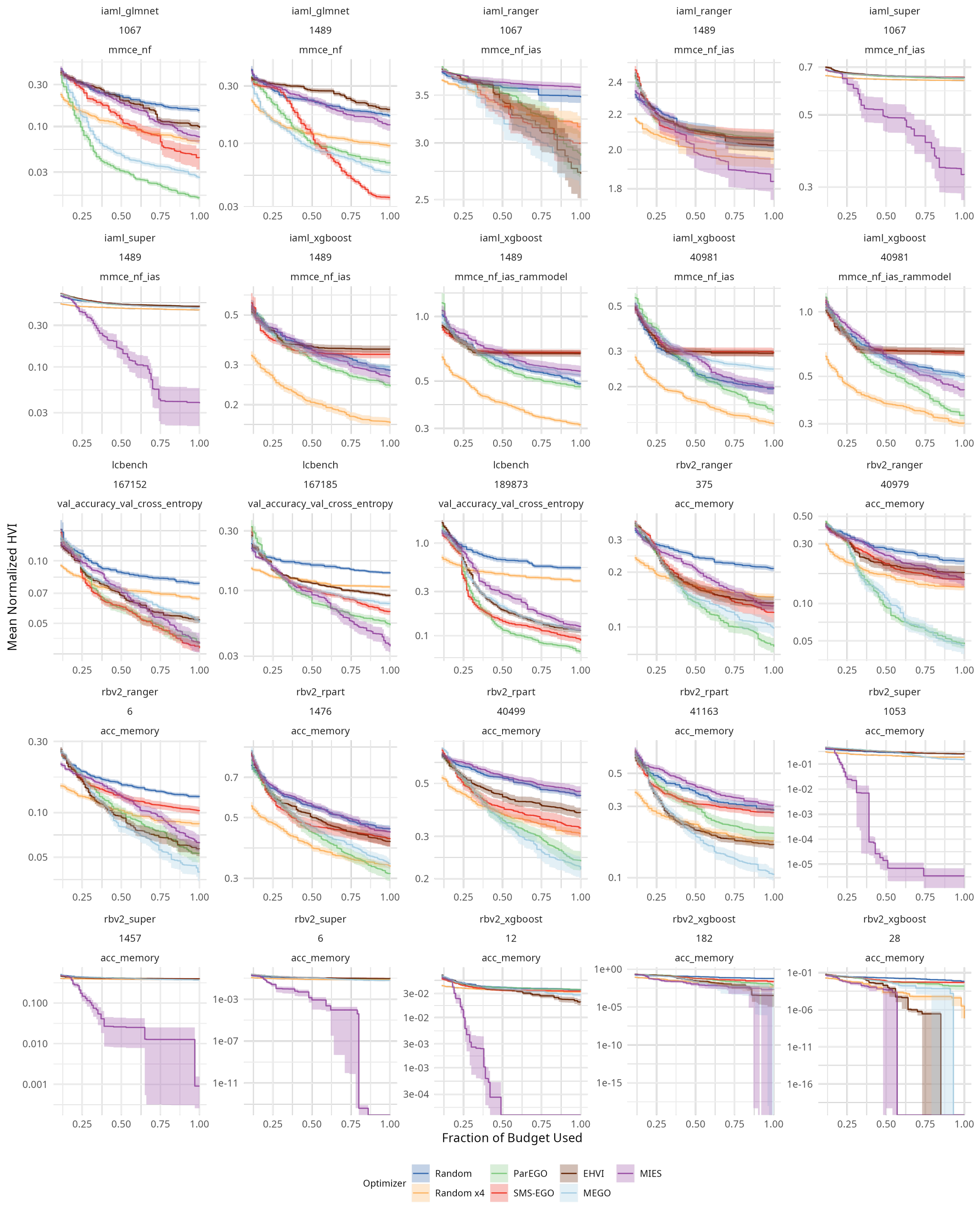


Figure 10: Mean normalized Hypervolume Indicator of HPO methods separate for each benchmark instance. x-axis starts after 10% of the optimization budget has been used.

4.8 Mutation is all you need

Contributed Article:
L. Schneider, F. Pfisterer, M. Binder, and B. Bischl. Mutation is all you need. In *AutoML Workshop at ICML*, 2021, arXiv:2107.07343

Declaration of contributions The contribution was developed based on the results of a master thesis by LS supervised by FP, MB and BB. The code, methodology, and manuscript were largely written by LS with input by FP, MB and BB who also revised and improved the final manuscript. FP and MB also provided feedback regarding the design of benchmark experiments which were then carried out by LS.

8th ICML Workshop on Automated Machine Learning (2021)

Mutation is all you need

Lennart Schneider LENNART.SCHNEIDER@STAT.UNI-MUENCHEN.DE
Florian Pfisterer FLORIAN.PFISTERER@STAT.UNI-MUENCHEN.DE
Martin Binder MARTIN.BINDER@STAT.UNI-MUENCHEN.DE
Bernd Bischl BERND.BISCHL@STAT.UNI-MUENCHEN.DE
Department of Statistics, LMU Munich, Germany

Abstract

Neural architecture search (NAS) promises to make deep learning accessible to non-experts by automating architecture engineering of deep neural networks. BANANAS is one state-of-the-art NAS method that is embedded within the Bayesian optimization framework. Recent experimental findings have demonstrated the strong performance of BANANAS on the NAS-Bench-101 benchmark being determined by its path encoding and not its choice of surrogate model. We present experimental results suggesting that the performance of BANANAS on the NAS-Bench-301 benchmark is determined by its acquisition function optimizer, which minimally mutates the incumbent.

1. Introduction

Neural architecture search (NAS) methods can be categorized along three dimensions (Elsken et al., 2019a): search space, search strategy, and performance estimation strategy. Focusing on search strategy, popular methods are given by Bayesian optimization (BO, e.g., Bergstra et al. 2013; Domhan et al. 2015; Mendoza et al. 2016; Kandasamy et al. 2018; White et al. 2019), evolutionary methods (e.g., Miller et al. 1989; Liu et al. 2017; Real et al. 2017, 2019; Elsken et al. 2019b), reinforcement learning (RL, e.g., Zoph and Le 2017; Zoph et al. 2018), and gradient-based algorithms (e.g., Liu et al. 2019; Pham et al. 2018).

Within the BO framework, BANANAS (White et al., 2019) has emerged as one state-of-the-art algorithm (White et al., 2019; Siems et al., 2020; Guerrero-Viu et al., 2021; White et al., 2021). The two main components of BANANAS are a (truncated) path encoding, where architectures represented as directed acyclic graphs (DAG) are encoded based on the possible paths through that graph, and an ensemble of feed-forward neural networks as surrogate model. Recently, White et al. (2021) investigated the performance of different surrogate models in the context of BO-based NAS and concluded that the strong performance of BANANAS on the NAS-Bench-101 benchmark (Ying et al., 2019) is determined by its path encoding and not its choice of surrogate model. Results suggest that path encoding leads to a performance boost on smaller search spaces (such as the one of NAS-Bench-101) but does not scale well on larger search spaces such as DARTS (Liu et al., 2019).

We hypothesize that for larger search spaces, the strong performance of BANANAS stems from its choice of acquisition function optimizer in the sense that local optimization of architectures is most important and other components have less impact on performance. To investigate this hypothesis, we vary the main BANANAS components, namely architecture representation, surrogate model, acquisition function and acquisition function optimizer in a factorial manner and examine the performance difference on the NAS-Bench-301 benchmark (Siems et al., 2020)[1].

2. BANANAS

BANANAS (White et al., 2019) uses a (truncated) path encoding, combined with an ensemble of feed-forward neural networks as surrogate model, to predict the performance of architectures. Cell-based search spaces such as DARTS can be encoded by representing cells as DAGs, with nodes as vertices and connections with operations between them as edges. For every *path*, i.e., every possible ordering of vertices, a binary feature is generated, indicating whether the DAG contains all directed edges along this path. If architectures are created by sampling edges in the DAG subject to a maximum edge constraint (i.e., limiting the number of edges), most possible paths have a low probability of occurring (White et al., 2019; Ying et al., 2019). Therefore, BANANAS truncates the least-likely paths, resulting in a relatively informative encoding that scales linearly with the size of the cell.

Let $\mathcal{A}$ denote the search space of architectures and $\{f_m\}_{m=1}^{M}$ denote an ensemble of M feed-forward neural networks (`NN`)[2], where $f_m : \mathcal{A} \rightarrow \mathbb{R}$. BANANAS uses independent Thompson sampling (`ITS`, Thompson 1933; White et al. 2019) as acquisition function:

$$\alpha_{\mathrm{ITS}}(x) = \tilde{f}_x(x),\ \tilde{f}_x(x) \sim \mathcal{N}(\hat{f}, \hat{\sigma}^2), \tag{1}$$

where $\hat{f} = \frac{1}{M}\sum_{m=1}^{M} f_m(x)$ and $\hat{\sigma} = \sqrt{\frac{\sum_{m=1}^{M}(f_m(x)-\hat{f})^2}{M-1}}$. $\alpha_{\mathrm{ITS}}(\cdot)$ is then optimized using the following mutation algorithm (`Mut`): The best performing architecture so far is selected and mutated in 100 different ways by changing a single operation or edge randomly and the architecture yielding the largest acquisition value is proposed as the next candidate for evaluation.

3. Experiments

To investigate the effectiveness of different components of BANANAS on NAS-Bench-301, we conducted a series of experiments where we replaced some of them with what we consider more "standard" choices. A simpler configuration could use a random forest (`RF`, Breiman 2001; notably used successfully in SMAC, Hutter et al. 2011) as a surrogate model which can either be fitted to path encodings (`Path`) or natural tabular representations (`Tabular`) of the architectures as provided in NAS-Bench-301 in the form of a ConfigSpace (see the `ConfigSpace` library, Lindauer et al. 2019). In the tabular encoding, architectures are represented by enumerating all nodes and potential edges and introducing categorical hyperparameters for each operation along each potential edge, where the nodes serving as input of each intermediate node are again defined as categorical hyperparameters and operations on a certain edge can only be specified if this edge is actually present in the DAG

1. NAS-Bench-301 uses architectures of the DARTS search space trained and evaluated on CIFAR-10 (Krizhevsky, 2009)
2. White et al. (2019) use $M = 5$ sequential fully-connected networks with 10 layers of width 20 by default, initialized with different random weights and trained using permuted training sets, the Adam optimizer with a learning rate of 0.01, and mean absolute error (MAE) loss

(Siems et al., 2020). Note that another possible architecture representation is given by adjacency matrix encoding (Ying et al., 2019; White et al., 2020a), which was not considered by us. Looking at the acquisition function, the expected improvement (`EI`) is a well-known alternative:

$$\alpha_{\mathrm{EI}}(x) = \mathrm{E}_y[\max(y - y_{\max}, 0)], \tag{2}$$

given in Jones et al. (1998), where in our context $y_{\max}$ is the best validation accuracy observed so far and y is the surrogate prediction of architecture x. As a very simple alternative, one could also only be interested in the posterior mean prediction (`Const. Mean`) as acquisition function, which does not take the surrogate model uncertainty estimates into account. Finally, looking at acquisition function optimizers, a popular choice is given by random search (`RS`): Drawing a large number of architectures uniformly at random (e.g., by sampling from the ConfigSpace) and selecting the architecture with the largest acquisition value. Our `RS` method samples 1000 architectures in each BO iteration.

3.1 Different BANANAS Configurations on NAS-Bench-301

Choices for the architecture encodings, surrogate candidates, acquisition functions, and acquisition function optimizers were crossed in a full factorial manner (where possible), resulting in overall 18 different algorithms. BANANAS, local search (`LS`) and random search (as NAS method, `Random`) were used as implemented in `naszilla` (White et al., 2020a). In `LS` (White et al., 2020b), all neighbors (e.g., all architectures differing in one operation or edge) of an incumbent are evaluated and the incumbent is replaced if a better architecture has been found and the process is repeated until no better architecture can be found (i.e., a local optimum is reached) or another termination criterion is met. Regarding the reference BANANAS implementation, two configurations were used differing in the frequency of updating their ensemble of feed-forward networks ($k = 1$, i.e., after every iteration, or $k = 10$, see White et al. 2019). The initial design for all methods consisted of ten architectures that were sampled uniformly at random (note that `LS` and `Random` do not rely on an initial design and simply start from zero evaluations). All methods were run for 100 iterations (architecture evaluations) and all runs were replicated 20 times. Results are shown in Figure 1, where the validation accuracy is plotted against the batch number. Note that in each facet, the reference `naszilla` implementations of BANANAS, `LS`, and `Random` are provided and by design, `Paths + NN + ITS + Mut` is a (re-)implementation of the BANANAS ($k = 1$) configuration. In general, using `Mut` as acquisition function optimizer always results in a strong performance boost compared to using `RS`. Notably, BANANAS' ensemble of feed-forward neural networks, together with path encoding only performs well if combined with `Mut` and is otherwise outperformed by `Random`. Moreover, the very simple configuration of `Tabular + RF + EI + Mut` performs similarly to the reference BANANAS implementation. Finally, neglecting all uncertainty in the predictions by opting for the `Const. Mean` acquisition function results in very good performance when combined with `Tabular + RF + Mut`. Performing a one-way ANOVA on the top seven algorithms indicated no significant difference in final performance, $F(6, 133) = 1.026, p = 0.411$. Table 1 presents results of a four-way ANOVA on the final performance of the 18 algorithms outlined above with respect to the factors architecture encoding, surrogate candidate, acquisition

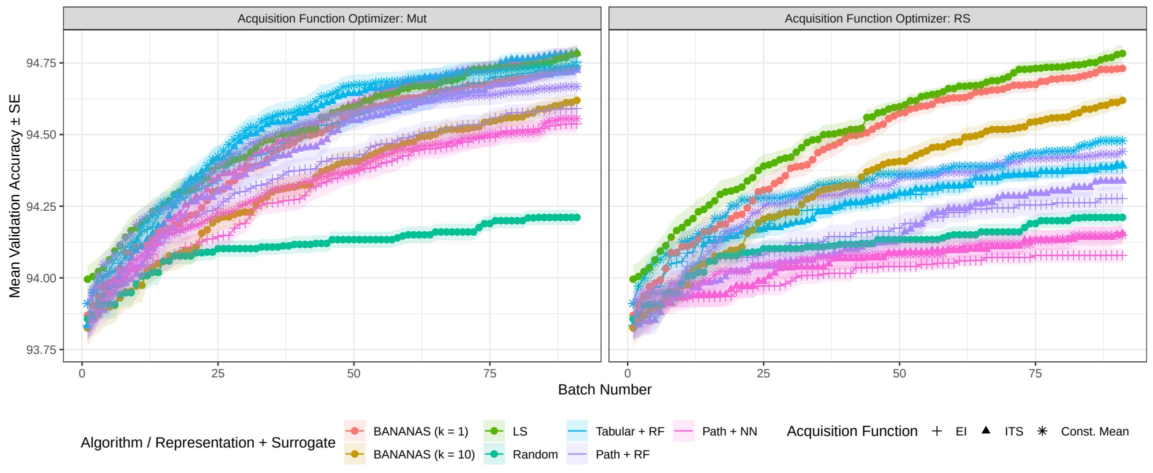


Figure 1: Different BANANAS configurations on NAS-Bench-301. Mean validation accuracy with standard error bands, higher is better. Color: optimization method and surrogate model. Facet: acquisition function optimizer, where applicable. Point shape: acquisition function, where applicable. The `ITS` acquisition function and `Mut` acquisition function optimizer is used for BANANAS methods, and `LS` and `Random` do not use an acquisition function; their accuracy is therefore shown in both facets of the graph.

function, and acquisition function optimizer. The acquisition function optimizer is by far the most important determinant of final performance.

	Sum Sq	Df	F value	Pr(>F)
Architecture Encoding	0.41	1	19.57	0.0000
Surrogate Candidate	1.01	1	48.31	0.0000
Acquisition Function	0.56	2	13.49	0.0000
Acq. F. Optimizer	13.18	1	632.43	0.0000
Residuals	7.38	354		

Table 1: Results of a four-way ANOVA on the factors architecture encoding, surrogate candidate, acquisition function, and acquisition function optimizer. Type II sums of squares.

3.2 Examining the Effect of the Acquisition Function Optimizer

To investigate the performance difference with respect to the acquisition function optimizers, another experiment was conducted. Based on the `Tabular + RF + EI` configuration three different acquisition function optimizers were compared: Random search with 100000 architectures drawn uniformly at random in each BO iteration (`RS+`), random search as described above (`RS`) and `Mut` as described above. Ten architectures were sampled uniformly at random and used as the initial design points for all replications. All methods were run for 100 iterations (architecture evaluations) and all runs were replicated 20 times. Results are given in Figure 2A. As can be seen, `Mut` strongly outperforms even the `RS+` optimizer.

We collected additional data in the `RS+` runs shown in Figure 2A. In each BO iteration of these runs, we also performed acquisition function optimization using the other two methods (`RS` and `Mut`) and investigated the properties of the proposed architectures. While the op-

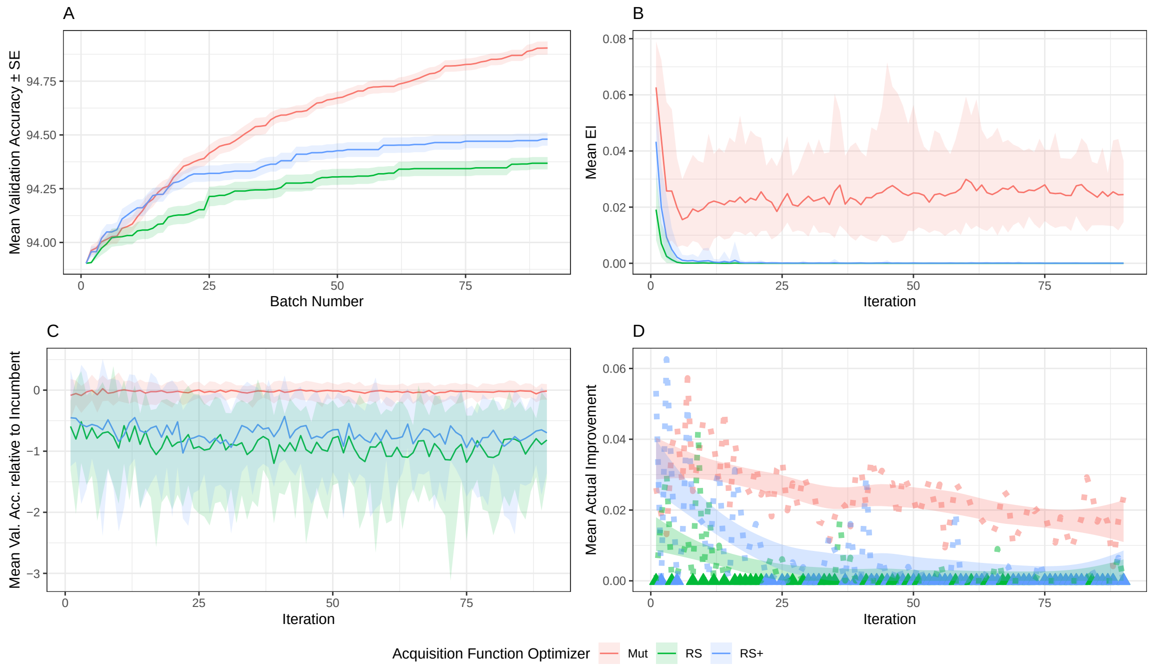


Figure 2: `Tabular + RF + EI` with different acquisition function optimizers on NAS-Bench-301. A: Validation accuracy. B: `EI`. C: Validation accuracy relative to the incumbent. D: Actual improvement. Ribbons in B and C represent 2.5% and 97.5% quantiles. In D, LOESS smoothing was performed and triangles indicate no improvement.

timization itself proceeded with the architectures proposed by `RS+`, the collected data gives information about the quality of architecture proposals done by the other methods. The data collected was the `EI` of each proposed architecture, according to the surrogate model (Figure 2B), the actual validation accuracy of each proposed architecture (when evaluated), minus the validation accuracy of the incumbent during that iteration (Figure 2C), and that same quantity, conditional on the proposed architecture giving higher validation accuracy than the incumbent ("actual improvement", Figure 2D).

`Mut` results in both higher `EI` and actual improvement, i.e., `Mut` solves the inner optimization problem better than the other optimizers and the actual improvement is comparably large. Note that the difference between the validation accuracy of proposed architectures and incumbent is mostly negative due to a fixed iteration seldom resulting in actual improvement. Looking at Figure 2D, we observe that following the proposals by `RS+` and `RS` results in many iterations with no improvement (as indicated by triangles).

In a final experiment, focus was given to the accuracy of the surrogate model when predicting the validation accuracy of architectures depending on the edit distance to the incumbent. Based on the `Tabular + RF + EI + Mut` configuration, the BO loop was run for 50 iterations (architecture evaluations); the construction of the initial design remained the same and all runs were replicated 100 times. For edit distances ranging from 1 to 8, 100 test architectures were constructed each by mutating a fixed number of parameters (operations or edges) of the incumbent. For these test architectures, Kendall's τ with respect to the predicted and true validation accuracy (after evaluation) is given in Figure 3A.

Additionally, the true validation accuracy is plotted against the edit distance (Figure 3B), with the gray point representing the incumbent. In Figure 3C, the expected improvement and the actual improvement is plotted. While the true validation accuracy decreases when increasing the edit distance, Kendall's τ increases, suggesting that the surrogate model is not capable of precise performance prediction for high performing architectures close to the incumbent. This finding goes in line with results of White et al. (2021) that model based NAS methods perform bad when predicting the performance of neighbors of high performing architectures when the search space is large. Moreover, the expected improvement is relatively unaffected by the edit distance, although the actual improvement is largest for close architectures. This may indicate that thorough optimization of the acquisition function is not needed, instead simply considering neighboring architectures as candidates may be sufficient.

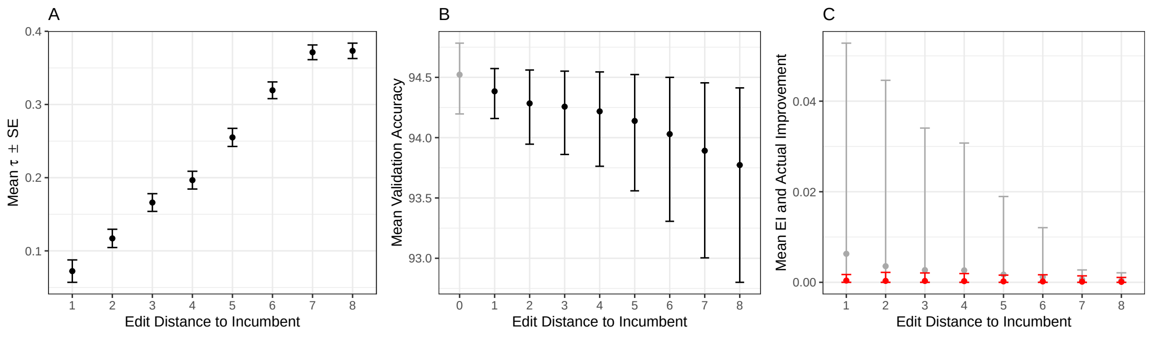


Figure 3: `Tabular + RF + EI + Mut` on NAS-Bench-301. A: Kendall's τ of the predicted and true validation accuracy of test architectures constructed to have different edit distances to the incumbent. B: True validation accuracy of these test architectures. Validation accuracy of the incumbent is given in gray. C: Expected Improvement (red) and actual improvement (gray) of these test architectures. Bars in B and C represent 2.5% and 97.5% quantiles.

4. Discussion

We have presented empirical results suggesting that the performance of BANANAS on large cell-based search spaces such as DARTS is predominantly determined by its choice of acquisition function optimizer that is effectively performing a randomized local search. Other components such as the architecture encoding, surrogate model and acquisition function have a comparably small effect on the performance, and exchanging most components of BANANAS with more "standard" choices results in a method that is not significantly worse. Local search, which uses no surrogate model at all, does in fact perform equally well (at least on the NAS-Bench-301 benchmark), giving more evidence that the local nature of BANANAS' mutation acquisition function optimization contributes mainly to its success. Minimally mutating the incumbent allows for solving the inner acquisition function optimization problem better than random search variants with large budget, although the surrogate model suffers from imprecise surrogate predictions for architectures close in edit distance to the incumbent. Future work on BO methods for NAS should therefore also focus on algorithms for solving the inner acquisition function optimization problem.

References

M. Becker, J. Richter, M. Lang, B. Bischl, and M. Binder. *bbotk: Black-Box Optimization Toolkit*, 2021. https://bbotk.mlr-org.com, https://github.com/mlr-org/bbotk.

H. Bengtsson. A unifying framework for parallel and distributed processing in R using futures. arXiv:2008.00553 [cs.DG], 2020.

J. Bergstra, D. Yamins, and D. Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *Proceedings of the 30th International Conference on Machine Learning*, pages 115–123, 2013.

M. Binder, F. Pfisterer, L. Schneider, B. Bischl, M. Lang, and S. Dandl. *mlr3pipelines: Preprocessing Operators and Pipelines for 'mlr3'*, 2020. URL `https://CRAN.R-project.org/package=mlr3pipelines`. R package version 0.3.0.

L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

T. Domhan, J. T. Springenberg, and F. Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Proceedings of the 24th International Conference on Artificial Intelligence*, page 3460–3468, 2015.

M. Dowle and A. Srinivasan. *data.table: Extension of 'data.frame'*, 2021. URL `https://CRAN.R-project.org/package=data.table`. R package version 1.14.0.

T. Elsken, J. H. Metzen, and F. Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019a.

T. Elsken, J. H. Metzen, and F. Hutter. Efficient multi-objective neural architecture search via Lamarckian evolution. In *Proceedings of the International Conference on Learning Representations*, 2019b.

J. Guerrero-Viu, S. Hauns, S. Izquierdo, G. Miotto, S. Schrodi, A. Biedenkapp, T. Elsken, D. Deng, M Lindauer, and F. Hutter. Bag of baselines for multi-objective joint neural architecture search and hyperparameter optimization. arXiv:2105.01015 [cs.LG], 2021.

F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization*, pages 507–523, 2011.

D. R. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13(4):455–492, 1998.

K. Kandasamy, W. Neiswanger, J. Schneider, B. Poczos, and E. Xing. Neural architecture search with Bayesian optimisation and optimal transport. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, 2018.

A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

M. Lang, M. Binder, J. Richter, P. Schratz, F. Pfisterer, S. Coors, Q. Au, G. Casalicchio, L. Kotthoff, and B. Bischl. mlr3: A modern object-oriented machine learning framework in R. *Journal of Open Source Software*, dec 2019.

M. Lindauer and F. Hutter. Best practices for scientific research on neural architecture search. arXiv:1909.02453 [cs.LG], 2019.

M. Lindauer, K. Eggensperger, M. Feurer, A. Biedenkapp, J. Marben, P. Müller, and F. Hutter. BOAH: A tool suite for multi-fidelity Bayesian optimization & analysis of hyperparameters. arXiv:1908.06756 [cs.LG], 2019.

H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu. Hierarchical representations for efficient architecture search. In *Proceedings of the International Conference on Learning Representations*, 2017.

H. Liu, K. Simonyan, and Y. Yang. DARTS: Differentiable architecture search. In *Proceedings of the International Conference on Learning Representations*, 2019.

H. Mendoza, A. Klein, M. Feurer, J. T. Springenberg, and F. Hutter. Towards automatically-tuned neural networks. In *ICML Workshop on Automatic Machine Learning*, 2016.

G. F. Miller, P. M. Todd, and S. U. Hegde. Designing neural networks using genetic algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 379–384, 1989.

H. Pham, M. Guan, B. Zoph, Q. V. Le, and J. Dean. Efficient neural architecture search via parameter sharing. In *Proceedings of the 35th International Conference on Machine Learning*, pages 4095–4104, 2018.

R Core Team. *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria, 2020. URL `https://www.R-project.org/`.

E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin. Large-scale evolution of image classifiers. In *Proceedings of the 34th International Conference on Machine Learning*, page 2902–2911, 2017.

E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 4780–4789, 2019.

J. Richter, M. Becker, M. Lang, B. Bischl, M. Binder, and J. Moosbauer. *mlr3mbo: Flexible Bayesian Optimization in R*, 2021. https://mlr3mbo.mlr-org.com, https://github.com/mlr-org/mlr3mbo.

J. Siems, L. Zimmer, A. Zela, J. Lukasik, M. Keuper, and F. Hutter. NAS-Bench-301 and the case for surrogate benchmarks for neural architecture search. arXiv:2008.09777 [cs.LG], 2020.

W. R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933.

K. Ushey, JJ. Allaire, and Y. Tang. *reticulate: Interface to 'Python'*, 2020. URL `https://CRAN.R-project.org/package=reticulate`. R package version 1.18.

C. White, W. Neiswanger, and Y. Savani. BANANAS: Bayesian optimization with neural architectures for neural architecture search. arXiv:1910.11858 [cs.LG], 2019.

C. White, W. Neiswanger, S. Nolen, and Y. Savani. A study on encodings for neural architecture search. In *Proceedings of the 34th Conference on Neural Information Processing Systems*, 2020a.

C. White, S. Nolen, and Y. Savani. Local search is state of the art for neural architecture search benchmarks. In *ICML Workshop on Automatic Machine Learning*, 2020b.

C. White, A. Zela, B. Ru, Y. Liu, and F. Hutter. How powerful are performance predictors in neural architecture search? arXiv:2104.01177 [cs.LG], 2021.

M. N. Wright and A. Ziegler. ranger: A fast implementation of random forests for high dimensional data in C++ and R. *Journal of Statistical Software*, 77(1):1–17, 2017.

C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter. NAS-Bench-101: Towards reproducible neural architecture search. In *Proceedings of the 36th International Conference on Machine Learning*, pages 7105–7114, 2019.

B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. In *Proceedings of the International Conference on Learning Representations*, 2017.

B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8697–8710, 2018.

Appendix A. Computational Details

The BO algorithms were implemented in R (R Core Team, 2020) within the `mlr3` (Lang et al., 2019) ecosystem relying on `mlr3mbo` (version 0.0.0.9999; Richter et al. 2021) and `bbotk` (version 0.3.0.9999; Becker et al. 2021). Random forests were used as implemented in the `mlr3extralearners` package wrapping `ranger::ranger` (version 0.12.1; Wright and Ziegler 2017) with `num.trees` set to `500`, `se.method` set to `"jack"`, and `respect.unordered.factors` set to `"order"`. Missing values were encoded with a new level ".missing" via a preprocessing pipeline built using `mlr3pipelines` (version 0.3.0; Binder et al. 2020).

Python 3.8.7 was used via the `reticulate` package (version 1.18; Ushey et al. 2020) within R. For NAS-Bench-301, `nasbench301` version 0.2 (Siems et al., 2020) was used relying on the `xgb_v1.0` surrogate model for the validation accuracy. The feed-forward ensemble of neural networks and path encoding as used by BANANAS was directly adopted as implemented in `naszilla` (version 1.0; White et al. 2020a). BANANAS, local search and random search (as NAS methods) were run using `naszilla` employing the same `nasbench301` setup as described above under Python 3.6.12 (due to different module requirements).

All computations were performed on 2 Intel© Xeon© E5-2650 v2 @ 2.60GHz CPUs each with 16 threads using R 4.0.3 under Ubuntu 20.04.1 LTS. Parallelization in R was done via the `future` (Bengtsson, 2020) and `future.apply` (Bengtsson, 2020) packages (version 1.21.0 and 1.7.0) on top of the internal parallelization of the `data.table` (Dowle and Srinivasan, 2021) package (version 1.14.0).

Appendix B. NAS Best Practices Checklist

Here, we answer to applicable questions of the NAS best practices checklist (version 1.0), see Lindauer and Hutter (2019).

- as NAS benchmark, NAS-Bench-301 (`nasbench301`) version 0.2 was used relying on the `xgb_v1.0` surrogate model (deterministic) for the validation accuracy
- all computations were run on the same hardware (2 Intel© Xeon© E5-2650 v2 @ 2.60GHz CPUs)
- all results reported are based on ablation studies
- the same evaluation protocol was used for all methods
- performance was compared with respect to the number of architecture evaluations
- random search was included as a NAS method
- multiple runs (20 or 100) were conducted; reproducibility with respect to algorithms implemented in R is given due to an initial random seed being set; regarding `naszilla`, no seed can be explicitly set

4.9 Tackling Neural Architecture Search With Quality Diversity Optimization

Contributed Article:
L. Schneider, F. Pfisterer, P. Kent, J. Branke, B. Bischl, and J. Thomas. Tackling neural architecture search with quality diversity optimization. In I. Guyon, M. Lindauer, M. van der Schaar, F. Hutter, and R. Garnett, editors, *Proceedings of the First International Conference on Automated Machine Learning*, volume 188 of *Proceedings of Machine Learning Research*, pages 9/1–30. PMLR, 25–27 Jul 2022

Declaration of contributions The project was initiated by LS, who also developed the core ideas and implementation of the methodology. FP contributed code for the use case on pruning studied in the paper. The manuscript was largely written by LS. FP contributed larger parts to the Appendix of the paper and reiterated parts of the Introduction and Conclusion. PK contributed the related work section on quality diversity optimization. FP, JT, PK, JB and BB advised throughout the project and further improved the manuscript.

Tackling Neural Architecture Search With Quality Diversity Optimization

Lennart Schneider[1] **Florian Pfisterer**[1] **Paul Kent**[2] **Juergen Branke**[3] **Bernd Bischl**[1] **Janek Thomas**[1]

[1]Department of Statistics, LMU Munich, Germany
[2]Mathematics of Real World Systems, University of Warwick, UK
[3]Warwick Business School, University of Warwick, UK

Abstract Neural architecture search (NAS) has been studied extensively and has grown to become a research field with substantial impact. While classical single-objective NAS searches for the architecture with the best performance, multi-objective NAS considers multiple objectives that should be optimized simultaneously, e.g., minimizing resource usage along the validation error. Although considerable progress has been made in the field of multi-objective NAS, we argue that there is some discrepancy between the actual optimization problem of practical interest and the optimization problem that multi-objective NAS tries to solve. We resolve this discrepancy by formulating the multi-objective NAS problem as a quality diversity optimization (QDO) problem and introduce three quality diversity NAS optimizers (two of them belonging to the group of multifidelity optimizers), which search for high-performing yet diverse architectures that are optimal for application-specific niches, e.g., hardware constraints. By comparing these optimizers to their multi-objective counterparts, we demonstrate that quality diversity NAS in general outperforms multi-objective NAS with respect to quality of solutions and efficiency. We further show how applications and future NAS research can thrive on QDO.

1 Introduction

The goal of neural architecture search (NAS) is to automate the manual process of designing optimal neural network architectures. Traditionally, NAS is formulated as a single-objective optimization problem with the goal of finding an architecture that has minimal validation error [13, 35, 45, 47, 46, 63]. Considerations for additional objectives such as efficiency have led to the formulation of constraint NAS methods that enforce efficiency thresholds [1] as well as multi-objective NAS methods [10, 12, 37, 53, 36] that yield a Pareto optimal set of architectures. However,

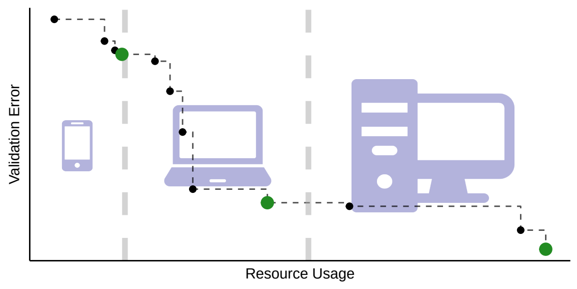


Figure 1: Optimizing neural network architectures for a discrete set of devices. We are interested in the best solution (green) within the constraints of the respective device (dashed vertical lines). Multi-objective optimization, in contrast, approximates the full Pareto front (black).

in most practical applications, we are not interested in the complete Pareto optimal set. Instead, we would like to obtain solutions for a discrete set of scenarios (e.g., end-user devices), which we henceforth refer to as *niches* in this paper. This is illustrated in Figure 1. A concrete example is finding neural architectures for microcontrollers [32] and other edge devices [38], e.g., in μNAS [32] architectures for "mid-tier" IoT devices are searched. To evaluate the benefits for larger devices, the search would need to be restarted with adapted constraints, thus wasting computational resources. Formulating the search as a multi-objective problem would also waste resources; once an architecture satisfies the constraints of a device, we are not interested in additional trade-offs, and we select only based on the validation error.

We therefore argue that the multi-objective NAS problem *can* and usually *should* be formulated as a *quality diversity optimization* (QDO) problem, which directly corresponds to the actual optimization problem of interest. The main contributions of this paper are: We (1) formulate multi-objective NAS as a QDO problem; (2) show how to adapt black-box optimization algorithms for the QDO setting; (3) modify existing QDO algorithms for the NAS setting; (4) propose novel multifidelity QDO algorithms for NAS; and (5) illustrate that our approach can be used to extend a broad range of NAS methods from conventional to Once-for-All methods.

2 Theoretical Background and Related Work

Let $\mathcal{A}$ denote a search space of architectures and Λ the search space of additional hyperparameters controlling the training of an architecture A. Furthermore, let $f_{\text{err}} : \mathcal{A} \times \Lambda \rightarrow \mathbb{R}$ denote the validation error obtained after training an architecture $A \in \mathcal{A}$ with a set of hyperparameters $\lambda \in \Lambda$ for a given number of epochs ($\lambda_{\text{epoch}} \in \lambda$). Typically, we consider $\lambda \in \Lambda$ to be fixed, except for λ_{epoch} in multifidelity methods, and we therefore omit λ in the following. The goal of single-objective NAS is to find the architecture with the lowest validation error, $A^* := \arg\min_{A \in \mathcal{A}} f_{\text{err}}(A)$.

NAS methods can be categorized along three dimensions: search space, search strategy, and performance estimation strategy [13]. For chain-structured neural networks (simply a connected sequence of layers), cell-based search spaces have gained popularity [46, 45]. In cell-based search spaces, different kinds of cells – typically, a normal cell preserving dimensionality of the input and a reduction cell reducing spatial dimension – are stacked in a predefined arrangement to form a final architecture. Regarding search strategy, popular methods utilize Bayesian optimization (BO) [3, 8, 39, 24, 57], evolutionary methods [41, 34, 47, 46, 12], reinforcement learning [63, 64], or gradient-based algorithms [35, 45]. For performance estimation, popular approaches leverage lower fidelity estimates [31, 14, 64] or make use of learning curve extrapolation [8, 27].

Multi-Objective Neural Architecture Search Contrary to the single-objective NAS formulation, multi-objective NAS does not solely aim for minimizing the validation error but simultaneously optimizes multiple objectives. These objectives typically take resource consumption – such as memory requirements, energy usage or latency – into account [10, 12, 37, 53, 36]. Denote by $f_1, \ldots, f_k$ the $k \geq 2$ objectives of interest, where typically $f_1 = f_{\text{err}}$ and denote by $\mathbf{f}(A)$ the vector of objective function values obtained for architecture $A \in \mathcal{A}$, $\mathbf{f}(A) = (f_1(A), \ldots, f_k(A))'$. The optimization problem of multi-objective NAS is then formulated as $\min_{A \in \mathcal{A}} \mathbf{f}(A)$. There is no architecture that minimizes all objectives at the same time since these are typically in competition with each other. Rather, there are multiple Pareto optimal architectures reflecting different trade-offs in objectives approximating the true (unknown) Pareto front. An architecture A is said to dominate another architecture A' iff $\forall i \in \{1, \ldots, k\} : f_i(A) \leq f_i(A') \land \exists j \in \{1, \ldots, k\} : f_j(A) < f_j(A')$.

Constrained and Hardware-Aware Neural Architecture Search In contrast, *Constrained NAS* [62, 15, 55] solves the problem of finding an architecture that optimizes one objective (e.g., validation error) with constraints on secondary objectives (e.g., model size). Constraints can be naturally given by the target hardware that a model should be deployed on. *Hardware-Aware NAS* in turn searches for an architecture that trades off primary objectives [60] against secondary, hardware-specific metrics. In Once-for-All [5], a large supernet is trained which can be efficiently searched for subnets that, e.g., meet latency constraints of target devices. For a recent survey, we refer to [1].

Quality Diversity Optimization The goal of a QDO algorithm is to find a set of high-performing, yet behaviorally diverse, solutions. Similarly to multi-objective optimization, there is no single best solution. However, whereas multi-objective optimization aims for the simultaneous minimization of multiple objectives, QDO minimizes a single-objective function with respect to diversity defined on one or more *feature functions*. A feature function measures a quality of interest and a combination of feature values points to a niche, i.e., a region in *feature space*. QDO could be considered a *set* of constrained optimisation problems over the same input domain where the niche boundaries are constraints in feature space. The key difference is that constrained optimisation seeks a single optimal configuration given some constraints, while QDO attempts to identify the optimal configuration for each of a set of constrained regions simultaneously. In this sense, QDO could be framed as a so-called multi-task optimization problem [43] where each task is to find the best solution belonging to a particular niche.

QDO algorithms maintain an archive of niche-optimal observations, i.e., a best-performing observed solution for each niche. Observations with similar feature values compete to be selected for the archive, and the solution set gradually improves during the optimization process. Once the optimization budget has been spent, QDO algorithms typically return this archive as their solution. QDO is motivated by applications where a group of diverse solutions is beneficial, such as the training of robot movement where a repertoire of behaviours must be learned [7], developing game playing agents with diverse strategies [44], and in automatic design where QDO can be used by human designers to search a large dimensional search space for diverse solutions before the optimization is finished by hand. Work on automatic design tasks have been varied and include air-foil design [18], computer game level design [16], and architectural design [9]. Recently, QDO algorithms were used for illuminating the interpretability and resource usage of machine learning models while minimizing their generalization error [52].

In the earliest examples, Novelty Search (NS; [29]) asks whether diversity alone can produce a good set of solutions. Despite not actively pursuing objective performance, NS performed surprisingly well in some settings and was followed by Novelty Search with Local Competition [30], the first true quality diversity (QD) algorithm. MAP-Elites [42], a standard evolutionary QDO algorithm, partitions the feature space a-priori into niches and attempts to identify the optimal solution in each of these niches. QDO has seen much work in recent years and a variant based on BO, BOP-Elites, was proposed recently [25]. BOP-Elites models the objective and feature functions with surrogate models and implements an acquisition function over a structured archive to achieve high sample efficiency even in the case of black-box features.

3 Formulating Neural Architecture Search as a Quality Diversity Optimization Problem

In the example in Figure 1, a quality diversity NAS (subsequently abbreviated as qdNAS) problem is given by the validation error and three behavioral niches (corresponding to different devices) that are defined via resource usage measured by a single feature function. Let $f_1 : \mathcal{A} \rightarrow \mathbb{R}, A \mapsto f_1(A)$ denote the objective function of interest (in our context, f_{err}). Denote by $f_i : \mathcal{A} \rightarrow \mathbb{R}, A \mapsto f_i(A), i \in \{2, \ldots, k\}, k \geq 2$ the feature function(s) of interest (e.g., memory usage). Behavioral niches $N_j \subseteq \mathcal{A}, j \in \{1, \ldots, c\}, c \geq 1$ are sets of architectures characterized via niche-specific boundaries $\mathbf{b}_{ij} = \left[l_{ij}, u_{ij}\right) \subseteq \mathbb{R}$ on the images of the feature functions. An architecture A belongs to niche N_j if its values with respect to the feature functions lie between the respective boundaries, i.e.:

$$A \in N_j \iff \forall i \in \{2, \ldots, k\} : f_i(A) \in \mathbf{b}_{ij}.$$

The goal of a QDO algorithm is then to find for each behavioral niche N_j the architecture that minimizes the objective function f_1:

$$A_j^* := \arg\min_{A \in N_j} f_1(A).$$

In other words, the goal is to obtain a set of architectures $\mathcal{S} := \left\{A_1^*, \dots, A_c^*\right\}$ that are diverse with respect to the feature functions and yet high-performing.

A Remark about Niches In the classical QDO literature, niches are typically constructed to be pairwise disjoint, i.e., a configuration can only belong to a single niche (or none) [42, 25]. However, depending on the concrete application, relaxing this constraint and allowing for *overlap* can be beneficial. For example, in our context, an architecture that fits on a mid-tier device should also be considered for deployment on a higher-tier device, i.e., in Figure 1, the boundaries indicated by vertical dashed lines resemble the respective upper bound of a niche whereas the lower bound is unconstrained. This results in niches being nested within each other, i.e., $N_1 \subsetneq N_2 \subsetneq \ldots \subsetneq N_c \subseteq \mathcal{A}$, with N_1 being the most restrictive niche, followed by N_2. In Supplement A, we further discuss different ways of constructing niches in the context of NAS.

3.1 Quality Diversity Optimizers for Neural Architecture Search

As the majority of NAS optimizers are iterative, we first demonstrate how any iterative optimizer can in principle be turned into a QD optimizer. Based on this correspondence, we introduce three novel QD optimizers for NAS: BOP-Elites*, qdHB and BOP-ElitesHB. Let $f_1 : \mathcal{A} \to \mathbb{R}, A \mapsto f_1(x)$ denote the objective function that should be minimized. In each iteration, an iterative optimizer proposes a new configuration (e.g., architecture) for evaluation, evaluates this configuration, potentially updates the incumbent (best configuration evaluated so far) if better performance has been observed, and updates its archive. For generic pseudo code, see Supplement B.
Moving to a QDO problem, there are now feature functions $f_i : \mathcal{A} \to \mathbb{R}, A \mapsto f_i(A)$, $i \in \{2, \dots, k\}, k \geq 2$, and niches N_j, $j \in \{1, \dots, c\}, c \geq 1$, defined via their niche-specific boundaries $\mathbf{b}_{ij} = \left[l_{ij}, u_{ij}\right) \subseteq \mathbb{R}$ on the images of the feature functions. Any iterative single-objective optimizer must then keep track of the best incumbent per niche (often referred to as an *elite* in the QDO literature) and essentially becomes a QD optimizer (see Algorithm 1). The challenge

Algorithm 1: Generic pseudo code for an iterative quality diversity optimizer.

```
   Input  : $f_1, f_i, i \in \{2, \dots, k\}, k \geq 2, N_j, j \in \{1, \dots, c\}, c \geq 1, \mathcal{D}_{\text{design}}, n_{\text{total}}$
   Result : $S = \{A_1^*, \dots, A_c^*\}$
1  $\mathcal{D} \leftarrow \mathcal{D}_{\text{design}}$
2  for $j \leftarrow 1$ to $c$ do
3      $A_j^* \leftarrow \arg\min_{A \in \mathcal{D}_{|N_j}} f_1(A)$ # initial incumbent of niche $N_j$ based on archive
4  end
5  for $n \leftarrow 1$ to $n_{\text{total}}$ do
6      Propose a new candidate $A^\star$ # subroutine
7      Evaluate $y \leftarrow f_1(A^\star), \forall i \in \{2, \dots, k\} : z_i \leftarrow f_i(A^\star)$
8      if $A^\star \in N_j \land y < f_1(A_j^*)$ then
9          $A_j^* \leftarrow A^\star$ # update incumbent of niche $N_j$
10     end
11     $\mathcal{D} \leftarrow \mathcal{D} \cup \left\{\left(A^\star, y, z_2, \dots, z_k\right)\right\}$
12 end
```

in designing an efficient and well-performing QD optimizer now mostly lies in proposing a new candidate for evaluation that considers improvement over all niches.

Bayesian Optimization A recently proposed model-based QD optimizer, BOP-Elites [25], extends BO [54, 17] to QDO. BOP-Elites relies on Gaussian process surrogate models for the objective function and all feature functions. New candidates for evaluation are selected by a novel acquisition function – the expected joint improvement of elites (EJIE), which measures the expected improvement to the ensemble problem of identifying the best solution in every niche:

$$\alpha_{\text{EJIE}}(A) \coloneqq \sum_{j=1}^{c} \mathcal{P}(A \in N_j|\mathcal{D})\mathbb{E}_y \left[\mathbb{I}_{|N_j}(A)\right]. \tag{1}$$

Here, $\mathcal{P}(A \in N_j|\mathcal{D})$ is the posterior probability of A belonging to niche N_j, and $\mathbb{E}_y \left[\mathbb{I}_{|N_j}(A)\right]$ is the expected improvement (EI; [23]) with respect to niche N_j:

$$\mathbb{E}_y \left[\mathbb{I}_{|N_j}(A)\right] = \mathbb{E}_y \left[\max\left(f_{\min_{\mathrm{N_j}}} - y, 0\right)\right],$$

where $f_{\min_{N_j}}$ is the best observed objective function value in niche N_j so far, and y is the surrogate model prediction for A. A new candidate is then proposed by maximizing the EJIE, i.e., Line 6 in Algorithm 1 looks like the following: $A^\star \leftarrow \arg\max_{A \in \mathcal{A}} \alpha_{\text{EJIE}}(A)$.

BOP-Elites* In order to adapt BOP-Elites for NAS, we introduce several modifications. First, we employ truncated (one-hot) path encoding [56, 57]. In path encoding, architectures are transformed into a set of binary features indicating presence for each path of the directed acyclic graph from the input to the output. By then truncating the least-likely paths, the encoding scales linearly in the size of the cell [57] allowing for an efficient representation of architectures. Second, we substitute the Gaussian process surrogate models used in BOP-Elites with random forests [4] allowing us to model non-continuous hyperparameter spaces. Random forests have been successfully used as surrogates in BO [21, 33], often performing on a par with ensembles of neural networks [57] in the context of NAS [51, 59]. Third, we introduce a local mutation scheme similarly to the one used by the BANANAS algorithm [57] for optimizing the infill criterion EJIE: Since our aim is to find high quality solutions across all niches, we maintain an archive of the incumbent architecture in each niche and perform local mutations on each incumbent. We refer to our adjusted version as BOP-Elites* in the remainder of the paper to emphasize the difference from the original algorithm. For the initial design, we sample architectures based on adjacency matrix encoding [56].

Multifidelity Optimizers For NAS, performance estimation is the computationally most expensive component [13], and almost all NAS optimizers can be made more efficient by allowing access to cheaper, lower fidelity estimates [13, 31, 14, 64]. By evaluating most architectures at lower fidelity and only promoting promising architectures to higher fidelity, many more architectures can be explored given the same total computational budget. The fidelity parameter is typically the number of epochs over which an architecture is trained.

qdHB One of the most prominent multifidelity optimizers is Hyperband (HB; [31]), a multi-armed bandit strategy that uses repeated Successive Halving (SH; [22]) as a subroutine to identify the best configuration (e.g., architecture) among a set of randomly sampled ones. Given an initial and maximum fidelity, a scaling parameter η, and a set of configurations of size n, SH evaluates all configurations on the initial smallest fidelity, then sorts the configurations by performance and only keeps the best $1/\eta$ configurations. These configurations are then trained with fidelity increased by a factor of η. This process is repeated until the maximum fidelity for a single configuration is reached. HB repeatedly runs SH with different sized sets of initial configurations called brackets. Only two inputs are required: R, the maximum fidelity and η, the scaling parameter that controls the proportion of configurations discarded in each round of SH. Based on these inputs, the number $s_{\max}$ and size n_i of different brackets is determined. To adapt HB to the QD setting, we must track the incumbent architecture in each niche and promote configurations based on their performance within the respective niche (see Supplement B): To achieve this, we choose the top $\lfloor n_i/\eta \rfloor$ configurations to be promoted uniformly over the c niches (done in the $\texttt{top}_{\texttt{k\_qdo}}$ function), i.e., we iteratively select one of the niches uniformly at random and choose the best configuration observed so far that has yet not been selected for promotion until $\lfloor n_i/\eta \rfloor$ configurations have been selected. Note that during this procedure, it may happen that not enough configurations belonging to a specific niche have been observed yet. In this case, we choose any configuration uniformly at random over the set of all configurations that have yet to be promoted. With those modifications, we propose qdHB, as a multifidelity QD optimizer.

BOP-ElitesHB While HB typically shows strong anytime performance [31], it only samples configurations at random and is typically outperformed by BO methods with respect to final performance if optimizer runtime is sufficiently large [14]. BOHB [14] combines the strengths of HB and BO in a single optimizer, resulting in strong anytime performance and fast convergence. This approach employs a fidelity schedule similar to HB to determine how many configurations to evaluate at which fidelity but replaces the random selection of configurations in each HB iteration by a model-based proposal. In BOHB, a Tree Parzen Estimator [2] is used to model densities $l(A) = p(y < \alpha | A, \mathcal{D})$ and $g(A) = p(y > \alpha | A, \mathcal{D})$, and candidates are proposed that maximize the ratio $l(A)/g(A)$, which is equivalent to maximizing EI [2]. Based on BOP-Elites* and qdHB, we can now derive the QD Bayesian optimization Hyperband optimizer (BOP-ElitesHB): Instead of selecting configurations at random at the beginning of each qdHB iteration, we propose candidates that maximize the EJIE criterion. This sampling procedure is described in Supplement B.

4 Main Benchmark Experiments and Results

We are interested in answering the following research questions: (**RQ1**) *Does qdNAS outperform multi-objective NAS if the optimization goal is to find high-performing architectures in pre-defined niches?* (**RQ2**) *Do multifidelity qdNAS optimizers improve over full-fidelity qdNAS optimizers?* To answer these questions, we benchmark our three qdNAS optimizers – BOP-Elites*, qdHB, and BOP-ElitesHB– on the well-known NAS-Bench-101 [61] and NAS-Bench-201 [11] and compare them to three multi-objective optimizers adapted for NAS: ParEGO*, moHB*, and ParEGOHB as well as a simple Random Search[1].

Experimental Setup It is important to compare optimizers using analogous implementation details. We therefore use truncated path encoding and random forest surrogates throughout our experiments for all model-based optimizers. Furthermore, we use local mutations as described in [57] in order to optimize acquisition functions in BOP-Elites*, BOP-ElitesHB, ParEGO*, and ParEGOHB. To control for differences in implementation, we re-implement all optimizers and take great care in matching the original implementations.

We provide full details regarding implementation in Supplement B and only briefly introduce conceptual differences: ParEGO* is a multi-objective optimizer based on ParEGO [28] and only deviates from BOP-Elites* in that it considers a differently scalarized objective in each iteration, which is optimized using local mutations similar to the acquisition function optimization of BOP-Elites*. moHB* is an extension of HB to the multi-objective setting (promoting configurations based on non-dominated sorting with hypervolume contribution for tie breaking, for similar approaches see, e.g., [48, 49, 50, 19]). ParEGOHB is a model-based extension of moHB* that relies on the ParEGO scalarization [28] and on the same acquisition function optimization as ParEGO*.

All optimizers were evaluated on NAS-Bench-101 (Cifar-10, validation error as the first objective and the number of trainable parameters as the feature function/second objective) and NAS-Bench-201 (Cifar-10, Cifar-100, ImageNet16-120, validation error as the first objective and latency as the feature function/second objective). For multifidelity, we train architectures for $4, 12, 36, 108$ epochs on NAS-Bench-101 and for $2, 7, 22, 67, 200$ epochs on NAS-Bench-201 (reflecting $\eta = 3$ in the HB variants). As the optimization budget, we consider 200 full architecture evaluations (resulting in

[1]using adjacency matrix encoding [56]

a total budget of 21600 epochs for NAS-Bench-101 and 40000 epochs for NAS-Bench-201). For each of these four settings, we construct three different scenarios by considering different niches of interest with respect to the feature function, resulting in a total of 12 benchmark problems. In the *small/medium/large* settings, two, five and ten niches are considered, respectively. Niches are constructed to be overlapping, and boundaries are defined based on percentiles of the feature function. For the *small* setting, the boundary is given by the 50% percentile ($q_{50\%}$), effectively resulting in two niches with boundaries $[0, q_{50\%})$ and $[0, \infty)$. For the *medium* and *large* settings, percentiles indicating progressively larger niches were used, ranging from: 1% to 30% and 70% respectively. More details on the niches can be found in Supplement C. All runs were replicated 100 times.

Results As an anytime performance measure, we are interested in the validation error obtained for each niche, which we aggregate in a single performance measure as $\sum_{j=1}^{c} f_{\text{err}}(A_j^*)$, i.e., we consider the sum of validation errors over the best-performing architecture per niche. If a niche is empty, we assign a validation error of 100 as a penalty (this is common practice in QDO, i.e., if no solution has been found for a niche, this niche is assigned the worst possible objective function value [25]). For the final performance, we also consider the analogous test error. Figure 2 shows the anytime performance of optimizers. We observe that model-based optimizers (BOP-Elites* and ParEGO*) in general strongly outperform Random Search, and BO HB optimizers (BOP-ElitesHB and ParEGOHB) generally outperform their full-fidelity counterparts, although this effect diminishes with increasing optimization budget. In general, HB variants that do not rely on a surrogate model (qdHB and moHB*) show poor performance compared to the model-based optimizers. Moreover, especially in the *small* number of niches setting, QD strongly outperforms multi-objective optimization. Mean ranks of optimizers with respect to final validation and test performance are given in Table 1. For completeness, we also report critical differences plots of these ranks in Supplement C.

We also conducted two four-way ANOVAs on the average performance after half and all of the optimization budget is used, with the factors problem (benchmark problem), multifidelity (whether an optimizer uses multifidelity), QDO (whether an optimizer is a QD optimizer) and model-based (whether the optimizer relies on a surrogate model)[2]. For half the budget used, results indicate significant main effects of the factors multifidelity ($F(1) = 19.13, p = 0.0001$), QDO ($F(1) = 11.08, p = 0.0017$) and model-based ($F(1) = 21.13, p < 0.0001$). For all of the budget used, the significance of multifidelity diminishes, whereas the main effects of QDO ($F(1) = 18.31, p = 0.0001$) and model-based ($F(43.44), p < 0.0001$) are still significant. We can conclude that QDO in general outperforms competitors when the goal is to find high-performing architectures in pre-defined niches. Multi-fidelity optimizers improve over full-fidelity optimizers but this effect diminishes with increasing budget. Detailed results are reported in Supplement C.

Regarding efficiency, we analyzed the expected running time (ERT) of the QD optimizers given the average performance of the respective multi-objective optimizers after half of the optimization budget: For each benchmark problem, we computed the mean validation performance of each multi-objective optimizer after having spent half of its optimization budget and investigated the ERT of the analogous[3] QD optimizer. For each benchmark problem, we then computed the ratio of ERTs between multi-objective and QD optimizers and averaged them over the benchmark problems. For BOP-ElitesHB, we observe an average ERT ratio of 2.41, i.e., in expectation, BOP-ElitesHB is a factor of 2.41 faster than ParEGOHB in reaching the average performance of ParEGOHB (after half the optimization budget). For qdHB and BOP-Elites*, the average ERT ratios are 1.14 and 1.44. We conclude that all QD optimizers are more efficient than their multi-objective counterparts. More details can be found in Supplement C.

[2]For this analysis, we excluded qdHB and moHB* due to their lackluster performance.

[3]BOP-ElitesHB for ParEGOHB, qdHB for moHB*, and BOP-Elites* for ParEGO*

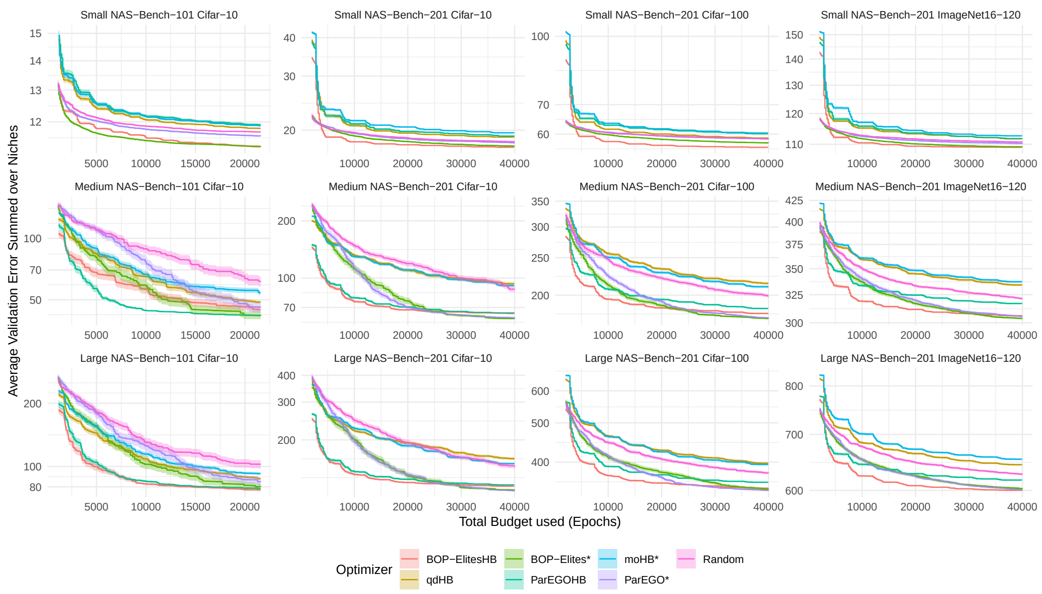


Figure 2: Anytime performance of optimizers. Ribbons represent standard errors over 100 replications. x-axis starts after 10 full-fidelity evaluations.

Table 1: Ranks of optimizers with respect to final performance, averaged over benchmark problems.

Mean Rank (SE)	BOP-ElitesHB	qdHB	BOP-Elites*	ParEGOHB	moHB*	ParEGO*	Random
Validation	2.08 (0.29)	5.92 (0.26)	1.83 (0.21)	4.25 (0.48)	6.42 (0.15)	2.58 (0.31)	4.92 (0.34)
Test	1.42 (0.19)	5.00 (0.17)	2.08 (0.23)	4.33 (0.50)	6.50 (0.19)	3.25 (0.35)	5.42 (0.42)

5 Additional Experiments and Applications

In this section, we illustrate how qdNAS can be used beyond the scenarios investigated so far and present results of additional experiments ranging from a comparison of qdNAS to multi-objective NAS on the `MobileNetV3` search space to an example on how to incorporate QDO in existing frameworks such as Once-for-All [5] or how to use qdNAS for model compression.

Benchmarks on the `MobileNetV3` Search Space We further investigated how qdNAS compares to multi-objective NAS on a search space that is frequently used in practice [20]. We consider CNNs divided into a sequence of units with feature map size gradually being reduced and channel numbers being increased. Each unit consists of a sequence of layers where only the first layer has stride 2 if the feature map size decreases and all other layers in the units have stride 1. Units can use an arbitrary number of layers (elastic depth chosen from $\{2, 3, 4\}$) and for each layer, an arbitrary number of channels (elastic width chosen from $\{3, 4, 6\}$) and kernel sizes (elastic kernel size chosen from $\{3, 5, 7\}$) can be used. Additionally, the input image size can be varied (elastic resolution ranging from 128 to 224 with a stride 4). For more details on the search space, see [5]. To allow for reasonable runtimes we use accuracy predictors (based on architectures trained and evaluated on ImageNet as described in [5]) and resource usage look-up tables of the Once-for-All module [5, 6] and construct a surrogate benchmark. As an objective function we select the validation error and as a feature function/second objective the latency (in ms) when deployed on a Samsung Note 10 (batch size of 1), or the number of FLOPS (M) used by the model. So far, we have only investigated qdNAS in the context of $k = 2$, i.e., considering one objective and one feature function. Here, we additionally consider a setting of $k = 3$, by incorporating both latency and the size of the model (in MB) as feature functions/second and third objective. We compare BOP-Elites* to ParEGO* and a

Random Search due to the accuracy predictors not supporting evaluations at multiple fidelities. We again construct three scenarios by considering different niches of interest with respect to the feature functions taking inspiration from latency and FLOPS constraints as used in [5] (details are given in Table 4 in Supplement C). Optimizers are given a total budget of 100 architecture evaluations. Figure 3 shows the anytime performance of optimizers with respect to the validation error summed over niches (averaged over 100 replications). BOP-Elites* strongly outperforms the competitors on all benchmark problems. More details are provided in Supplement D.

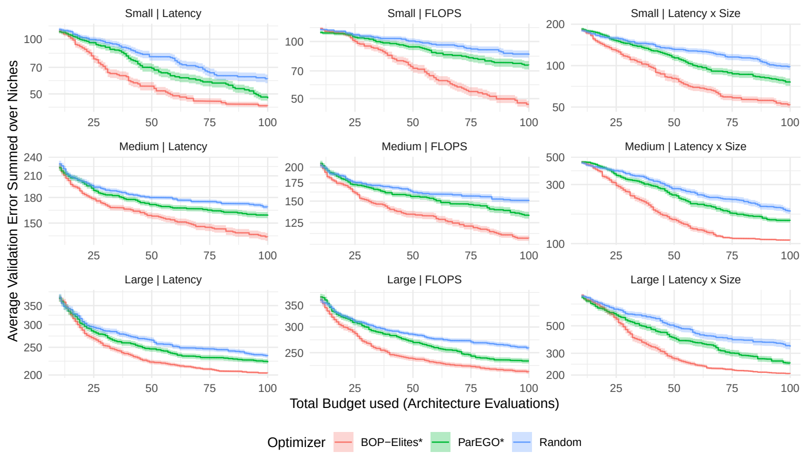


Figure 3: `MobileNetV3` search space. Anytime performance of optimizers. Ribbons represent standard errors over 100 replications. x-axis starts after 10 evaluations.

Making Once-for-All Even More Efficient In Once-for-All [5], an already trained supernet is searched via regularized evolution [46] for a well performing subnet that meets hardware requirements of a target device relying on an accuracy predictor and resource usage look-up tables. This is sensible if only a single solution is required, however, if various subnets meeting different constraints on the same device are desired, repeated regularized evolution is not efficient. Moreover, look-up tables do not generalize to new target devices in which case using as few as possible architecture evaluations suddenly becomes relevant again. We notice that the search for multiple architectures within Once-for-All can again be formulated as a QDO problem and therefore compare MAP-Elites [42] to regularized evolution when performing a joint search for architectures meeting different latency constraints on a Samsung Note 10. Results are given in Table 2 with MAP-Elites consistently outperforming regularized evolution, making this novel variant of Once-for-All even more efficient. More details are provided in Supplement E.

Table 2: MAP-Elites vs. regularized evolution within Once-for-All.

Method	Best Validation Error for Different Latency Constraints						
	$[0, 15)$	$[0, 18)$	$[0, 21)$	$[0, 24)$	$[0, 27)$	$[0, 30)$	$[0, 33)$
Reg. Evo.	**21.57** (0.01)	20.34 (0.02)	19.29 (0.01)	18.48 (0.02)	17.81 (0.02)	17.40 (0.02)	17.06 (0.02)
MAP-Elites	21.60 (0.01)	**20.28** (0.01)	**19.21** (0.01)	**18.39** (0.01)	**17.70** (0.01)	**17.25** (0.01)	**16.90** (0.01)

Average over 100 replications based on the accuracy predictor of Once-for-All [5, 6]. Standard errors in parentheses. Reg. Evo. = regularized evolution.

Applying qdNAS to Model Compression We are interested in deploying a `MobileNetV2` across different devices that are mainly constrained by memory. For each device, we can therefore only consider models up to a fixed amount of parameters, similarly as depicted in Figure 1. Given that we have a pretrained model that achieves high performance, we want to *compress* this model exploiting redundancies in model parameters. In our application, we use the Stanford Dogs dataset [26] and rely on the neural network intelligence (NNI; [40]) toolkit for model compression. Pruning consists of several (iterative) steps as well as re-training of the pruned architectures. Choices for the pruner itself, pruner hyperparameters, and hyperparameters controlling retraining are available and must be carefully selected to obtain optimal models (see Supplement F). We consider the number of model parameters as a proxy measure for memory requirement, yielding three overlapping niches for different devices. The pre-trained `MobileNetV2` achieves a validation error of 20.25 using around 2.34 million model parameters. We define niches with boundaries corresponding to compression rates (number of parameters after pruning) of 40% to 50%, 40% to 60%, and 40% to 70%. As the QD optimizer, we use BOP-ElitesHB and specify the number of fine-tuning epochs a pruner can use as a fidelity parameter, since fine-tuning after pruning is costly but also strongly influences final performance. After evaluating only 69 configurations (a single BOP-ElitesHB run with $\eta = 3$), we obtain high-performing pruner configurations for each niche, resulting in the performance vs. memory requirement (number of parameters) trade-offs shown in Table 3.

Table 3: Results of using BOP-ElitesHB for model compression of `MobileNetV2` on Stanford Dogs.

Niche	Validation Error	# Params (in millions; rounded)	% (# Params$_{\text{Baseline}}$)
Niche 1 $[0.94, 1.17)$	31.20	1.13	48.10%
Niche 2 $[0.94, 1.41)$	29.07	1.29	54.99%
Niche 3 $[0.94, 1.64)$	27.76	1.62	68.97%
Baseline	20.25	2.34	100.00%

6 Conclusion

We demonstrated how multi-objective NAS can be formulated as a QDO problem that, contrary to multi-objective NAS, directly corresponds to the actual optimization problem of interest, i.e., finding high-performing architectures in different niches. We have shown how any iterative black box optimization algorithm can be adapted to the QDO setting and proposed three QDO algorithms for NAS, with two of which making use of multifidelity evaluations. In benchmark experiments, we have shown that qdNAS outperforms multi-objective NAS while simultaneously being more efficient. We furthermore illustrated how qdNAS can be used for model compression and how future NAS research can thrive on QDO. QDO is orthogonal to the NAS strategy of an algorithm and can be similarly used to extend, e.g., one-shot NAS methods.

Limitations The framework we describe relies on pre-defined niches, e.g., memory requirements of different devices. If niches are mis-specified or cannot be specified a priori, multi-objective NAS may outperform qdNAS. However, an initial study (see Supplement H) how qdNAS performs in a true multi-objective setting, which would correspond to unknown niches, shows little to no performance degradation depending on the choice of niches. Moreover, we only investigated the performance of qdNAS in the deterministic setting. Additionally, our multifidelity optimizers require niche membership to be unaffected by the multifidelity parameter. Finally, we mainly focused on model-based NAS algorithms that we have extended to the QDO setting.

Broader Impact Our work extends previous research on NAS and therefore inherits its implications on society and individuals such as potential discrimination in resulting models. Moreover, evaluating a large number of architectures is computationally costly and can introduce serious environmental issues. We have shown that qdNAS allows for finding better solutions, while simultaneously being more efficient than multi-objective NAS. As performance estimation is extremely costly in NAS, we believe that this is an important contribution towards reducing resource usage and the CO_2 footprint of NAS.

7 Reproducibility Checklist

1. For all authors...

   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]

   (b) Did you describe the limitations of your work? [Yes] See Section 6.

   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See Section 6.

   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]

2. If you are including theoretical results...

   (a) Did you state the full set of assumptions of all theoretical results? [N/A]

   (b) Did you include complete proofs of all theoretical results? [N/A]

3. If you ran experiments...

   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results, including all requirements (e.g., `requirements.txt` with explicit version), an instructive `README` with installation, and execution commands (either in the supplemental material or as a URL)? [Yes] The full code for experiments, application, figures and tables can be obtained from the following GitHub repository: `https://github.com/slds-lmu/qdo_nas`.

   (b) Did you include the raw results of running the given instructions on the given code and data? [Yes] Raw results are provided via the same GitHub repository.

   (c) Did you include scripts and commands that can be used to generate the figures and tables in your paper based on the raw results of the code, data, and instructions given? [Yes] Scripts to generate figures and tables based on raw results are provided via the same GitHub repository.

   (d) Did you ensure sufficient code quality such that your code can be safely executed and the code is properly documented? [Yes]

   (e) Did you specify all the training details (e.g., data splits, pre-processing, search spaces, fixed hyperparameter settings, and how they were chosen)? [Yes] For our benchmark experiments we used NAS-Bench-101 and NAS-Bench-201. Regarding the Additional Experiments and Applications Section, all details are reported in Supplement D, Supplement E and Supplement F.

   (f) Did you ensure that you compared different methods (including your own) exactly on the same benchmarks, including the same datasets, search space, code for training and hyperparameters for that code? [Yes] As described in Section 4.

   (g) Did you run ablation studies to assess the impact of different components of your approach? [Yes] See Supplement C, Supplement G and Supplement H.

   (h) Did you use the same evaluation protocol for the methods being compared? [Yes] As described in Section 4.

   (i) Did you compare performance over time? [Yes] Anytime performance was assessed with respect to the number of epochs as described in Section 4 or the number of architecture evaluations as described in Section 5.

(j) Did you perform multiple runs of your experiments and report random seeds? [Yes] All runs of main benchmark experiments were replicated 100 times. Random seeds can be obtained via `https://github.com/slds-lmu/qdo_nas`.

(k) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] All results include error bars accompanying mean estimates.

(l) Did you use tabular or surrogate benchmarks for in-depth evaluations? [Yes] We used the tabular NAS-Bench-101 and NAS-Bench-201 benchmarks.

(m) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] As described in Supplement I.

(n) Did you report how you tuned hyperparameters, and what time and resources this required (if they were not automatically tuned by your AutoML method, e.g. in a NAS approach; and also hyperparameters of your own method)? [N/A] No tuning of hyperparameters was performed.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes] NAS-Bench-101, NAS-Bench-201, Naszilla, the Once-for-All module, NNI, and the Stanford Dogs dataset are cited appropriately.

(b) Did you mention the license of the assets? [Yes] Done in Supplement I.

(c) Did you include any new assets either in the supplemental material or as a URL? [Yes] We provide all our code via `https://github.com/slds-lmu/qdo_nas`.

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] All assets used are either released under the Apache-2.0 License or MIT License.

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Acknowledgements. The authors of this work take full responsibilities for its content. Lennart Schneider is supported by the Bavarian Ministry of Economic Affairs, Regional Development and Energy through the Center for Analytics - Data - Applications (ADACenter) within the framework of BAYERN DIGITAL II (20-3410-2-9-8). This work was supported by the German Federal Ministry of Education and Research (BMBF) under Grant No. 01IS18036A. Paul Kent acknowledges support from EPSRC under grant EP/L015374. This work has been carried out by making use of AI infrastructure hosted and operated by the Leibniz-Rechenzentrum (LRZ) der Bayerischen Akademie der Wissenschaften and funded by the German Federal Ministry of Education and Research (BMBF) under Grant No. 01IS18036A. The authors gratefully acknowledge the computational and data resources provided by the Leibniz Supercomputing Centre (`www.lrz.de`).

References

[1] H. Benmeziane, K. El Maghraoui, H. Ouarnoughi, S. Niar, M. Wistuba, and N. Wang. Hardware-aware neural architecture search: Survey and taxonomy. In Z.-H. Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 4322–4329, 2021.

[2] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*, volume 24, 2011.

[3] J. Bergstra, D. Yamins, and D. Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *Proceedings of the 30th International Conference on Machine Learning*, pages 115–123, 2013.

[4] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[5] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han. Once-for-All: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*, 2020.

[6] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han. Once-for-All: Train one network and specialize it for efficient deployment. `https://github.com/mit-han-lab/once-for-all`, 2020.

[7] A. Cully and J.-B. Mouret. Evolving a behavioral repertoire for a walking robot. *Evolutionary Computation*, 24(1):59–88, 2016.

[8] T. Domhan, J. T. Springenberg, and F. Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Proceedings of the 24th International Conference on Artificial Intelligence*, pages 3460–3468, 2015.

[9] S. Doncieux, N. Bredeche, L. L. Goff, B. Girard, A. Coninx, O. Sigaud, M. Khamassi, N. Díaz-Rodríguez, D. Filliat, T. Hospedales, A. Eiben, and R. Duro. Dream architecture: A developmental approach to open-ended learning in robotics. *arXiv:2005.06223 [cs.AI]*, 2020.

[10] J.-D. Dong, A.-C. Cheng, D.-C. Juan, W. Wei, and M. Sun. DPP-Net: Device-aware progressive search for pareto-optimal neural architectures. In *European Conference on Computer Vision*, 2018.

[11] X. Dong and Y. Yang. NAS-Bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations*, 2020.

[12] T. Elsken, J. H. Metzen, and F. Hutter. Efficient multi-objective neural architecture search via Lamarckian evolution. In *Proceedings of the International Conference on Learning Representations*, 2019.

[13] T. Elsken, J. H. Metzen, and F. Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019.

[14] S. Falkner, A. Klein, and F. Hutter. BOHB: Robust and efficient hyperparameter optimization at scale. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

[15] I. Fedorov, R. P. Adams, M. Mattina, and P. Whatmough. Sparse: Sparse architecture search for CNNs on resource-constrained microcontrollers. *Advances in Neural Information Processing Systems*, 32, 2019.

[16] M. C. Fontaine, R. Liu, J. Togelius, A. K. Hoover, and S. Nikolaidis. Illuminating mario scenes in the latent space of a generative adversarial network. In *AAAI Conference on Artificial Intelligence*, 2021.

[17] P. I. Frazier. A tutorial on Bayesian optimization. *arXiv:1807.02811 [stat.ML]*, 2018.

[18] A. Gaier, A. Asteroth, and J.-B. Mouret. Aerodynamic design exploration through surrogate-assisted illumination. In *18th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, 2017.

[19] J. Guerrero-Viu, S. Hauns, S. Izquierdo, G. Miotto, S. Schrodi, A. Biedenkapp, T. Elsken, D. Deng, M. Lindauer, and F. Hutter. Bag of baselines for multi-objective joint neural architecture search and hyperparameter optimization. In *8th ICML Workshop on Automated Machine Learning*, 2021.

[20] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam. Searching for MobileNetV3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1314–1324, 2019.

[21] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization*, pages 507–523, 2011.

[22] K. Jamieson and A. Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2015.

[23] D. R. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13(4):455–492, 1998.

[24] K. Kandasamy, W. Neiswanger, J. Schneider, B. Poczos, and E. Xing. Neural architecture search with Bayesian optimisation and optimal transport. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, 2018.

[25] P. Kent and J. Branke. BOP-Elites, a Bayesian optimisation algorithm for quality-diversity search. *arXiv:2005.04320 [math.OC]*, 2020.

[26] A. Khosla, N. Jayadevaprakash, B. Yao, and L. Fei-Fei. Novel dataset for fine-grained image categorization. In *First Workshop on Fine-Grained Visual Categorization (FGVC), IEEE Conference on Computer Vision and Pattern Recognition*, 2011.

[27] A. Klein, S. Falkner, J. T. Springenberg, and F. Hutter. Learning curve prediction with Bayesian neural networks. In *International Conference on Learning Representations*, 2017.

[28] J. Knowles. ParEGO: A hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. *IEEE Transactions on Evolutionary Computation*, 10(1):50–66, 2006.

[29] J. Lehman and K. O. Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, 19(2):189–223, 2011.

[30] J. Lehman and K. O. Stanley. Evolving a diversity of virtual creatures through novelty search and local competition. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, pages 211–218, 2011.

[31] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: Bandit-based configuration evaluation for hyperparameter optimization. In *International Conference on Learning Representations*, 2017.

[32] E. Liberis, Ł. Dudziak, and N. D. Lane. μnas: Constrained neural architecture search for microcontrollers. In *Proceedings of the 1st Workshop on Machine Learning and Systems*, pages 70–79, 2021.

[33] M. Lindauer, K. Eggensperger, M. Feurer, A. Biedenkapp, D. Deng, C. Benjamins, T. Ruhopf, R. Sass, and F. Hutter. SMAC3: A versatile Bayesian optimization package for hyperparameter optimization. *Journal of Machine Learning Research*, 23(54):1–9, 2022.

[34] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu. Hierarchical representations for efficient architecture search. In *Proceedings of the International Conference on Learning Representations*, 2017.

[35] H. Liu, K. Simonyan, and Y. Yang. DARTS: Differentiable architecture search. In *Proceedings of the International Conference on Learning Representations*, 2019.

[36] Z. Lu, K. Deb, E. Goodman, W. Banzhaf, and V. N. Boddeti. NSGANetV2: Evolutionary multi-objective surrogate-assisted neural architecture search. In A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, editors, *European Conference on Computer Vision*, pages 35–51, 2020.

[37] Z. Lu, I. Whalen, V. Boddeti, Y. Dhebar, K. Deb, E. Goodman, and W. Banzhaf. NSGA-Net: A multi-objective genetic algorithm for neural architecture search. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 419–427, 2019.

[38] B. Lyu, H. Yuan, L. Lu, and Y. Zhang. Resource-constrained neural architecture search on edge devices. *IEEE Transactions on Network Science and Engineering*, 9(1):134–142, 2021.

[39] H. Mendoza, A. Klein, M. Feurer, J. T. Springenberg, and F. Hutter. Towards automatically-tuned neural networks. In *ICML Workshop on Automatic Machine Learning*, 2016.

[40] Microsoft. Neural Network Intelligence. `https://github.com/microsoft/nni`, 2021.

[41] G. F. Miller, P. M. Todd, and S. U. Hegde. Designing neural networks using genetic algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 379–384, 1989.

[42] J.-B. Mouret and Jeff. Clune. Illuminating search spaces by mapping elites. *arXiv:1504.04909 [cs.AI]*, 2015.

[43] M. Pearce and J. Branke. Continuous multi-task Bayesian optimisation with correlation. *European Journal of Operational Research*, 270(3):1074–1085, 2018.

[44] D. Perez-Liebana, C. Guerrero-Romero, A. Dockhorn, D. Jeurissen, and L. Xu. Generating diverse and competitive play-styles for strategy games. In *2021 IEEE Conference on Games (CoG)*, pages 1–8, 2021.

[45] H. Pham, M. Guan, B. Zoph, Q. V. Le, and J. Dean. Efficient neural architecture search via parameter sharing. In *Proceedings of the 35th International Conference on Machine Learning*, pages 4095–4104, 2018.

[46] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 4780–4789, 2019.

[47] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin. Large-scale evolution of image classifiers. In *Proceedings of the 34th International Conference on Machine Learning*, pages 2902–2911, 2017.

[48] D. Salinas, V. Perrone, O. Cruchant, and C. Archambeau. A multi-objective perspective on jointly tuning hardware and hyperparameters. In *2nd Workshop on Neural Architecture Search at ICLR 2021*, 2021.

[49] R. Schmucker, M. Donini, V. Perrone, M. B. Zafar, and C. Archambeau. Multi-objective multi-fidelity hyperparameter optimization with application to fairness. In *NeurIPS Workshop on Meta-Learning*, 2020.

[50] R. Schmucker, M. Donini, M. B. Zafar, D. Salinas, and C. Archambeau. Multi-objective asynchronous successive halving. *arXiv:2106.12639 [stat.ML]*, 2021.

[51] L. Schneider, F. Pfisterer, M. Binder, and B. Bischl. Mutation is all you need. In *8th ICML Workshop on Automated Machine Learning*, 2021.

[52] L. Schneider, F. Pfisterer, J. Thomas, and B. Bischl. A collection of quality diversity optimization problems derived from hyperparameter optimization of machine learning models. *arXiv:2204.14061 [cs.LG]*, 2022.

[53] C. Schorn, T. Elsken, S. Vogel, A. Runge, A. Guntoro, and G. Ascheid. Automated design of error-resilient and hardware-efficient deep neural networks. *Neural Computing and Applications*, 32:18327–18345, 2020.

[54] J. Snoek, H. Larochelle, and R. P. Adams. Practical Bayesian optimization of machine learning algorithms. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25, 2012.

[55] D. Stamoulis, R. Ding, D. Wang, D. Lymberopoulos, B. Priyantha, J. Liu, and D. Marculescu. Single-Path NAS: Designing hardware-efficient ConvNets in less than 4 hours. In U. Brefeld, E. Fromont, A. Hotho, A. Knobbe, M. Maathuis, and C. Robardet, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 481–497, 2020.

[56] C. White, W. Neiswanger, S. Nolen, and Y. Savani. A study on encodings for neural architecture search. In *Advances in Neural Information Processing Systems*, 2020.

[57] C. White, W. Neiswanger, and Y. Savani. BANANAS: Bayesian optimization with neural architectures for neural architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021.

[58] C. White, S. Nolen, and Y. Savani. Exploring the loss landscape in neural architecture search. In *Uncertainty in Artificial Intelligence*, pages 654–664, 2021.

[59] C. White, A. Zela, B. Ru, Y. Liu, and F. Hutter. How powerful are performance predictors in neural architecture search? In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, 2021.

[60] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer. FBNet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10734–10742, 2019.

[61] C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter. NAS-Bench-101: Towards reproducible neural architecture search. In *Proceedings of the 36th International Conference on Machine Learning*, pages 7105–7114, 2019.

[62] Y. Zhou, S. Ebrahimi, S. Ö. Arık, H. Yu, H. Liu, and G. Diamos. Resource-efficient neural architect. *arXiv:1806.07912 [cs.NE]*, 2018.

[63] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. In *Proceedings of the International Conference on Learning Representations*, 2017.

[64] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8697–8710, 2018.

A Niches in NAS

In the classical QDO literature, niches are assumed to be pairwise disjoint. This implies that each architecture $A \in \mathcal{A}$ yields feature function values $f_i(A), i \geq 2$ that map to a single niche (or none). In practice, this does not necessarily have to be the case though, as an architecture can belong to multiple niches. For example, when considering memory or latency constraints, a model with lower latency or lower memory requirements can always be used in settings that allow for accommodating slower or larger models. This is illustrated in Figure 4. Note that we index niches in the disjoint scenario in Figure 4 with two indices, to highlight that some niches share the same boundaries on a given feature function (e.g., $N_{1,1}$ and $N_{2,1}$ share the same latency boundaries and only differ with respect to the memory boundaries). In this paper, we mainly investigated the scenario of nested niches. The setting for QDO in the NAS context as described in Figure 1 in the main paper is given by the search of models for deployment on multiple different end-user devices. Similarly, qdNAS can also be applied in the context of searching for models for deployment on a single end-user device, meeting different constraints, e.g., as illustrated in Section 5 (Benchmarks on the `MobileNetV3` Search Space) in the main paper. Typically, relevant boundaries of feature functions that form niches naturally arise given the target device(s) and concrete application at hand.

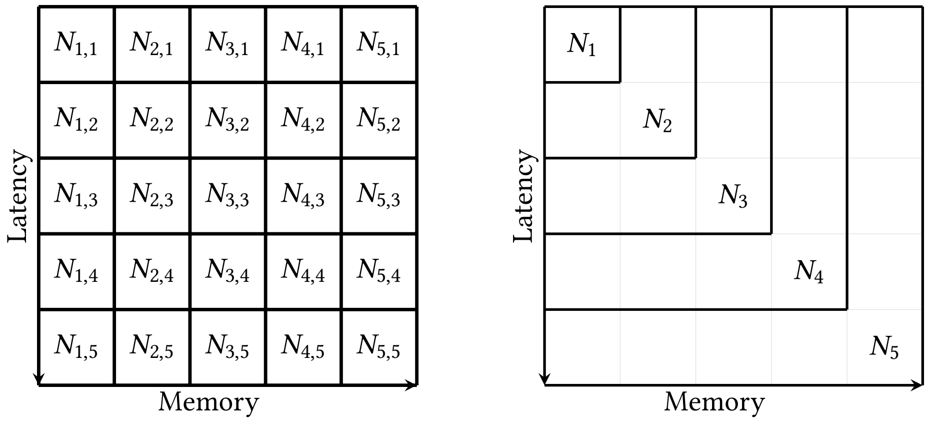

$N_{1,1}$ $N_{2,1}$ $N_{3,1}$ $N_{4,1}$ $N_{5,1}$
$N_{1,2}$ $N_{2,2}$ $N_{3,2}$ $N_{4,2}$ $N_{5,2}$
$N_{1,3}$ $N_{2,3}$ $N_{3,3}$ $N_{4,3}$ $N_{5,3}$
$N_{1,4}$ $N_{2,4}$ $N_{3,4}$ $N_{4,4}$ $N_{5,4}$
$N_{1,5}$ $N_{2,5}$ $N_{3,5}$ $N_{4,5}$ $N_{5,5}$
Latency
Memory
N_1
N_2
N_3
N_4
N_5
Latency
Memory

Figure 4: Disjoint (left) and nested (right) niches.

B Optimizers

In this section, we provide additional information on optimizers used throughout this paper. Algorithm 2 illustrates a generic iterative single-objective optimizer in pseudo code.

qdHB Algorithm 3 presents qdHB in pseudo code. qdHB requires only R (maximum fidelity) and η (scaling parameter) as input parameters and proceeds to determine the maximum number of brackets $s_{\max}$ and the approximate total resources B which each bracket is assigned. In each bracket s, the number of configurations n and the fidelity r at which they should be evaluated is calculated and these parameters are used within the SH subroutine. The central step within the SH subroutine is the selection of the $\lfloor n_i/\eta \rfloor$ configurations that should be promoted to the next stage. Here, the $\texttt{top}_{\text{k\_qdo}}$ function (highlighted in grey) works as follows: We iteratively select one of the niches uniformly at random and choose the best configuration within this niche observed so far that has yet not been selected for promotion. This procedure is repeated until $\lfloor n_i/\eta \rfloor$ configurations have been selected in total. If not enough configurations belonging to a specific niche have been observed so far, we choose any configuration uniformly at random over the set of all configurations that have yet to be promoted. Note that feature functions and thereupon derived niche membership are assumed to be unaffected by the multifidelity parameter. Niche membership is determined by the

Algorithm 2: Generic pseudo code for an iterative single-objective optimizer.

```
Input  : $f_1$, $\mathcal{D}_{\text{design}}$, $n_{\text{total}}$
Result : $A^*$
1  $\mathcal{D} \leftarrow \mathcal{D}_{\text{design}}$
2  $A^* \leftarrow \arg\min_{A \in \mathcal{D}} f_1(A)$ # initial incumbent based on archive
3  for $n \leftarrow 1$ to $n_{\text{total}}$ do
4      Propose a new candidate $A^\star$
5      Evaluate $y \leftarrow f_1(A^\star)$
6      if $y < f_1(A^*)$ then
7          $A^* \leftarrow A^\star$ # update incumbent
8      end
9      $\mathcal{D} \leftarrow \mathcal{D} \cup \{(A^\star, y)\}$
10 end
```

`get_niche_membership` function which simply checks for each niche whether feature values of an architecture are within the respective niche boundaries. Moreover, we assume that all evaluations are written into an archive similarly as in Algorithm 1 in the main paper which allows us to return the best configuration per niche as the final result. Note that in practice, evaluating all stages of brackets with the same budget instead of iterating over brackets (like in the original HB implementation) can be more efficient. We use this scheduling variant throughout our benchmark experiments and application study. More details regarding our implementation can be obtained via `https://github.com/slds-lmu/qdo_nas`.

Algorithm 3: Quality Diversity Hyperband (qdHB).

```
Input  : $R, \eta$ # maximum fidelity and scaling parameter
Result : Best configuration per niche
1  $s_{\max} = \lfloor \log_\eta(R) \rfloor, B = (s_{\max} + 1)R$
2  for $s \in \{s_{\max}, s_{\max} - 1, \ldots, 0\}$ do
3      $n = \lceil \frac{B}{R} \frac{\eta^s}{(s+1)} \rceil, r = R\eta^{-s}$
4      # begin SH with $(n, r)$ inner loop
5      $\mathbf{A}$ = sample_configuration($n$)
6      $\mathbf{Z} = \{(f_i(A), \ldots, f_k(A)) : A \in \mathbf{A}, i \in \{2, \ldots, k\}\}$ # evaluate feature functions
7      $\mathbf{N}$ = get_niche_membership($\mathbf{A}, \mathbf{Z}$)
8      for $i \in \{0, \ldots, s\}$ do
9          $n_i = \lfloor n\eta^{-i} \rfloor$
10         $r_i = r\eta^i$
11         $\mathbf{Y} = \{f_1(A, r_i) : A \in \mathbf{A}\}$ # evaluate objective function
12         $\mathbf{A}$ = top$_{\text{k\_qdo}}$($\mathbf{A}, \mathbf{Y}, \mathbf{N}, \lfloor n_i/\eta \rfloor$)
13     end
14 end
```

BOP-ElitesHB In Algorithm 4 we describe the sampling procedure (for a single configuration) used in BOP-ElitesHB in pseudo code. In contrast to the original BOHB algorithm, we use random forest as surrogate models, similarly as done in SMAC-HB [33]. Throughout our benchmark experiments and application study we set $\rho = 0$. Furthermore, we employ a variant that directly proposes batches of size n. This can be done by simply sorting all candidate architectures obtained via local mutation of the incumbent architectures of each niche within the acquisition function optimization step by their EJIE values and selecting the top n candidate architectures. Note that surrogate models are fitted on all available data contained in the current archive (this includes the multifidelity parameter) and predictions are obtained with respect to the fidelity parameter set to the current fidelity level. More details regarding our implementation can be obtained via `https://github.com/slds-lmu/qdo_nas`.

Algorithm 4: Sampling procedure in BOP-ElitesHB.

```
    Input  : $\rho$ # fraction of configurations sampled at random
    Result : Next configuration to evaluate
1   if rand() < $\rho$ then
2       return sample_configuration(1)
3   else
4       $A^\star \leftarrow \arg\max_{A \in \mathcal{A}} \alpha_{\mathrm{EJIE}}(A)$ # Equation (1)
5       return $A^\star$
6   end
```

ParEGO* ParEGO [28] is a multi-objective model-based optimizer that at each iteration scalarizes the objective functions differently using the augmented Tchebycheff function. First, the k objectives are normalized and at each iteration a weight vector λ is drawn uniformly at random from the following set of $\binom{s+k-1}{k-1}$ different weight vectors[4]:

$$\left\{\lambda = (\lambda_1, \lambda_2, \dots, \lambda_k) \mid \sum_{i=1}^{k} \lambda_i = 1 \,\wedge\, \lambda_i = \frac{l}{s}, l \in \{0, \dots, s\}\right\}.$$

The scalarization is then obtained via $f_\lambda(A) = \max_{i=1}^{k} (\lambda_i \cdot f_i(A)) + \gamma \sum_{i=1}^{k} \lambda_i \cdot f_i(A)$, where γ is a small positive value (in our benchmark experiments we use 0.05). In ParEGO* we use the same truncated path encoding as in BOP-Elites* as well as a random forest surrogate modeling the scalarized objective function. For optimizing the EI, we use a local mutation scheme similarly to the one utilized by BANANAS [57], adapted for the multi-objective setting (conceptually similar to the one proposed by [19]): For each Pareto optimal architecture in the current archive, we obtain candidate architectures via local mutation and out of all these candidates we select the architecture with the largest EI for evaluation. More details regarding our implementation can be obtained via `https://github.com/slds-lmu/qdo_nas`.

moHB* moHB* [28] is an extension of HB to the multi-objective setting. The optimizer follows the basic HB routine except for the selection mechanism of configurations that should be promoted to the next stage: Configurations are promoted based on non-dominated sorting with hypervolume contribution for tie breaking. For similar approaches, see [48, 49, 50, 19]. In our benchmark experiments we again use a scheduling variant that evaluates all stages of brackets with the same budget instead of iterating over brackets. More details regarding our implementation can be obtained via `https://github.com/slds-lmu/qdo_nas`.

ParEGOHB ParEGOHB combines BO with moHB* by using the same scalarization as ParEGO*. Instead of selecting configurations at random at the beginning of each moHB* iteration, ParEGOHB proposes candidates that maximize the EI with respect to the scalarized objective. In our benchmark experiments we again set $\rho = 0$ (fraction of configurations sampled uniformly at random) and employ a variant that directly proposes batches of size n. Note that surrogate models are fitted on all available data contained in the current archive (this includes the multifidelity parameter) and predictions are obtained with respect to the fidelity parameter set to the current fidelity level. More details regarding our implementation can be obtained via `https://github.com/slds-lmu/qdo_nas`.

[4]note that s simply determines the number of different weight vectors

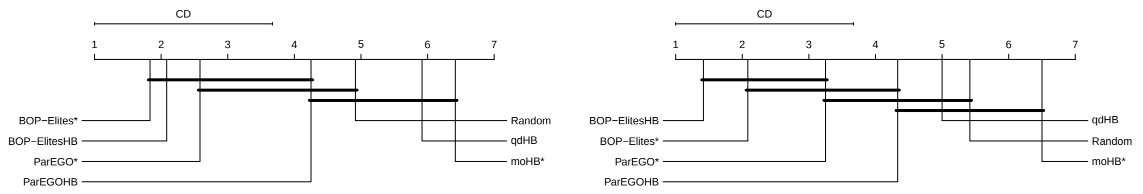
Table 4: Niches and their boundaries used throughout all benchmark experiments.

Benchmark	**Dataset**	**Niches**	**Niche Boundaries**									
			Niche 1	Niche 2	Niche 3	Niche 4	Niche 5	Niche 6	Niche 7	Niche 8	Niche 9	Niche 10
NAS-Bench-101 # Params	Cifar-10	Small	[0, 5356682)	$[0, \infty)$	-	-	-	-	-	-	-	-
		Medium	[0, 650520)	[0, 1227914)	[0, 1664778)	[0, 3468426)	$[0, \infty)$	-	-	-	-	-
		Large	[0, 650520)	[0, 824848)	[0, 1227914)	[0, 1664778)	[0, 2538506)	[0, 3468426)	[0, 3989898)	[0, 5356682)	[0, 8118666)	$[0, \infty)$
NAS-Bench-201 Latency	Cifar-10	Small	[0, 0.015000444871408)	$[0, \infty)$	-	-	-	-	-	-	-	-
		Medium	[0, 0.00856115)	[0, 0.01030767)	[0, 0.01143533)	[0, 0.01363741)	$[0, \infty)$	-	-	-	-	-
		Large	[0, 0.00856115)	[0, 0.00893427)	[0, 0.01030767)	[0, 0.01143533)	[0, 0.01250159)	[0, 0.01363741)	[0, 0.01429903)	[0, 0.01500044)	[0, 0.01660615)	$[0, \infty)$
	Cifar-100	Small	[0, 0.0159673188862048)	$[0, \infty)$	-	-	-	-	-	-	-	-
		Medium	[0, 0.00919228)	[0, 0.01138714)	[0, 0.01232998)	[0, 0.01475572)	$[0, \infty)$	-	-	-	-	-
		Large	[0, 0.00919228)	[0, 0.00957457)	[0, 0.01138714)	[0, 0.01232998)	[0, 0.01327515)	[0, 0.01475572)	[0, 0.01534633)	[0, 0.01596732)	[0, 0.01768237)	$[0, \infty)$
	ImageNet16-120	Small	[0, 0.014301609992981)	$[0, \infty)$	-	-	-	-	-	-	-	-
		Medium	[0, 0.00767465)	[0, 0.0094483)	[0, 0.01054566)	[0, 0.01271056)	$[0, \infty)$	-	-	-	-	-
		Large	[0, 0.00767465)	[0, 0.00826192)	[0, 0.0094483)	[0, 0.01054566)	[0, 0.01173623)	[0, 0.01271056)	[0, 0.01352221)	[0, 0.01430161)	[0, 0.01595311)	$[0, \infty)$
MobileNetV3 Latency	ImageNet	Small	[0, 17.5)	[0, 30)	-	-	-	-	-	-	-	-
		Medium	[0, 15)	[0, 20)	[0, 25)	[0, 30)	[0, 35)	-	-	-	-	-
		Large	[0, 17)	[0, 19)	[0, 21)	[0, 23)	[0, 25)	[0, 27)	[0, 29)	[0, 31)	[0, 33)	[0, 35)
MobileNetV3 FLOPS	ImageNet	Small	[0, 150)	[0, 400)	-	-	-	-	-	-	-	-
		Medium	[0, 150)	[0, 200)	[0, 250)	[0, 300)	[0, 400)	-	-	-	-	-
		Large	[0, 150)	[0, 175)	[0, 200)	[0, 225)	[0, 250)	[0, 275)	[0, 300)	[0, 325)	[0, 350)	[0, 400)
MobileNetV3 Latency $\times$ Size	ImageNet	Small	$[0, 20) \times [0, 20)$	$[0, 35) \times [0, 20)$	-	-	-	-	-	-	-	-
		Medium	$[0, 20) \times [0, 20)$	$[0, 25) \times [0, 20)$	$[0, 30) \times [0, 20)$	$[0, 35) \times [0, 20)$	$[0, 40) \times [0, 20)$	-	-	-	-	-
		Large	$[0, 20) \times [0, 20)$	$[0, 23) \times [0, 20)$	$[0, 26) \times [0, 20)$	$[0, 29) \times [0, 20)$	$[0, 32) \times [0, 20)$	$[0, 35) \times [0, 20)$	$[0, 38) \times [0, 20)$	$[0, 41) \times [0, 20)$	$[0, 44) \times [0, 20)$	$[0, 47) \times [0, 20)$

C Additional Benchmark Details and Results

In this section, we provide additional details and analyses with respect to our main benchmark experiments. Table 4 summarizes all niches and their boundaries used throughout our benchmarks (including the additional ones on the `MobileNetV3` search space).

The following results extends the results reported for the main benchmark experiments. Critical differences plots ($\alpha = 0.05$) of optimizer ranks (with respect to final performance) are given in Figure 5. Friedman tests ($\alpha = 0.05$) that were conducted beforehand indicated significant differences in ranks for both the validation ($\chi^2(6) = 53.46, p < 0.001$) and test performance ($\chi^2(6) = 52.14, p < 0.001$). However, note that critical difference plots based on the Nemenyi test are underpowered if only few optimizers are compared on few benchmark problems.



(a) Validation error summed over niches.

(b) Test error summed over niches.

Figure 5: Critical differences plots of the ranks of optimizers.

Figure 6 and Figure 7 show the average best validation and test performance for each niche for each optimizer on each benchmark problem.

Table 5 summarizes results of a four way ANOVA on the average performance (validation error summed over niches) of BOP-ElitesHB, BOP-Elites*, ParEGOHB, ParEGO* and Random after having used half of the total optimization budget. Prior to conducting the ANOVA, we checked the ANOVA assumptions (normal distribution of residuals and homogeneity of variances) and found no violation of assumptions. The factors are given as follows: Problem indicates the benchmark problem (e.g., NAS-Bench-101 on Cifar-10 with small number of niches), multifidelity denotes if the optimizer uses multifidelity (`TRUE` for BOP-ElitesHB and ParEGOHB), QDO denotes whether the optimizer is a QD optimizer (`TRUE` for BOP-ElitesHB and BOP-Elites*) and model-based denotes whether the optimizer relies on a surrogate model (`TRUE` for BOP-ElitesHB, BOP-Elites*, ParEGOHB and ParEGO*). All main effects are significant at an α level of 0.05. We also computed confidence intervals based on Tukey's Honest Significant Difference method for the estimated differences between factor levels: Multifidelity $-12.45[-18.19 - 6.72]$, QDO $-9.34[-15.08, -3.61]$, model-based $-13.55[-20.57, -6.52]$. Note that the negative sign indicates a decrease in the average validation error summed over niches.

Table 5: Results of a four way ANOVA on the average performance (validation error summed over niches) after having used half of the total optimization budget. Type II sums of squares.

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Problem	11	1810569.79	164597.25	1410.16	0.0000
Multifidelity	1	2233.44	2233.44	19.13	0.0001
QDO	1	1293.41	1293.41	11.08	0.0017
Model-Based	1	2466.45	2466.45	21.13	0.0000
Residuals	45	5252.51	116.72		

We conducted a similar ANOVA on the final performance of optimizers (Table 6). Prior to conducting the ANOVA, we checked the ANOVA assumptions (normal distribution of residuals and homogeneity of variances) and found no violation of assumptions. While the effects of QDO

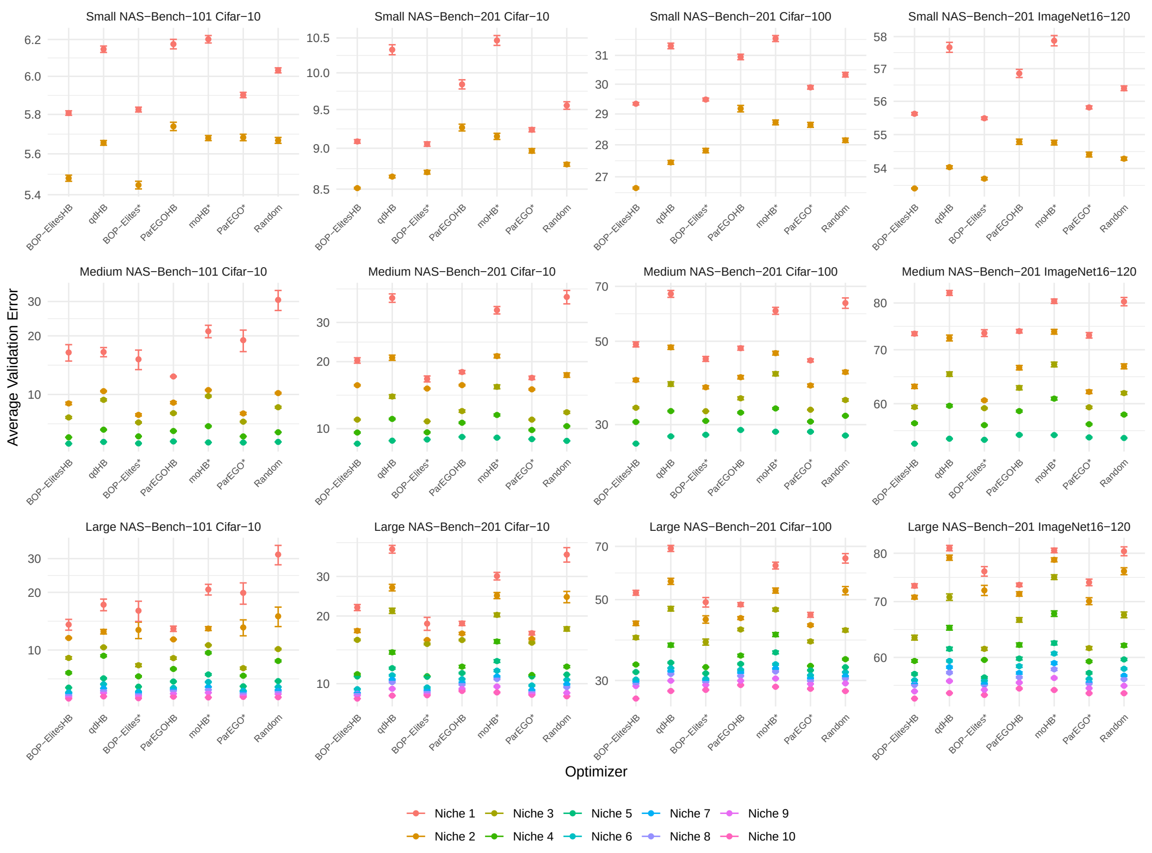


Figure 6: Best solution found in each niche with respect to validation performance. Bars represent standard errors over 100 replications.

and model-based are still significant at an α level of 0.05, the effect of multifidelity no longer is, indicating that full-fidelity optimizer caught up in performance (which is the expected behavior). We again computed confidence intervals based on Tukey's Honest Significant Difference method for the estimated differences between factor levels: QDO $-6.85[-10.12 - 3.58]$, model-based $-11.08[-15.08, -7.07]$.

Table 6: Results of a four way ANOVA on the average final performance (validation error summed over niches). Type II sums of squares.

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Problem	11	1724557.85	156777.99	4130.33	0.0000
Multifidelity	1	97.76	97.76	2.58	0.1155
QDO	1	695.13	695.13	18.31	0.0001
Model-Based	1	1648.94	1648.94	43.44	0.0000
Residuals	45	1708.10	37.96		

We analyzed the ERT of the QD optimizers given the average performance of the respective multi-objective optimizers after half of the optimization budget. For each benchmark problem, we computed the mean validation performance of each multi-objective optimizer after having spent half of its optimization budget and investigated the analogous QD optimizer. We then computed the ratio of ERTs between multi-objective and QD optimizers (see Table 7).

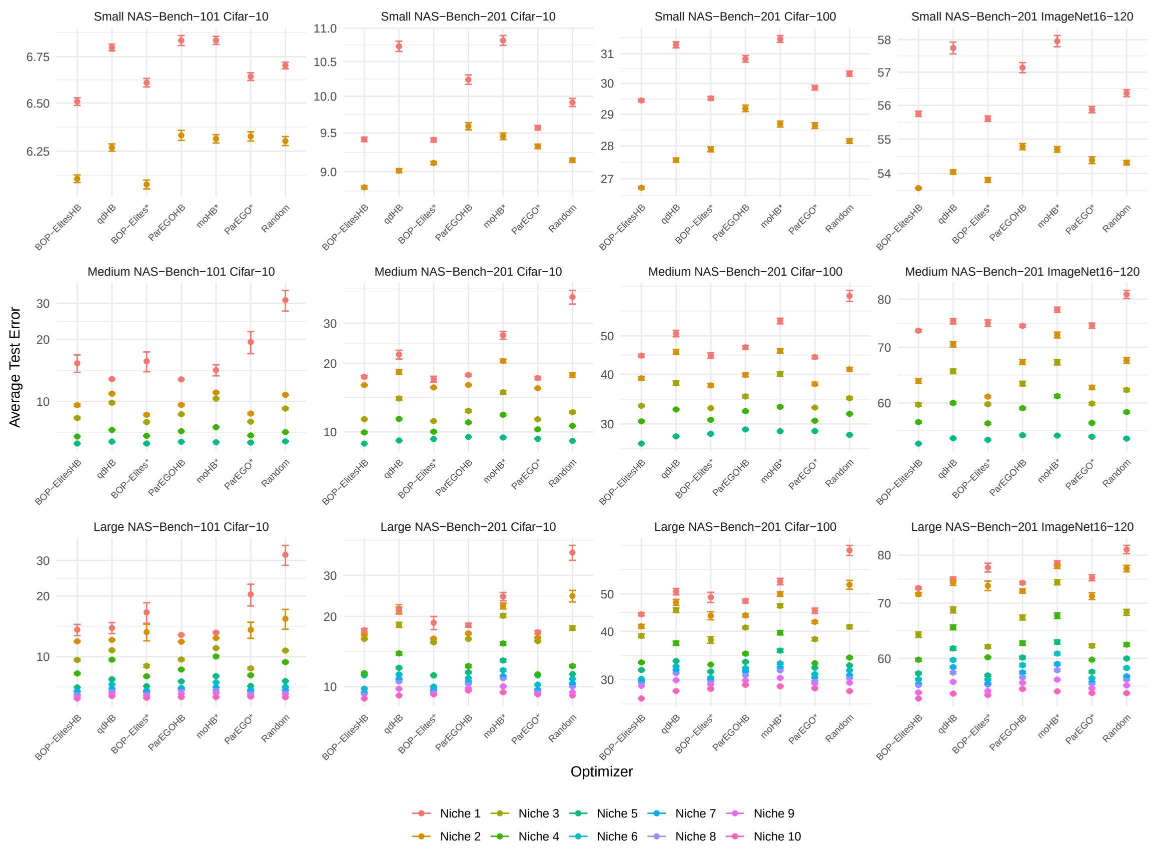


Figure 7: Best solution found in each niche with respect to test performance. Bars represent standard errors over 100 replications.

D Details on Benchmarks on the `MobileNetV3` Search Space

In this section, we provide additional details regarding our benchmarks on the `MobileNetV3` Search Space. We use `ofa_mbv3_d234_e346_k357_w1.2` as a pretrained supernet and rely on accuracy predictors and latency/FLOPS look-up tables as provided by [6]. The search space of architectures is the same as used in [5]. For the model-based optimizers we employ the following encoding of architectures: Given an architecture, we encode each layer in the neural network into a one-hot vector based on its kernel size and expand ratio and we assign zero vectors to layers that are skipped. Besides, we have an additional one-hot vector that represents the input image size. We concatenate these vectors into a large vector that represents the whole neural network architecture and input image size. This is the same encoding as used by [5]. Acquisition function optimization is performed by sampling 1000 architectures uniformly at random.

E Details on Making Once-for-All Even More Efficient

In this section, we provide additional details regarding replacing regularized evolution with MAP-Elites within Once-for-All. We use `ofa_mbv3_d234_e346_k357_w1.2` as a pretrained supernet and rely on accuracy predictors and latency look-up tables as provided by [6]. Seven niches were defined via the following latency constraints (in ms): $[0, 15)$, $[0, 18)$, $[0, 21)$, $[0, 24)$, $[0, 27)$, $[0, 30)$, $[0, 33)$. Regularized evolution is run with an initial population of size 100 for 71 generations [5] resulting in

[5]this is exactly $\lceil(50100 - 7 \cdot 100)/(7 * 100)\rceil$ with 50100 being the budget MAP-Elites is allowed to use

Table 7: ERT ratios of multi-objective and QD optimizers to reach the average performance (after half of the optimization budget) of the respective multi-objective optimizer.

Benchmark	Dataset	Niches	ERT Ratio		
			ParEGOHB/ BOP-ElitesHB	moHB*/ qdHB	ParEGO*/ BOP-Elites*
NAS-Bench-101	Cifar-10	Small	3.94	1.19	1.99
		Medium	0.76	1.43	1.85
		Large	1.47	1.20	1.58
NAS-Bench-201	Cifar-10	Small	4.31	1.96	1.45
		Medium	0.94	0.73	1.34
		Large	1.04	0.72	1.35
	Cifar-100	Small	4.82	1.77	1.46
		Medium	1.35	0.67	1.42
		Large	1.57	0.78	0.98
	ImageNet16-120	Small	4.30	1.20	1.73
		Medium	2.31	0.93	1.14
		Large	2.11	1.15	0.96

7200 architecture evaluations per latency constraint and 50400 architecture evaluations in total. We use a mutation probability of 0.1, a mutation ratio of 0.5 and a parent ratio of 0.25. MAP-Elites searches for optimal architectures jointly for the seven niches and is configured to use a population of size 100 and is run for 500 generations, resulting in 50100 architecture evaluations in total. The number of generations for each regularized evolution run and the MAP-Elites run were chosen in a way so that the total number of architecture evaluations is roughly the same for both methods. We again use a mutation probability of 0.1. Note that the basic MAP-Elites (as used by us) does not employ any kind of crossover. We visualize the best validation error obtained for each niche in Figure 8 (left). MAP-Elites outperforms regularized evolution in almost every niche, making this variant of Once-for-All even more efficient. In the scenario of using Once-for-All for new devices, look-up tables do not generalize and the need for using as few as possible architecture evaluations is of central importance. To illustrate how MAP-Elites compares to regularized evolution in this scenario, we reran the experiments above but this time we used a population of size 50 and 100 generations for MAP-Elites (and therefore 14 generations for each run of regularized evolution). Results are illustrated in Figure 8 (right). Again, MAP-Elites generally outperforms regularized evolution.

F Details on Applying qdNAS to Model Compression

In this section, we provide additional details regarding our application of qdNAS to model compression. BOP-ElitesHB was slightly modified due to the natural tabular representation of the search space. Instead of using a truncated path encoding we simply use the tabular representation of parameters. To optimize the EJIE during the acquisition function optimization step we employ a simple Random Search, sampling 10000 configurations uniformly at random and proposing the configuration with the largest EJIE. Table 8 shows the search space used for tuning NNI pruners on `MobileNetV2`.

G Analyzing the Effect of the Choice of the Surrogate Model and Acquisition Function Optimizer

In this section, we present results of a small ablation study regarding the effect of the choice of the surrogate model and acquisition function optimizer. In the main benchmark experiments, we observed that our qdNAS optimizers sometimes fail to find any architecture belonging to a certain

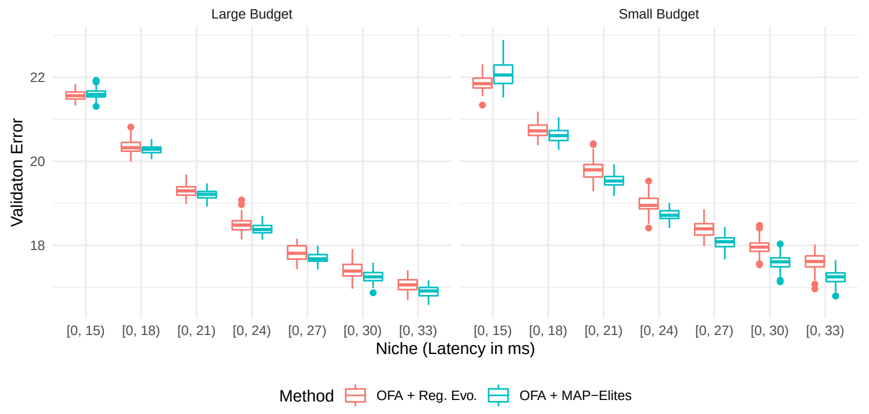


Figure 8: Regularized evolution vs. MAP-Elites within Once-for-All. Left: Large budget of total architecture evaluations. Right: Small budget of total architecture evaluations. Boxplots are based on 100 replications.

Table 8: Search space for NNI pruners on `MobileNetV2`.

Hyperparameter	Type	Range	Info
pruning_mode	categorical	{conv0, conv1, conv2, conv1andconv2, all}	
pruner_name	categorical	{l1, l2, slim, agp, fpgm, mean_activation, apoz, taylorfo}	
sparsity	continuous	[0.4, 0.7]	
agp_pruning_alg	categorical	{l1, l2, slim, fpgm, mean_activation, apoz, taylorfo}	
agp_n_iters	integer	[1, 100]	
agp_n_epochs_per_iter	integer	[1, 10]	
slim_sparsifying_epochs	integer	[1, 30]	
speed_up	boolean	{TRUE, FALSE}	
finetune_epochs	integer	[1, 27]	fidelity
learning_rate	continuous	[1e-06, 0.01]	log
weight_decay	continuous	[0, 0.1]	
kd	boolean	{TRUE, FALSE}	
alpha	continuous	[0, 1]	
temp	continuous	[0, 100]	

"agp_pruning_alg", "agp_n_iters", and "agp_n_epochs_per_iter" depend on "pruner_name" being "agp". "slim_sparsifying_epochs" depends on "pruner_name" being "slim". "alpha" and "temp" depend on "kd" being "TRUE". "log" in the Info column indicates that this parameter is optimized on a logarithmic scale.

niche (even after having used all available budget). This was predominantly the case for the very small niches in the medium and large number of niches settings (i.e., Niche 1, 2 or 3). Figure 9 shows the relative frequency of niches missed by optimizers (over 100 replications). Note that for the small number of niches settings, relative frequencies are all zero and therefore omitted. In general, model-based multifidelity variants perform better than the full-fidelity optimizers and QD optimizers sometimes perform worse than multi-objective optimizers.

We hypothesized that this could be caused by the choice of the surrogate model used for the feature functions: A random forest cannot properly extrapolate values outside the training set and therefore, if the initial design does not contain an architecture for a certain niche, the optimizer may fail to explore relevant regions in the feature space. We therefore conducted a small ablation study on the NAS-Bench-101 Cifar-10 medium number of niches benchmark problem. BOP-Elites* was configured to either use a random forest (as before) or an ensemble of feed-forward neural networks[6] (as used by BANANAS [57]) as a surrogate model for the feature function. Moreover, we

[6]with an ensemble size of five networks

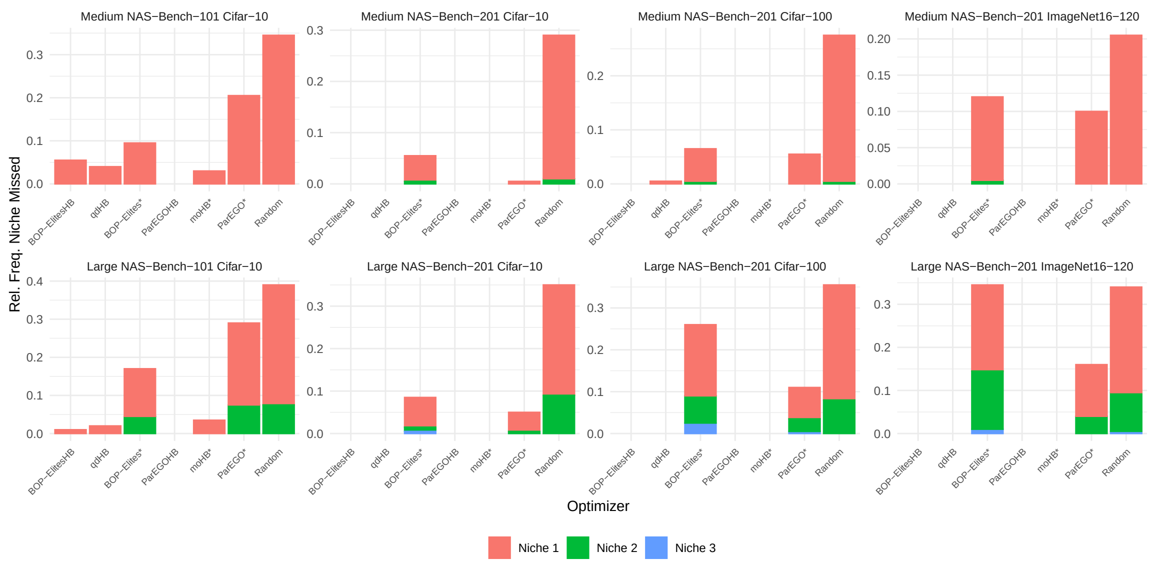


Figure 9: Relative frequency of niches missed by optimizers over 100 replications. For the small number of niches settings, relative frequencies are all zero and therefore omitted.

varied the acquisition function optimizer between a local mutation (as before) or a simple Random Search (generating the same number of candidate architectures but sampling them uniformly at random using adjacency matrix encoding). Optimizers were given a budget of 100 full architecture evaluations and runs were replicated 30 times. Figure 10 shows the anytime performance of these BOP-Elites* variants. We observe that switching to an ensemble of neural networks as a surrogate model for the feature function results in a performance boost which can be explained by the fact that this BOP-Elites* variant no longer struggles with finding solutions in the smallest niche. The relative frequencies of a solution for Niche 1 being missing are: 26.67% for the random forest + Random Search, 16.67% for the random forest + mutation, 3.33% for the ensemble of neural networks + Random Search, and 3.33% for the ensemble of neural networks + mutation. Regarding the other niches, a solution is always found. Results also suggest that the choice of the acquisition function optimizer may be more important in case of using a random forest as a surrogate model for the feature function.

H Judging Quality Diversity Solutions by Means of Multi-Objective Performance Indicators

In this section, we analyze the performance of our qdNAS optimizers in the context of a multi-objective optimization setting. As an example, suppose that niches were mis-specified and the actual solutions (best architecture found for each niche) returned by the QD optimizers are no longer of interest. We still could ask the question of how well QDO performs in solving the multi-objective optimization problem. To answer this question, we evaluate the final performance of all optimizers compared in Section 4 by using multi-objective performance indicators. Figure 11 shows the average Hypervolume Indicator (the difference in hypervolume between the resulting Pareto front approximation of an optimizer for a given run and the best Pareto front approximation found over all optimizers and replications). For these computations, the feature function was transformed to the logarithmic scale for the NAS-Bench-101 problems. As nadir points we used $(100, \log(49979275))'$ for the NAS-Bench-101 problems and $(100, 0.0283)'$ for the NAS-Bench-201 problems obtained by taking the theoretical worst validation error of 100 and feature function upper limits as found in the tabular benchmarks (plus some additional small numerical tolerance).

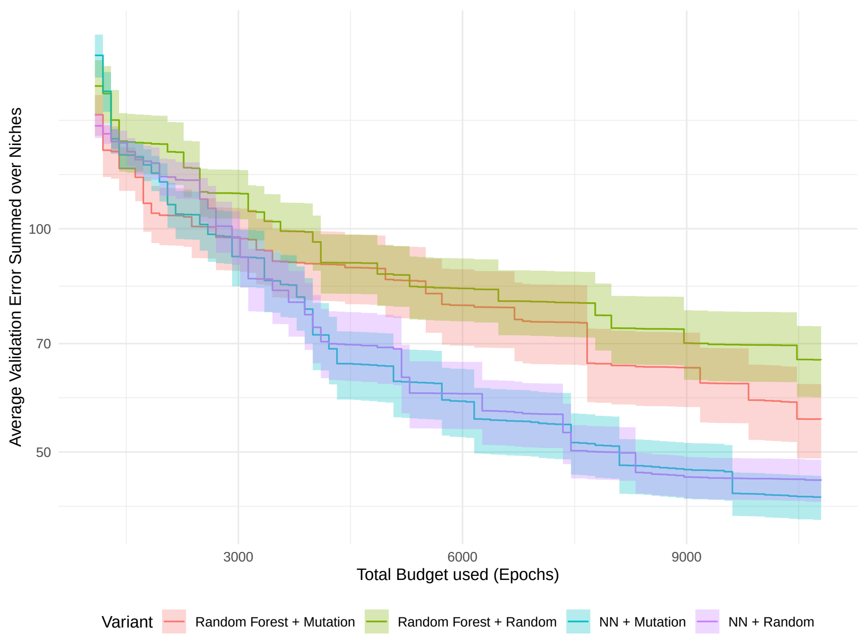


Figure 10: Anytime performance of BOP-Elites* variants configured to either use a random forest or an ensemble of neural networks as a surrogate model for the feature function crossed with either using a local mutation or a Random Search as acquisition function optimizer. NAS-Bench-101 Cifar-10 medium number of niches benchmark problem. Ribbons represent standard errors over 30 replications. x-axis starts after 10 full-fidelity evaluations.

Note that for all optimizers which are not QD optimizers, results with respect to the different number of niches settings (small vs. medium vs. large) are only statistical replications because these optimizers are not aware of the niches. We observe that ParEGOHB and ParEGO* perform well but BOP-ElitesHB also shows good performance in the medium and large number of niches settings. This is the expected behavior, as the number and nature of the niches directly corresponds to the ability of qdNAS optimizers to search along the whole Pareto front, i.e., in the small number of niches settings, qdNAS optimizers have no intention to explore.

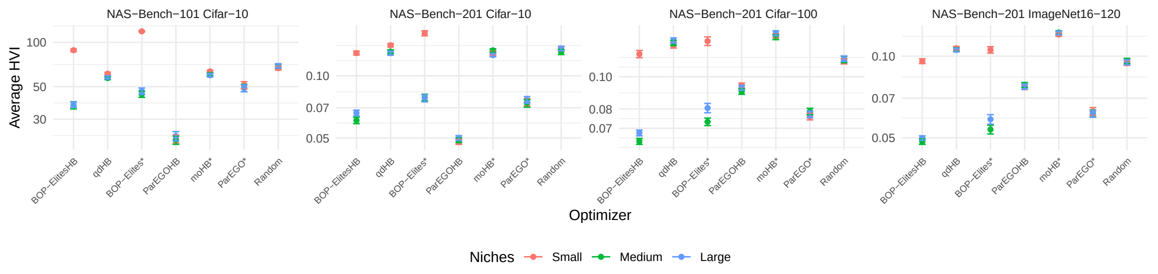


Figure 11: Average Hypervolume Indicator. Bars represent standard errors over 100 replications.

Critical differences plots ($\alpha = 0.05$) of optimizer ranks (with respect to the Hypervolume Indicator) are given in Figure 12. A Friedman test ($\alpha = 0.05$) that was conducted beforehand indicated significant differences in ranks ($\chi^2(6) = 41.61, p < 0.001$). Again, note that critical difference plots based on the Nemenyi test are underpowered if only few optimizers are compared on few benchmark problems.

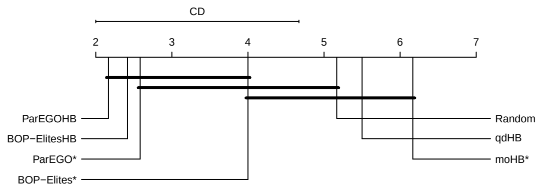


Figure 12: Critical differences plot of the ranks of optimizers with respect to the Hypervolume Indicator.

In Figure 13 we plot the average Pareto front (over 100 replications) for BOP-Elites*, ParEGO* and Random. The average Pareto fronts of BOP-Elites* and ParEGO* are relatively similar, except for the small number of niches settings, where ParEGO* has a clear advantage. Summarizing, qdNAS optimizers can also perform well in a multi-objective optimization setting, but their performance strongly depends on the number and nature of niches.

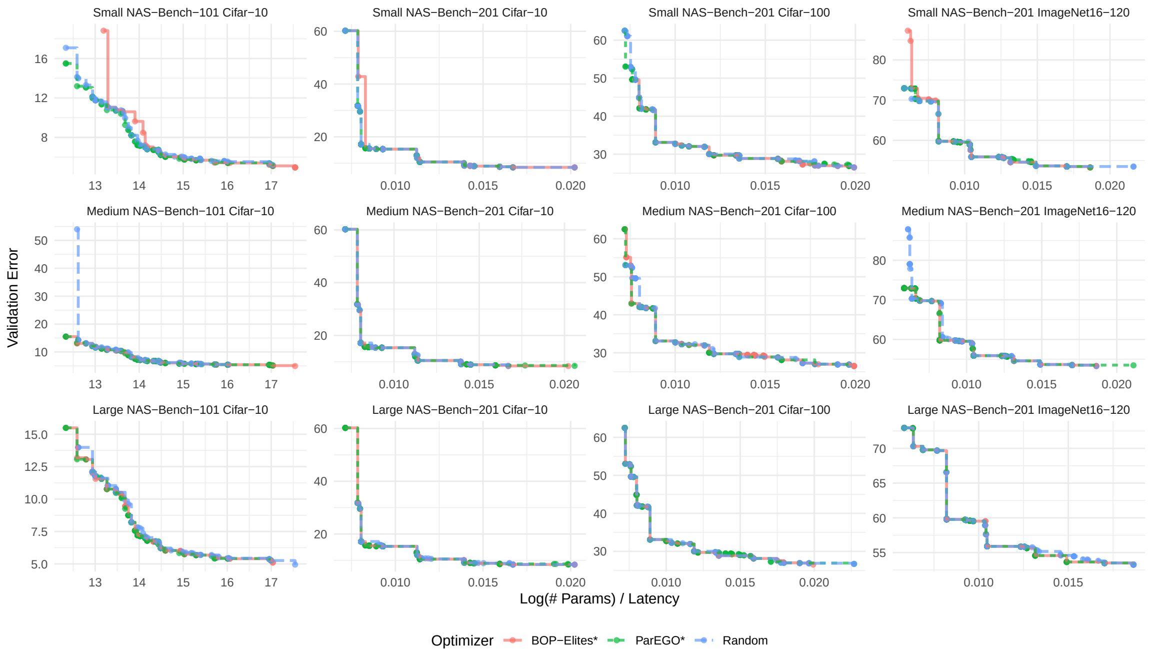


Figure 13: Average Pareto front (over 100 replications) for BOP-Elites*, ParEGO* and Random.

I Technical Details

Benchmark experiments were run on NAS-Bench-101 (Apache-2.0 License) [61] and NAS-Bench-201 (MIT License) [11]. More precisely, we used the `nasbench_full.tfrecord` data for NAS-Bench-101 and the `NAS-Bench-201-v1_1-096897.pth` data for NAS-Bench-201. Parts of our code rely on code released in Naszilla (Apache-2.0 License) [56, 57, 58]. For our benchmarks on the `MobileNetV3` search space we used the Once-for-All module [6] released under the MIT License. We rely on `ofa_mbv3_d234_e346_k357_w1.2` as a pretrained supernet and accuracy predictors and resource usage look-up tables as provided by [6]. NNI is released under the MIT License [40]. Stanford Dogs is released under the MIT License [26]. Figure 1 in the main paper has been designed using resources from `Flaticon.com`. Benchmark experiments were run on Intel Xeon E5-2697 instances taking around 939 CPU hours (benchmarks and ablation studies). The model compression application was performed on an NVIDIA DGX A100 instance taking around 3 GPU days. Total emissions are estimated to be an equivalent of 72.30 kg CO_2. All our code is available at `https://github.com/slds-lmu/qdo_nas`.

4.10 High Dimensional Restrictive Federated Model Selection with Multi-objective Bayesian Optimization over Shifted Distributions

Contributed Article:
X. Sun, A. Bommert, F. Pfisterer, J. Rahnenführer, M. Lang, and B. Bischl. High dimensional restrictive federated model selection with multi-objective Bayesian Optimization over shifted distributions. In Y. Bi, R. Bhatia, and S. Kapoor, editors, *Intelligent Systems and Applications*, pages 629–647, Cham, 2020. Springer International Publishing

Copyright Springer.

Declaration of contributions XS proposed the idea of Restrictive Federated Model Selection (RFMS) based on earlier research ideas of Michel Lang, JR and BB of multi-objective tuning of cohort. Michel Lang and BB suggested the evaluation strategy for openbox and curator datasite. The implementation of RFMS is done by XS, together with the benchmark codes. He conducted the experiments and collected the results. AB wrote some of the critical analytical code for result analysis with refinements by XS. FP wrote a first version of code for the clustering of data to simulate distribution shift with refinement by XS. XS wrote the manuscript, with refinements from all authors, especially AB.

High Dimensional Restrictive Federated Model Selection with Multi-objective Bayesian Optimization over Shifted Distributions

Xudong Sun[1(✉)], Andrea Bommert[2(✉)], Florian Pfisterer[1(✉)], Jörg Rähenfürher[2(✉)], Michel Lang[2(✉)], and Bernd Bischl[1(✉)]

[1] LMU Munich, Munich, Germany
smilesun.east@gmail.com,
{florian.pfisterer,bernd.bischl}@stat.uni-muenchen.de
[2] TU Dortmund, Dortmund, Germany
{lang,bommert,rahnenfuehrer}@statistik.tu-dortmund.de

Abstract. A novel machine learning optimization process coined Restrictive Federated Model Selection (RFMS) is proposed under the scenario, for example, when data from healthcare units can not leave the site it is situated on and it is forbidden to carry out training algorithms on remote data sites due to either technical or privacy and trust concerns. To carry out a clinical research in this scenario, an analyst could train a machine learning model only on local data site, but it is still possible to execute a statistical query at a certain cost in the form of sending a machine learning model to some of the remote data sites and get the performance measures as feedback, maybe due to prediction being usually much cheaper. Compared to federated learning, which is optimizing the model parameters directly by carrying out training across all data sites, RFMS trains model parameters only on one local data site but optimizes hyper parameters across other data sites jointly since hyper-parameters play an important role in machine learning performance. The aim is to get a Pareto optimal model with respective to both local and remote unseen prediction losses, which could generalize well across data sites. In this work, we specifically consider high dimensional data with different distributions over data sites. As an initial investigation, Bayesian Optimization especially multi-objective Bayesian Optimization is used to guide an adaptive hyper-parameter optimization process to select models under the RFMS scenario. Empirical results shows that solely using the local data site to tune hyper-parameters generalizes poorly across data sites, compared to methods that utilize the local and remote performances. Furthermore, in terms of hypervolumes, multi-objective Bayesian Optimization algorithms show increased performance across multiple data sites among other candidates.

Keywords: Federated learning · Multi-objective Bayesian Optimization · High dimensional data · Differential privacy · Distribution shift · Model selection

Y. Bi et al. (Eds.): IntelliSys 2019, AISC 1037, pp. 629–647, 2020.
https://doi.org/10.1007/978-3-030-29516-5_48

1 Introduction

1.1 Background

Federated Learning [20,24] has drawn increasing attention recently due to overwhelmingly growing data volume and an emerging request for privacy protection from the perspective of individuals, as well as the perspective of data owners, e.g. due to GDPR [26]. Usually in federated learning, a server moderates several data sites to carry out optimization iterations, like gradient descent updates, on each data site. Each data site then sends an intermediate result to the server. The server side aggregates the results and distributes it, so that each data site obtains an updated model. This distributed model training process circumvents the bottleneck of data transmission and prevents private data from leaving the data center. To further increase privacy security against attacks [26], differential private federated learning algorithms have been proposed [19,37].

Current federated learning algorithms rely on an efficient and synchronized communication protocol [20,25] across the server and different data sites as well as the availability that data on each data site can be used for training. However, it might also be expensive to meet the technical requirements to have a synchronized communication framework needed by federated learning.

From a privacy protection perspective, several attacks and defenses that undermine privacy in a federated learning context have been proposed [3,27,32]. Differential private federated learning algorithms [12,26,37] are based on standard Federated Learning algorithms, with some detail being tailored to fit the need for differential privacy.

However, there might be restrictions that the data from the remote data site can not be used for training at all. Especially when there is no established trust between parties, privacy protection and attack becomes an arm race, in which case, data owners might want to restrict the access of the data to a maximum extent but still want to participate in the community to build a predictive model that could benefit all sides. To the best of our knowledge, this is a problem that current differential private federated learning algorithms do not address yet.

In both restricted cases, sending a model to the remote data sites and asking for how good the sent model performs on the remote data sites comes at a certain cost (transmission cost and prediction computation cost for instance). This is comparably acceptable, as only aggregated statistics (typically a single number) need to be reported back.

We coin this new learning scenario Restrictive Federated Learning, emphasizing the point that only data situated locally could be used for model training, while data on the other data sites are partially observed in the sense that the analyst could only observe a scalar performance measure of a sent model on the remote data site, which is restrictive.

In this restrictive learning scenario, we could only access limited data locally for training a machine learning model, but still want to have a model that could generalize well across the data sites. Therefore, how to do model selection in this special restricted federated learning scenario is of significant interest.

Bayesian Optimization has proved to be really successful in optimizing machine learning hyper-parameters [34]. In this work, we want to investigate how it works under the RFMS scenario.

1.2 Challenges

A critical challenge in federated learning is unevenly distributed data. For example, there are situations where most features are not available on all data sites [19] or the class distribution is extremely unbalanced across different computation nodes [43].

In RFMS, there is also the challenge that data can be differently distributed on each data site. Specifically, in this work we consider the challenge that distribution of features from one data site might be considerably different from another, due to different sub-populations frequenting a given clinic for example.

Furthermore, the number of observations in clinical research is usually relatively small, while with the inclusion of genetic data, the number of features can be rather large. This makes model selection [6, 14] quite challenging. Finding stable predictive models that could generalize well to data collected from different clinical studies or cohorts is difficult.

2 Problem Statement

2.1 Terminology and Notation

To clearly address the problem, at the first step, terminologies and notations used throughout the remainder of this paper are explained.

Data Site: Data of a specific domain, clinical research for example, could be located in different places and it is expensive to carry data from one site to another due to technical or privacy concerns. We denote one of such a integrated data unity as a data site. There is a need to train a specific machine learning model for the domain, which requires collaboration across data sites. We consider data sites of following types.

Openbox Data Site D_{ob}: On the openbox data site, the analyst has full access to the data. A machine learning model can be trained locally using the data situated on openbox data site.

Curator Data Site D_{cu}: From the openbox side, curator data site can be queried for model performance, which can assist the analyst on the openbox data site to get a better model that might generalize across data sites. The curator data site D_{cu} can only be queried with respect to predictive performance, i.e. a single aggregate statistic, but the analyst from the openbox side can not access the data in any other way. This name stems from the field of differential privacy [9] where there is a curator that controls the data flow which acts like a firewall to D_{cu}. The curator has full access to D_{cu} but decides on its strategy w.r.t. which feedback value to give to the statistical query by actively perturbing and coordinating the answers given to the queries. In this work, we assume a honest answer to the query except otherwise specified.

Lockbox Data Site D_{lb}: Lockbox [13] data site refers to data sites which the analyst from the openbox side can not access by any means. In practice, lockbox correspond to data sites that could not contribute in the process of building a machine learning model due to various reasons, but are likely to participate in the future or simply benefit from the model built. From a model evaluation perspective, D_{lb} on the other hand could measure how good a machine learning model generalizes to completely unseen data.

Inbag and Outbag: For evaluation purposes, we hold out a fraction (say 20%) of the curator and openbox data which we call **outbag**, denoted by D_{cu}^{og} and D_{ob}^{og}, the leftover is called **inbag**, which is D_{cu}^{ig} and D_{ob}^{ig}. For simplicity, we use D_{ob} to represent D_{ob}^{ig} when the context is about learning and use D_{ob} to represent D_{ob}^{og} when the context is about evaluating how good a method is. Also, we define the inbag and outbag of lockbox to be identical to lockbox itself, i.e. $D_{lb} = D_{lb}^{ig} = D_{lb}^{og}$.

Model Parameter θ and Hyper-parameter ϕ: A machine learning algorithm, given a dataset D_l, where l means "learn" or "local", $D_l = D_{ob}^{ig}$, for example, and a set of hyperparameters ϕ, learns a model specified by a set of model parameters $\theta = \mathcal{L}(D_l \mid \phi)$ where $\mathcal{L}$ represent the learning process to map a dataset D_l associated with a set of hyper-parameter ϕ to a machine learning model parameter θ.

Model Performance and Loss: The performance of a model characterized by θ to a data site D is given by

$$\mathcal{F}\left(D \mid \theta = \mathcal{L}(D_l \mid \phi)\right)$$

where $\mathcal{F}$ computes an estimate of predictive performance on D, under model parameter θ trained from dataset D_l, based on hyper-parameter ϕ. By convention, we use J to represent a regret that need to be minimized, which could be $1 - \textit{accuracy}$ for example.

Restricted Federated Model Selection (RFMS) Scenario: The analyst from the openbox side want to initiate a study to a specific domain (clinical studies like cancer research for example). A machine learning model that fits the data well on the openbox side, as well as one that could generalize to a certain extent across the other data sites is required. Due to privacy sensitivity or technical difficulty, some data sites could only collaborate in a model selection process in the form of curators. Each query to the curator from the openbox side is at a certain cost. Note that all forms of data sites including openbox, curator and lockbox should be used to evaluate the selected model whenever possible.

2.2 An Example of RFMS on High Dimensional Unevenly Distributed Data

Gene Expression Omnibous (GEO) is a public available functional genomics data repository with array and sequence based data that researchers from around the world could contribute to. Although the data in GEO is publicly available instead of privacy sensitive, the origin that the datasets in GEO comes from different sources makes it a perfect example of RFMS. We use the breast cancer datasets GSE16446, GSE20194, GSE20271, GSE32646, and GSE6861 from the GEO database [8,23]. Each dataset we consider here could be regarded as a data site due to the fact that they come from different sources, by different contributors.

The publicly available microarray gene expression datasets were accessed via tools provided by the Gene Expression Omnibus (GEO) data repository. Frozen robust multiarray analysis (fRMA) [23] was used for normalization. All breast cancer datasets were checked for duplicates and a pair of patients was considered duplicate when the correlation of their expression values was at least 0.999. Duplicates were removed. The response variable is binary (classes "pathological complete response" and "residual disease") for all datasets. The six observations with a missing value for the response variable are omitted. The resulting numbers of observations per dataset are displayed in Table 1. The datasets contain clinical and gene expression data. We do not consider the clinical variables because many values are missing. The gene expression data has been measured on three different types of microarray chips (HG-U133-Plus2 for GSE16446 and GSE32646, HG-U133-A for GSE20194 and GSE20271, and HG-U133-X3P for GSE6861). As the measured genes differ between the three chips, we only consider the genes that are measured on all of the chips. Out of these 1965 genes, we only use the 1000 genes with the highest variances across all patients and datasets.

Table 1. Number of observations per GEO dataset

GEO-ID	16446	20194	20271	32646	6861
Observations	114	211	178	115	161

It can be assumed that the relation between the response variable and the covariates is not identical across the datasets and the features distribution also varies from data site to data site. This is typical for gene expression data, especially if it has been measured on different chips, at different times, at different places and after different times until the tissue was frozen. A T-SNE [22] plot by pooling the feature part of the data together from these data sites can be found in Fig. 1 where the colors indicate different data sites. From Fig. 1, it is obvious that the data sites lie on different locations in the low dimension embedding, which is a clear indicator of distribution shift across data sites. We will use this example as a major case in this paper.

2.3 Evaluation Criteria

To further explain the problem, before discussing any potential solution, we first address the question of how to evaluate model performance, which will help deeper understanding of the problem.

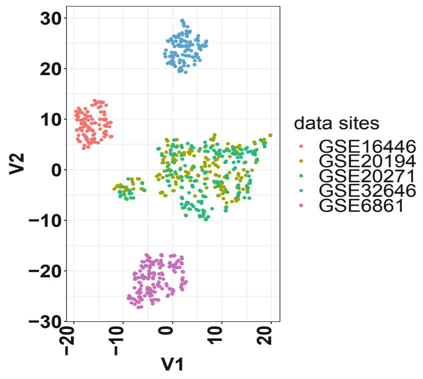


Fig. 1. T-SNE plot for the GEO datasets over data sites.

In RFMS, we want to obtain a model that generalize well for the openbox, curator and hopefully for the lockbox as well, which is a multi-objective problem. Accordingly, the selected model should also be evaluated with method that could take different objectives into consideration.

Dominated Hypervolume: A natural criterion is to measure the Dominated Hypervolume [2] of the model performance on the outbag part of openbox and curator site, as well as the lockbox, as in Eq. (1)

$$\begin{aligned}
&J^{hv}(\phi \mid D_{ob}^{og},\ D_{cu}^{og},\ D_{lb}) = \mathcal{H}\left[f_{ob}^{og},\ f_{cu}^{og},\ f_{lb}\right],\\
&f_{ob}^{og} = \mathcal{F}\left(D_{ob}^{og} \mid \theta = \mathcal{L}(D_{ob}^{ig} \mid \phi)\right),\\
&f_{cu}^{og} = \mathcal{F}\left(D_{cu}^{og} \mid \theta = \mathcal{L}(D_{ob}^{ig} \mid \phi)\right),\\
&f_{lb} = \mathcal{F}\left(D_{lb} \mid \theta = \mathcal{L}(D_{ob}^{ig} \mid \phi)\right).
\end{aligned} \tag{1}$$

where, $\mathcal{H}$ represent the calculation of the Dominated Hypervolume, and the performance on each data site outbag part is represented as $f_{ob}^{og},\ f_{cu}^{og},\ f_{lb}$ respectively. Dominated Hypervolume Indicator is also known as Lebesgue Measure or S-Metric which is the hypervolume between a non-dominated front and a reference point. Due to space limit, we invite readers who are not familiar with these multi-objective concepts to refer to the references.

3 Related Work

In this section, we review recent works that has connections with RFMS.

Nested Cross Validation (NCV): NCV [14] uses an outer loop cross validation to safe guard the risk of overfitting during the hyper-parameter tuning process. However, RFMS does not allow cross validation due to the constraint that remote data site can not be used for training.

Federated Learning: Federated learning [24] also consider situations where data is distributed non-i.i.d. across several data sites and possibly unbalanced, but they assume scenarios where data is fully accessible over a huge amounts of data sites compared to a smaller number of data points available at each site. This is different from RFMS, where we consider data can only be accessed through prediction. Moreover, in RFMS, we consider a relatively small amount of data sites with less instances but high dimensional data.

Distribution Shift: Distribution Shift refers to a mismatch in distribution between the data an algorithm was trained on, and data used for model validation or prediction. Detecting and characterizing such shift remains an open problem [29,42]. In this work, we do not drive deeper in theory of the data shift problem, but provides an empirical study which partially addresses the data shift problem, especially when feature distribution varies across data sites.

Train On Validation: In [36] the authors use parts of the validation dataset for training to generate a stable algorithm. In [41], a progressive resampling process is used. However, both works assume that all the data in question is available for training, which is not possible in RFMS.

Thresholdout Family: The author in [7] shows that differential privacy is deeply associated with model generalization and propose the Thresholdout algorithm to avoid overfitting on the validation set due to repetitive usage. [13] extends the instance wise Thresholdout to AUC measures. However, these methods rely on the i.i.d assumption of data which does not fit our scenario here.

Adaptive Regularization: In [30], the author proposed an alternative update method for model parameter θ and hyper-parameter $\phi = \lambda$ of a recommendation system [21], where the λ is the regularization parameter. In adaptive regularization, the update for the λ is based on the "future" value of performance which is also similar to the EM algorithm update process. However, adaptive regularization only works with gradient based algorithms. Especially, it is only implemented for Factorization Machine in libFM. So in general it does not work for non-gradient based optimization typed machine learning models.

Model Agnostic Meta Learning (MAML): Model Agnostic Meta Learning [10] originates from few shot learning. It aims at adapting to new instances, in which sense is similar to RFMS. However, MAML works only with gradient based method and pre-assumes that the algorithm could see the full subsequent dataset which is not possible in RFMS problem setting.

4 Methods

In this section, we first describe the general RFMS process in Sect. 4.1, then in Sect. 4.2, we propose how to handle the RFMS process with Bayesian Optimization.

4.1 Restrictive Federated Model Selection

The general process of RFMS is illustrated in Fig. 2, which depicts an asynchronous communication process during optimization. At step i, based on hyperparameter ϕ_i, the machine learning model is trained on D_{ob} to get the model parameter $\theta^i = \mathcal{L}(D_{ob} \mid \phi^i)$.

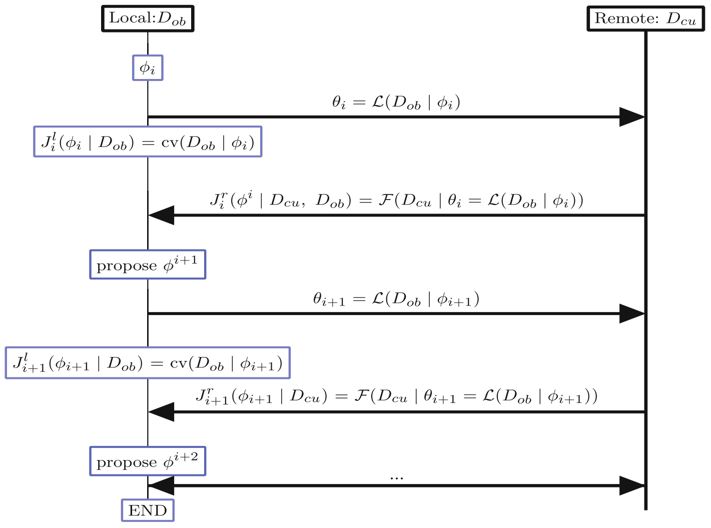

Local:D_{ob}
Remote: D_{cu}
ϕ_i
$\theta_i = \mathcal{L}(D_{ob} \mid \phi_i)$
$J_i^l(\phi_i \mid D_{ob}) = \mathrm{cv}(D_{ob} \mid \phi_i)$
$J_i^r(\phi^i \mid D_{cu},\ D_{ob}) = \mathcal{F}(D_{cu} \mid \theta_i = \mathcal{L}(D_{ob} \mid \phi_i))$
propose ϕ^{i+1}
$\theta_{i+1} = \mathcal{L}(D_{ob} \mid \phi_{i+1})$
$J_{i+1}^l(\phi_{i+1} \mid D_{ob}) = \mathrm{cv}(D_{ob} \mid \phi_{i+1})$
$J_{i+1}^r(\phi_{i+1} \mid D_{cu}) = \mathcal{F}(D_{cu} \mid \theta_{i+1} = \mathcal{L}(D_{ob} \mid \phi_{i+1}))$
propose ϕ^{i+2}
...
END

Fig. 2. Restrictive Federated Model Selection starting from step i

With the same hyper-parameter ϕ_i, a 10-fold cross validation is carried out on the openbox inbag part D_{ob}, which gives us one loss function in Eq. (2).

$$J_i^l(\phi_i \mid D_{ob}^{ig}) = cv(D_{ob}^{ig} \mid \phi_i) \tag{2}$$

where $cv(D_{ob}^{ig} \mid \phi_i)$ represent the average loss of the cross validation and J_i^l means local loss at the ith step.

Another loss function is obtained by sending the model parameters θ^i to the remote side as shown in Eq. (3)

$$J_i^r(\phi_i \mid D_{cu}^{ig}) = \mathcal{F}(D_{cu}^{ig} \mid \ \theta_i = \mathcal{L}(D_{ob}^{ig} \mid \phi_i)) \tag{3}$$

Here J_i^r means loss on the remote curator at the ith step.

At the next step, a decision process β (see Algorithm 1) based on all historical observations will propose a new hyper-parameter to be tried out for a potential better performance. This process is repeated until budget reached. The process should return the optimal hyper-parameters. The complete procedure is listed in Algorithm 1, where the decision process β to generate the proposal is approximately greedily taking the optimal of a Gaussian Process originated surrogate $\mu(\phi \mid \mathcal{R},\ \Phi)$, Expected Improvement [33], for instance. We use Φ (with an initial design sized n^{ini}) to represent the hyper-parameter buffer and $\mathcal{R}$ to represent the corresponding objective(s) buffer.

4.2 Bayesian Optimization and Baselines

Bayesian optimization tries to solve the problem of optimizing (often expensive-to-evaluate) black-box functions by using an internal empirical performance model which learns a surrogate model of the objective function while optimizing it. A widely used application for Bayesian Optimization [16] is the optimization of hyperparameters [1,33] of machine learning algorithms. Its aim is to find an optimal configuration $\phi^\star$ from the feasible region. The choice of hyperparameters for a machine learning model influences the learned model and can thus result in different performances (cf. [28,31]).

Since the distribution of the data across different data sites is unknown, we propose to treat the model selection approach as a black box optimization problem. Specifically, we use Bayesian Optimization in Algorithm 1 to solve the Restrictive Federated Model Selection problem with the following variants.

Local Single Objective (lso) Bayesian Optimization: In local single objective (lso) Bayesian Optimization, we set objective function as cross validation performance on the local openbox data site, hyper-parameters are tuned based on $J^{lso}(\phi) = J^l = cv\left(D^{ig}_{ob} \mid \phi\right)$ where J^l is defined in Eq. (2).

Federated Single Objective (fso) Bayesian Optimization: In Federated Single Objective Bayesian Optimization, we combine the openbox cross validation aggregated results in Eq. (2) and curator performance in Eq. (3) linearly as objective function, hyper-parameters are tuned based on

$$\begin{aligned}&J^{fso}(\phi) = \alpha\ J^l(\phi \mid D^{ig}_{ob}) + (1-\alpha)\ J^r(\phi \mid D^{ig}_{cu})\\&\alpha \in [0,\ 1]\,.\end{aligned} \tag{4}$$

Specifically, we use $fso2$ to represent $\alpha = 0.2$ and $fso8$ to represent $\alpha = 0.8$ and so on. Note that $\alpha = 1$ corresponds to lso. We use different α to check if there is an obvious effects by changing α.

Federated Multiobjective Objective (fmo) Bayesian Optimization: Multiobjective Bayesian Optimization [15] optimizes multiple objectives simultaneously, by random linear combination or optimization a S-metric based objective, which avoid deciding which linear combination parameter α to choose. In this work, we use the Parego algorithm [18] to optimize the local objective in Eq. (2) and remote objective in Eq. (3) jointly.

Random Search Multiobjective (rand_mo): To evaluate whether Bayesian optimization makes sense, we randomly search the hyper-parameter space and select the pareto front [38] as final output, which we call random search multi-objective.

4.3 Semi-simulation of Data Sites

Publicly available datasets which could fit into the RFMS scenario intrinsically are rare. To get data from a diversified source aside from the Gene Expression Ominbus, we turn to approximate the RFMS scenario by splitting an existing dataset into different parts as if each part sits on a different data site. In practice,

Algorithm 1. RFMS with Bayesian Optimization (RFMS-BO)

```
 1: procedure RFMS-BO                      ▷ data site notation here refer to the inbag part
 2:     $\Phi_{1:n^{ini}} = \{\phi_1, \ldots, \phi_{n^{ini}}\}$          ▷ initial design as hyper-parameter buffer
 3:     $\mathcal{R}_0 = \emptyset$                                   ▷ objective buffer
 4:     for $i$ in $1 : n^{ini}$, $\phi_i$ in $\Phi_{1:n^{ini}}$ do
 5:         $J_i^l(\phi_i \mid D_{ob}) = cv(D_{ob} \mid \phi_i)$   ▷ Cross validation performance aggregation as loss
 6:         $\theta_i = \mathcal{L}(D_{ob} \mid \phi_i)$                       ▷ training on $D_{ob}$ with $\phi_i$
 7:         $J_i^r(\phi_i \mid D_{cu}, D_{ob}) = \mathcal{F}(D_{cu} \mid \theta_i)$          ▷ test on curator
 8:         $\mathcal{R}_i = \mathcal{R}_{i-1} \| \left[J_i^l, J_i^r\right]$               ▷ populate objective buffer
 9:     end for
10:     fit $\mu(\phi \mid \mathcal{R}_i,\ \Phi_{1:n^{ini}})$                     ▷ train Surrogate Function
11:     $j = i + 1$
12:     while budget not reached do
13:         $\phi_j = \beta(\mu(\phi \mid \mathcal{R}_{j-1},\ \Phi_{1:j-1}))$        ▷ propose new hyper-parameter
14:         $\Phi_{1:j} = \Phi_{1:j-1} \| [\phi_j]$              ▷ populate hyper-parameter buffer
15:         $J_j^l(\phi_j \mid D_{ob}) = cv(D_{ob} \mid \phi_j)$
16:         $\theta_j = \mathcal{L}(D_{ob} \mid \phi_j)$
17:         $J_j^r(\phi_j \mid D_{cu}) = \mathcal{F}(D_{cu} \mid \theta_j)$
18:         $\mathcal{R}_j = \mathcal{R}_{j-1} \| \left[J_j^l,\ J_j^r\right]$               ▷ populate objective buffer
19:         $j \leftarrow j + 1$
20:         update $\mu(\phi \mid \mathcal{R}_j,\ \Phi_{1:j})$                        ▷ update surrogate
21:     end while
22:     $i^* = \arg\max_i(\mathcal{R})$
23:     $\{\phi^*\} = \Phi_{i*}$
24:     $\{\theta^*\} = \mathcal{L}(D_{ob}; \phi^*)$
25:     return $\phi^*, \theta^*$
26: end procedure
```

we always split an existing dataset into 5 parts to keep consistence with our GEO datasets.

Since we use real data, but kind of simulate to split the dataset into different data sites to fit into the RFMS scenario, we call this semi-simulation of data sites. We propose the following strategy to semi-simulate the data sites.

Stratified Random Split (SRS): First, split the dataset into two parts according to a factor column. Specifically, we use the target column in a classification dataset. Then, each factor part is randomly split into 5 buckets. The positive class part got $b_1^p, \ldots, b_5^p$ and the negative class part got $b_1^n, \ldots, b_5^n$, where b_i^n and b_i^p represent the ith bucket in the negative part and positive part respectively. Lastly, sort the buckets in each factor part according to the number of instances and combine the buckets in reversing order to form each data site, i.e., $d_i = b_{s^n(i)}^n \| b_{s^p(6-i)}^p$, where d_i represents the ith combined data site, s^n and s^p are the sorted index vector of each part. We use $\|$ to denote pooling two data buckets.

Dimension Reduction and Clustering (DRC): First, carry out a dimension reduction technique on the dataset like Principal Component Analysis. Then split the dataset into positive class part and negative class part. Cluster each part into 5 clusters, i.e. $c_1^n, \ \ldots, \ c_5^n$ for the negative class part and $c_1^p, \ \ldots, \ c_5^p$ for the positive class part. Sort the clusters with respect to the cluster size in each part and combine them in reversed order to form each data site, i.e., $d_i = c^n_{s^n(i)} \| c^p_{s^p(6-i)}$, where d_i represent the ith combined data site, s^n and s^p are the sorted index vector of each part. We use $\|$ to denote pooling two data together.

We choose Mixture of Gaussian Model (MOG) for the clustering, due to consideration that MOG could also serve as a density estimator.

$$p(X) = \textstyle\sum_{k=1}^5 c_k \mathcal{N}(X|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{5}$$

In MOG, each cluster is represented by a Gaussian distribution $\mathcal{N}(X|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ with its own parameters $\boldsymbol{\mu}_k$(mean) and $\boldsymbol{\Sigma}_k$(covariance), as shown in Eq. (5), c_k is the mixing coefficient of each cluster. For each of the chosen datasets in Table 2, we model the data distribution as $p(X)$ in Eq. (5) and approximately, each cluster resulted data site represent a different distribution. For simplicity, we assume all clusters are with different mean vectors but share the same covariance matrix to assemble a distribution shift. The T-SNE plot is done to the SRS scenario (Fig. 3) and the DRC scenario. In DRC, we use PCA as dimension reduction, keeping 10% (Fig. 4) and 50% (Fig. 5) of the total variance to tell if the reduced dimension makes a big difference in generating an unevenly distributed data sites scenario). From these figures, we do not observe a big difference between different percentage of variance to reserve in PCA, but observe a big difference between SRS and DRC where SRS generates a more evenly distributed data sites, while DRC generates more uneven distributions across different clusters (data sites).

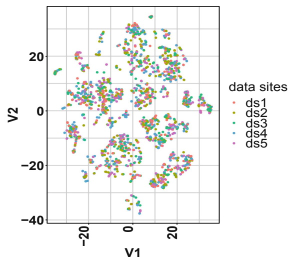


Fig. 3. Stratified random split (SRS)

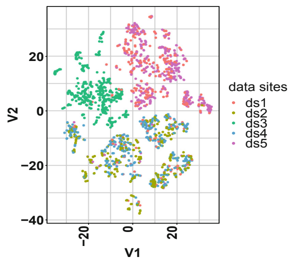


Fig. 4. DRC with PCA and keep 10% variance

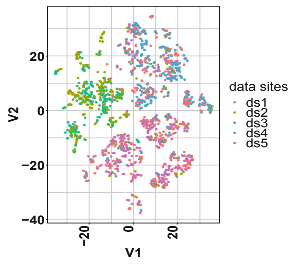


Fig. 5. DRC with PCA and keep 50% variance

5 Experiment

5.1 Settings

Since we have selected 5 datasets from the Gene Expression Ominibus to represent 5 data sites, we will consider the exemplary problem of 5 data sites for the remainder of the paper.

In the experiment, one of the 5 data sites is used as openbox D_{ob}, another one as lockbox D_{lb} and the three left over are used as curators D_{cu}. We choose to have only one openbox to simulate the scenario, that usually only local data at the current data sites are fully available to the analyst. We choose to have 3 curators and only 1 lockbox to simulate the scenario that more data sites want to collaborate with the openbox data site. Curator data site losses are weighted by the size of the each curator data site during optimization. With this strategy, there are in total $5 \times 4 = 20$ combinations of openbox-curator-lockbox on the 5 datasets. Each openbox and lockbox combination defines one scenario. Each scenario is repeated 10 times (10 replications) where we call each replication one experiment. We sequentially run all RFMS methods, described in Sect. 4.2, with 3 machine learning algorithms (kernel support vector machine, random forest and elastic net). Thus, we have in total $20 \times 10 \times 3 = 600$ experiments given a RFMS problem with 5 data sites. All Bayesian Optimization procedures share the same initial design of 20 randomly selected configurations, and are then run for another 40 iterations. Thus in total we have a budget of 60 evaluations. To have a fair comparison, Random Search use the same number of evaluations.

In order to evaluate our method, we randomly partition openbox and the curator into two parts, namely an inbag part (80%) and an outbag part (20%). Replications mentioned above could average out the random splits and other stochastic factors. We use D_{ob}^{ig} for training a model, and use D_{cu}^{ig} as well as D_{ob}^{ig} for model selection. The outbag parts of openbox and curator are reserved for post-hoc analysis. This allows us to assess, whether our methods overfit in each of the two boxes. Additionally, performance is also recorded on the lockbox site for another aspect of evaluation. We then compare the different methods described in Sect. 4.2 on the outbag portion of the respective boxes (as noted in Sect. 2.1, all data of lockbox belongs to outbag).[1]

5.2 Selection of Dataset for Semi-simulation

In order to validate our results on different data sources, we obtain additional data sets from OpenML [39]. As no datasets with an intrinsic splitting mechanism such as the GEO dataset (where each dataset comes from a particular source) are available, we simulate the RFMS scenario according to the strategies described in Sect. 4.3.

Model generalization becomes more difficult when there are comparatively more features than instances. Therefore, we restrict ourselves to datasets with a

[1] source code in https://github.com/compstat-lmu/paper_2019_multiobjective_rfms

relatively high-dimension characteristics: Since we intend to split a dataset into 5 parts as 5 data sites, the number or instances in each data site is approximately reduced by 5 times compared to the original dataset (we rebalanced cluster results which generate too small clusters but adding instances to the smallest cluster from the biggest cluster until the smallest cluster reaches 10% of the total number of instances), but the number of features over the number of instances get to be approximately 5 times of the original ratio, so a p (number of features) over n (number of instances) ratio of more than 0.2 in the original dataset corresponds to $\frac{p}{n} = 1$ in each data site, thus we consider datasets with $\frac{p}{n}$ ratio around 0.2 to be high-dimensional.

Too few instances is more prone to problems in data resampling processes like cross validation. For example, one fold of the cross validation might contain no instance from the underrepresented class. Thus we do not want too extremely unbalanced classification datasets. In order to have a sufficient amount of data in each of the 5 boxes, we select only data sets with more than 500 instances. For the purpose of simplicity, we additionally restrict our data set selection to data sets that are (i) binary class, (ii) do not have missing values. As a result, we use the data sets in Table 2 to provide additional validation of the proposed methods.

Table 2. List of datasets from OpenML

Name	n	p	p/n	Class ratio
gina agnostic	3468	970	0.28	0.97
Bioresponse	3751	1776	0.47	0.84
$fri\_c4\_500\_100$[a]	500	100	0.2	0.77

[a]https://www.openml.org/d/742

Since close or even identical predicative performance values on a problem can occur for varying machine learning hyperparameters, when the predicative performance is used as the target for Gaussian Process regression, it can create numerical difficulties, so hyperparameter tuning might fail for a particular algorithm, even though we use a nugget value of $1e-6$. Therefore, to get fair comparison, all algorithms are run sequentially over a problem on the same computing node. Only those experiments with all algorithms finished are used for analysis, where in practice, we only get neglectable number of experiments (around 100 out of 1800 experiments, which is 5 percent) within which at least one algorithm is not finished, see Fig. 9 and Fig. 11. The Winner-vs-Loser plots are more effective than carrying out statistical tests.

5.3 Machine Learning Algorithms and Hyper-Parameters

We choose 3 machine learning algorithms (which we call learner) based on the consideration that the learners should be representative to different mechanisms of various machine learning algorithms. Elastic net logistic regression (implemented in R package *glmnet* [11]) is a good representative for linear classifier which could deal with high dimensional data (**classif.glmnet**), thus chosen because according to [35], one should not rule out simple models prematurely. R package *ranger* [40] implements a random forest (**classif.ranger**) which is a state of art non-linear learner that has shown outstanding performance. Kernel support vector machine (*ksvm*) (**classif.ksvm**) implemented in [17], is a nonlinear classifier which could deal with high dimensional data. The hyper-parameters to be optimized with their ranges are shown in Table 3. Hyper-parameter tuning is done with mlr[4] and $mlrMBO$[5]. Meaning of hyper-parameters can be found in respective packages.[2]

Table 3. List of hyperparameters

Classifier	Hyperparameter	Type	Range
glmnet	alpha	numeric	(0, 1)
glmnet	s	numeric	$(2^{-10}, 2^{10})$
ksvm	C	numeric	$(2^{-15}, 2^{15})$
ksvm	sigma	numeric	$(2^{-15}, 2^{15})$
random forest	num.trees	integer	(100, 5000)
random forest	min.node.size	integer	(1, 50)
random forest	sample.fraction	numeric	(0.1, 1)

5.4 Results and Discussion

In this section, we compare different candidates of RMFS methods proposed in Sect. 2.3 with respect to their predictive performance. Our aim is to obtain machine learning models, that generalize well across data sites. As an aggregate measure, we choose the dominated hypervolume of the data kept out-of-bag in the openbox D_{ob}^{og}, curator D_{cu}^{og} and lockbox D_{lb} respectively as shown in Eq. (1). We consider the average performance on the curators for calculating the hypervolume. Lockbox data measures how our methods generalize to subpopulations not considered at all during the training and model selection process. Using hypervolume results in a comprehensive overview of them.

Results on the GEO Datasets. As shown in Fig. 6, we compare the mean dominated hypervolume from Eq. (1) of 3 machine learning algorithms (corresponding to the 3 panels in the plot) and several RFMS methods. We aggregate

[2] https://github.com/mlr-org/mlr/blob/3edac9f65ed5c157a3d868fe8d2908eaa2a09ebd/R/RLearner_classif_glmnet.R\#L7.

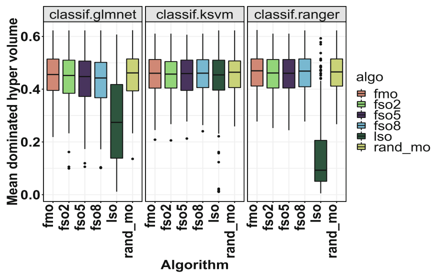


Fig. 6. Dominated hypervolume on GEO datasets

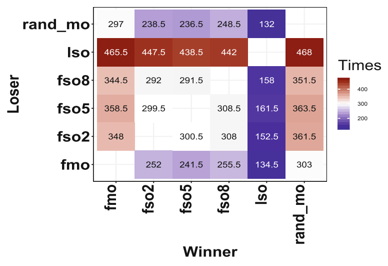


Fig. 7. Comparison of wins and losses on GEO dataset

over 10 replications and 20 combinations of possible openbox-lockbox combinations.

From Fig. 6, we can observe that *lso* performs the worst among other candidates, showing that in the RFMS scenario, solely tuning hyper-parameters on the local openbox data site will usually not lead to a model that generalizes well across data sites, which is in accordance with intuition. The other candidates methods including **fmo** and several **fso** variants, that predicting on the data of the curator and using this performance as a feedback performs better, showing that the feedback could help in arriving at models which generalize better. However, the considered Bayesian Optimization approaches do not overrate the multi-objective random search **rand_mo**, nor do we observe any effect of changing α in the performance of **fso**. In order to make a more precise comparison, we compare the pairwise wins and losses of all the RFMS methods in terms of dominated hypervolume. For each experiment, we build a $0-1$ matrix to compare the win and loss of each algorithm pair (when method A is compared against method B, we take 0 for loss, 1 for win, and 0.5 for tie) and aggregate the matrix across all 600 experiments. Results are shown in Fig. 7, where the horizontal axis corresponds to winners and the vertical axis correspond to losers. The elements in the matrix correspond to how many times the winner has won against the loser. It is easily observable that both bi-criteria methods (**fmo** and **rand_mo**) are slightly better than other candidates, as they win more than half of the experiments.

Results on the Semi-simulated RFMS Scenario. To avoid single dataset bias, we also analyze how the same algorithms compare under our semi-simulated RFMS scenario described in Sect. 4.3 over data of various sources.

Dimension Reduction and Clustering (DRC): We first simulate the RFMS scenario with DRC explained in Sect. 4.3, which could result in a situation that data from different data sites are differently distributed, where we keep 10 percent variance in the PCA step.

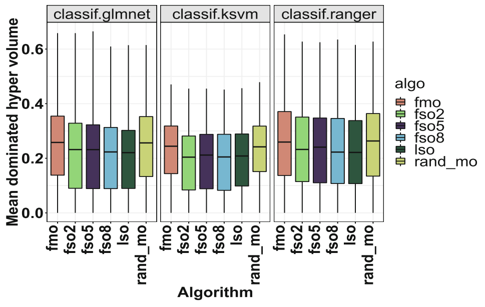


Fig. 8. Aggregated mean dominated hypervolume under DRC scenarios obtained over OpenML datasets

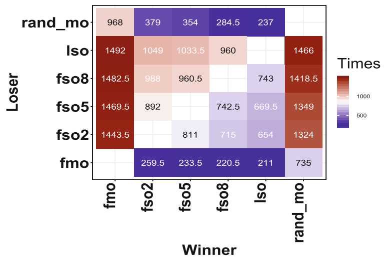


Fig. 9. Aggregated wins and losses on the DRC scenarios obtained over OpenML datasets

Figure 8 shows the dominated hypervolume by aggregating across all the datasets in Table 2. Compared to Fig. 6, it is more obvious here that the multi-objective methods work better than the single objective Bayesian optimization methods. In Fig. 9, we have the Winner-vs-Loser plot for the aggregated results on the OpenML datasets listed in Table 2, where the multi-objective candidates outperform the rest by a large margin. Furthermore, **fmo** wins **rand_mo** by a considerable margin, giving confidence that Bayesian Optimization make a difference compared to random search.

Stratified Random Split (SRS): To answer the question if a different data splitting technique affects the comparison, we use the stratified technique described in Sect. 4.3 which corresponds to the situation that data being more evenly distributed across data sites. Figure 10 shows the hypervolume plot, from which we can still observe the pattern that the multi-objective candidates perform better in terms of hypervolume, while compared to Fig. 8, all methods show increased performance under this evenly distributed data scenario across data sites, possibly due to the bonus of evenly distributed data scenario. In Fig. 11, we compare the wins and losses for each pair of candidates, where in this case, the **fmo** wins **rand_mo** by a larger margin, maybe because the generate a simpler RFMS scenario for the Bayesian Optimization.

6 Summary

We introduce a novel learning scenario, Restrictive Federated Model Selection (RFMS), which could play an important role in clinical research, where privacy sensitive immobile high dimensional data is differently distributed among various data sites, in which case federated learning is not applicable due to a lack of access to data from all data sites to be used for training. RFMS is a model selection process in this scenario, with the aim to obtain a model that generalizes comparably well across data sites with potential different distributions. Compared to Federated Learning, RFMS can be carried out in an asynchronous

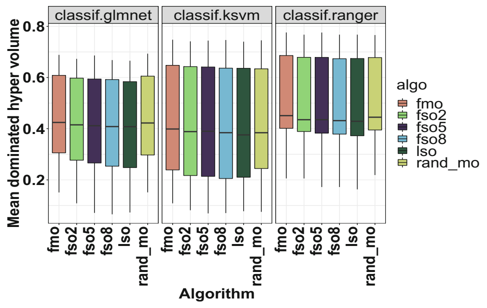


Fig. 10. Aggregated mean dominated hyper-volume under SRS scenario obtained over OpenML datasets

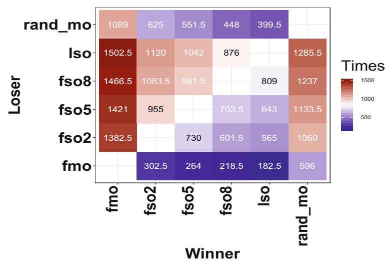


Fig. 11. Aggregated wins and losses over SRS scenario obtained over OpenML datasets

fashion, which is not communication hungry compared to standard federated learning and much easier to be deployed. Additionally, the amount of information that needs to be transferred for each query is comparatively small which takes less efforts to be deployed.

As an initial investigation, we compare various methods for model selection and hyper-parameter tuning using Bayesian Optimization. Empirical results from various data sources indicate that Federated Multi-objective Bayesian Optimization compares favorably against other single objective candidates as well as multi-objective random search, in terms of better generalization across data sites.

Acknowledgment. This work was supported by Deutsche Forschungsgemeinschaft (DFG), Project RA 870/7-1 and BI 1902/1-1. The authors thank Janek Thomas, Philip Probst and Martin Binder for helpful suggestions.

References

1. Bergstra, J.S., Bardenet, R., Bengio, Y., Kégl, B.: Algorithms for hyper-parameter optimization. In: Advances in Neural Information Processing Systems, pp. 2546–2554 (2011)
2. Beume, N., Rudolph, G.: Faster s-metric calculation by considering dominated hypervolume as klee's measure problem. Universitätsbibliothek Dortmund (2006)
3. Bhowmick, A., Duchi, J., Freudiger, J., Kapoor, G., Rogers, R.: Protection against reconstruction and its applications in private federated learning. arXiv:1812.00984 (2018)
4. Bischl, B., Lang, M., Kotthoff, L., Schiffner, J., Richter, J., Studerus, E., Casalicchio, G., Jones, Z.M.: mlr: Machine learning in R. J. Mach. Learn. Res. **17**(1), 5938–5942 (2016)
5. Bischl, B., Richter, J., Bossek, J., Horn, D., Thomas, J., Lang, M.: mlrMBO: a modular framework for model-based optimization of expensive black-box functions. arXiv preprint arXiv:1703.03373 (2017)
6. Cawley, G.C., Talbot, N.L.: On over-fitting in model selection and subsequent selection bias in performance evaluation. J. Mach. Learn. Res. **11**, 2079–2107 (2010)

7. Dwork, C., Feldman, V., Hardt, M., Pitassi, T., Reingold, O., Roth, A.: Guilt-free data reuse. Commun. ACM **60**(4), 86–93 (2017)
8. Edgar, R., Domrachev, M., Lash, A.E.: Gene Expression Omnibus: NCBI gene expression and hybridization array data repository. Nucl. Acids Res. **30**(1), 207–210 (2002)
9. Elder, S.: Bayesian adaptive data analysis guarantees from subgaussianity. arXiv preprint arXiv:1611.00065 (2016)
10. Finn, C., Abbeel, P., Levine, S.: Model-agnostic meta-learning for fast adaptation of deep networks. arXiv preprint arXiv:1703.03400 (2017)
11. Friedman, J., Hastie, T., Tibshirani, R.: glmnet: Lasso and elastic-net regularized generalized linear models. R Packag. Version **1**(4) (2009)
12. Geyer, R.C., Klein, T., Nabi, M.: Differentially private federated learning: A client level perspective. arXiv preprint arXiv:1712.07557 (2017)
13. Gossmann, A., Pezeshk, A., Sahiner, B.: Test data reuse for evaluation of adaptive machine learning algorithms: over-fitting to a fixed 'test' dataset and a potential solution. In: Medical Imaging 2018: Image Perception, Observer Performance, and Technology Assessment, vol. 10577, p. 105770K. International Society for Optics and Photonics (2018)
14. Guyon, I., Saffari, A., Dror, G., Cawley, G.: Model selection: beyond the Bayesian/frequentist divide. J. Mach. Learn. Res. **11**, 61–87 (2010)
15. Horn, D., Dagge, M., Sun, X., Bischl, B.: First investigations on noisy model-based multi-objective optimization. In: International Conference on Evolutionary Multi-Criterion Optimization, pp. 298–313. Springer (2017)
16. Jones, D.R., Schonlau, M., Welch, W.J.: Efficient global optimization of expensive black-box functions. J. Glob. Optim. **13**(4), 455–492 (1998)
17. Karatzoglou, A., Smola, A., Hornik, K., Zeileis, A.: kernlab-an S4 package for kernel methods in R. J. Stat. Softw. **11**(9), 1–20 (2004)
18. Knowles, J.: ParEGO: a hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. IEEE Trans. Evol. Comput. **10**, 50–66 (2006)
19. Konecnỳ, J., McMahan, H.B., Ramage, D., Richtárik, P.: Federated optimization: distributed machine learning for on-device intelligence. arXiv preprint arXiv:1610.02527 (2016)
20. Konečnỳ, J., McMahan, H.B., Yu, F.X., Richtárik, P., Suresh, A.T., Bacon, D.: Federated learning: strategies for improving communication efficiency. arXiv preprint arXiv:1610.05492 (2016)
21. Kushwaha, N., Sun, X., Singh, B., Vyas, O.: A lesson learned from pmf based approach for semantic recommender system. J. Intell. Inf. Syst. **50**(3), 441–453 (2018)
22. Maaten, L.v.d., Hinton, G.: Visualizing data using t-SNE. J. Mach. Learn. Res. **9**, 2579–2605 (2008)
23. McCall, M.N., Bolstad, B.M., Irizarry, R.A.: Frozen robust multiarray analysis (fRMA). Biostatistics **11**(2), 242–253 (2010)
24. McMahan, H.B., Moore, E., Ramage, D., Hampson, S., y Arcas, B.A.: Communication-efficient learning of deep networks from decentralized data. In: AISTATS (2017)
25. McMahan, H.B., Moore, E., Ramage, D., Hampson, S., et al.: Communication-efficient learning of deep networks from decentralized data. arXiv preprint arXiv:1602.05629 (2016)
26. Melis, L.: Building and evaluating privacy-preserving data processing systems. Ph.D. thesis, UCL (University College London) (2018)

27. Melis, L., Song, C., De Cristofaro, E., Shmatikov, V.: Inference attacks against collaborative learning. arXiv preprint arXiv:1805.04049 (2018)
28. Probst, P., Bischl, B., Boulesteix, A.L.: Tunability: Importance of hyperparameters of machine learning algorithms. arXiv preprint arXiv:1802.09596 (2018)
29. Rabanser, S., Günnemann, S., Lipton, Z.C.: Failing loudly: an empirical study of methods for detecting dataset shift. arXiv preprint arXiv:1810.11953 (2018)
30. Rendle, S.: Learning recommender systems with adaptive regularization. In: Proceedings of the Fifth ACM International Conference on Web Search and Data Mining, pp. 133–142. ACM (2012)
31. van Rijn, J.N., Hutter, F.: Hyperparameter importance across datasets. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 2367–2376. ACM (2018)
32. Shokri, R., Stronati, M., Song, C., Shmatikov, V.: Membership inference attacks against machine learning models. In: 2017 IEEE Symposium on Security and Privacy (SP), pp. 3–18 (2017). https://doi.org/10.1109/SP.2017.41
33. Snoek, J., Larochelle, H., Adams, R.P.: Practical Bayesian optimization of machine learning algorithms. In: Advances in Neural Information Processing Systems, pp. 2951–2959 (2012)
34. Snoek, J., Rippel, O., Swersky, K., Kiros, R., Satish, N., Sundaram, N., Patwary, M., Prabhat, M., Adams, R.: Scalable Bayesian optimization using deep neural networks. In: International Conference on Machine Learning, pp. 2171–2180 (2015)
35. Strang, B., van der Putten, P., van Rijn, J.N., Hutter, F.: Don't rule out simple models prematurely: a large scale benchmark comparing linear and non-linear classifiers in OpenML. In: International Symposium on Intelligent Data Analysis, pp. 303–315. Springer (2018)
36. Tennenholtz, G., Zahavy, T., Mannor, S.: Train on validation: squeezing the data lemon. arXiv preprint arXiv:1802.05846 (2018)
37. Truex, S., Baracaldo, N., Anwar, A., Steinke, T., Ludwig, H., Zhang, R.: A hybrid approach to privacy-preserving federated learning. arXiv preprint arXiv:1812.03224 (2018)
38. Van Veldhuizen, D.A., Lamont, G.B.: Evolutionary computation and convergence to a pareto front. In: Late Breaking Papers at the Genetic Programming 1998 Conference, pp. 221–228 (1998)
39. Vanschoren, J., van Rijn, J.N., Bischl, B., Torgo, L.: OpenML: networked science in machine learning. ACM SIGKDD Explor. Newsl. **15**(2), 49–60 (2014)
40. Wright, M.N., Ziegler, A.: Ranger: a fast implementation of random forests for high dimensional data in C++ and R. arXiv preprint arXiv:1508.04409 (2015)
41. Zeng, X., Luo, G.: Progressive sampling-based bayesian optimization for efficient and automatic machine learning model selection. Health Inf. Sci. Syst. **5**(1), 2 (2017)
42. Zhang, K., Schölkopf, B., Muandet, K., Wang, Z.: Domain adaptation under target and conditional shift. In: International Conference on Machine Learning, pp. 819–827 (2013)
43. Zhao, Y., Li, M., Lai, L., Suda, N., Civin, D., Chandra, V.: Federated learning with non-iid data. CoRR abs/1806.00582(2018)

CHAPTER 5

CONTRIBUTIONS - FAIRNESS

5.1 Debiasing classifiers: is reality at variance with expectation?

Contributed Article:
A. Agrawal, F. Pfisterer, B. Bischl, J. Chen, S. Sood, S. Shah, F. Buet-Golfouse, B. A. Mateen, and S. Vollmer. Debiasing classifiers: is reality at variance with expectation?, 2020, arXiv:2011.02407

Declaration of contributions The project originated with an implementation of several debiasing methods in Julia [1] by AA with guidance from SV and JC. Experimental results reported in the manuscript stem from a subsequent benchmark mostly conducted by AA with input from FP, JC and SV. JC, with help from AA, FP and SV drafted the manuscript with feedback by all other authors. FBG contributed the theoretical analysis on the convergence on performance - fairness tradeoffs. FP contributed several smaller experiments regarding class balance and comparisons to previous work. All authors provided feedback and iteratively refined the manuscript in subsequent iterations.

[1]https://github.com/ashryaagr/Fairness.jl

Debiasing classifiers: is reality at variance with expectation?

Ashrya Agrawal
Birla Institute of Technology and Science
Pilani, India
`ashryaagr@gmail.com`

Florian Pfisterer
Ludwig-Maximilians-University
Münich, Germany
`florian.pfisterer@stat.uni-muenchen.de`

Bernd Bischl
Ludwig-Maximilians-University
Münich, Germany
`bernd.bischl@stat.uni-muenchen.de`

Francois Buet-Golfouse
J.P. Morgan
London, United Kingdom
`francois.buet-golfouse@jpmorgan.com`

Srijan Sood
J.P. Morgan AI Research
New York, New York, USA
`srijan.sood@jpmorgan.com`

Jiahao Chen
J.P. Morgan AI Research
New York, New York, USA
`jiahao.chen@jpmorgan.com`

Sameena Shah
J.P. Morgan AI Research
New York, New York, USA
`sameena.shah@jpmorgan.com`

Sebastian Vollmer
University of Warwick
Warwick, United Kingdom
`svollmer@warwick.ac.uk`

Abstract

We present an empirical study of debiasing methods for classifiers, showing that debiasers often fail in practice to generalize out-of-sample, and can in fact make fairness worse rather than better. A rigorous evaluation of the debiasing treatment effect requires extensive cross-validation beyond what is usually done. We demonstrate that this phenomenon can be explained as a consequence of bias-variance trade-off, with an increase in variance necessitated by imposing a fairness constraint. Follow-up experiments validate the theoretical prediction that the estimation variance depends strongly on the base rates of the protected class. Considering fairness–performance trade-offs justifies the counterintuitive notion that partial debiasing can actually yield better results in practice on out-of-sample data.

1 Introduction

Artificial intelligence and machine learning (AI/ML) are now used for many high-stakes decision-making processes at scale [36, 43], such as credit decisions [15, 50], medical diagnoses [49], and criminal sentencing [3, 18, 7]. In these use cases, unfairness is not just an ethical concern, but has legal and regulatory dimensions as well [5, 15, 53, 33, 26]. As such, regulators have signalled their interests in detecting and remediating bias in these real-world applications [26, 48].

Bias can originate from any part of the machine learning modeling process, ranging from exclusionary biases [5, 13, 41] in training data, to problem definitions or feedback cycles that reinforce historical and systemic discrimination [23, 34, 4, 37]. To address bias in a model, one must first identify the relevant fairness metrics [38, 51] and then select a method to debias the model with respect to those

Preprint. Under review.

metrics. However, both aspects are challenging in practice. It is not always obvious which fairness definitions are relevant for a particular application [3, 7], and remediating bias usually comes at a cost. For example, a credit decisioning model has to be accurate in order to be profitable, which motivates fairness notions like equality of opportunity. At the same time, there are reputational and regulatory risks associated with bias in incorrect decisions, leading to considerations of equalized false negative rate and equalized false positive rate [25]. These different definitions of fairness cannot be satisfied simultaneously due to well-known impossibility theorems [31, 17]. Furthermore, a debiased model will not be used in practice if its performance degrades too much. Therefore, in practice, we have to consider not only fairness–fairness trade-offs, but also the fairness–performance trade-offs to determine the best debiased model [40, 30].

Assumptions. The setting for our paper assumes that 1) membership in protected classes is fully known, ignoring practical concerns to the contrary [16, 27], 2) all relevant fairness and performance metrics can be clearly identified at the outset, and 3) remediation is only at single point in time, ignoring time-evolving concerns [34]. Despite this restricted setting, we find that the practicalities of debiasing are already sufficiently rich for in-depth study. Other work identified technical challenges resulting from the lack of native support for fairness or debiasing concerns in major machine learning software libraries, but do not consider variance or sensitivity issues [10]. More recently, Rodolfa et al. [42] show that trade-offs between recall parity and precision@k are often small in real-world projects.

Given this body of work, we were therefore surprised to it is surprising to see results like those in Figure 2, which suggest that classifiers can exhibit any combination of improved or worsened fairness, and also improved or worsened performance, after using standard debiasing algorithms. In this paper, we show that out-of-sample generalization error is responsible for the fluctuations observed in the aforementioned figure, and that careful estimation of such error is essential for proper evaluation of debiasing methods. While previous works have studied distributionally robust optimization for fairness [35] and data-dependent constraint generalization [19], we focus on the generalization of fairness algorithms.

Our Contributions. In Section 2, we show how to generalize existing debiasers to apply to fairness metrics other than what they were originally defined for. We introduce generalized reweighing which can apply to other fairness definitions beyond demographic parity and identify fairness definitions for which reweighing cannot generalize to. We also introduce a new NLinProg debiaser which generalizes the equalized odds debiaser, and is capable of handling multiple fairness and performance metrics simultaneously. In Section 3, we present a detailed empirical study across nine different models, showing that debiasing methods generally fail to achieve perfect fairness in out-of-sample measurements, and produce large variance in the actual metrics, and tend to overfit on training data. In Section 4, we present our main theoretical result, Theorem 1, showing that the fluctuations we observed empirically can be attributed to bias-variance trade-off. In Section 5, we verify a prediction from this analysis, that the ability to debias varies with the base rate of the protected class. In Section 6, we show how an explicit consideration of the fairness–performance trade-offs motivates the notion of partial debiasing. We also show experimentally the somewhat counter-intuitive result that a *partial* debiasing treatment can actually yield classifiers with better out-of-sample fairness. We introduce other related work throughout the exposition of this paper, in lieu of a dedicated section.

Notation. **Sets.** In general, calligraphic letters like $\mathcal{A}$ denote a set, capital letters A denote a variable that is an element of a set $\mathcal{A}$, and small letters a denote a value that the variable A can take. Let $S \in \mathcal{S} = \{0, 1\}$ be a binary *protected class*, $X \in \mathcal{X}$ be some set of *features* that explicitly excludes $\mathcal{S}$, $Y \in \mathcal{Y} = \{0, 1\}$ be a binary *outcome variable*, and $\hat{Y} \in \mathcal{Y}$ be an *estimator* for Y. While we specialize to the cases of binary $\mathcal{Y}$ and $\mathcal{S}$ for the ease of presentation, our results generalize to larger finite classes. Furthermore, let $Z = (X, Y) \in \mathcal{Z} = \mathcal{X} \times \mathcal{Y}$ and $W = (X, Y, S) \in \mathcal{W} = \mathcal{X} \times \mathcal{Y} \times \mathcal{S}$, Additionally, define $\mathcal{D} \in \mathcal{W}^n$ to be *in-sample (training) data* with n points, $\mathcal{D}^\star \in \mathcal{W}^{n^\star}$ to be out-of-sample *(testing) data* with $n^\star$ points, and $\Delta^k = \{z \in \mathbb{R}^{k+1} : z \geq 0, \|z\|_1 = 1\}$ be the standard non-negative simplex of dimensionality k. **Classification functions.** Let $\mathcal{F} : \mathcal{X} \times \mathcal{S} \to \mathcal{Y}$ be the function space of *$\mathcal{S}$-aware classifiers*, where each element $f \in \mathcal{F}$ is a classification function, and $\mathcal{F}_0 : \mathcal{X} \to \mathcal{Y}$ be the function space of *$\mathcal{S}$-oblivious classifiers*. Each $\mathcal{S}$-oblivious classifier $f_0 \in F_0$ has a 1:1 relation to a trivial $\mathcal{S}$-aware classifier $f \in F : f(x, s) = f_0(s)$ which simply ignores the s argument. We differentiate between aware and oblivious classifiers only where necessary. Also, let $\mathcal{H} \subseteq \mathcal{F}$ be some family of classifiers, and $\mathrm{id}_{\mathcal{A}} : \mathcal{A} \to \mathcal{A}$ be the identity function over the set $\mathcal{A}$.

Debiasing functions. Let $g_{\text{pre}} : \mathcal{X}\times\mathcal{S} \to \mathcal{X}$ be a pre-processing debiasing function, $g_{\text{post}} : \mathcal{Y}\times\mathcal{S} \to \mathcal{Y}$ be a post-processing debiasing function, and $G : \mathcal{F} \to \mathcal{F}$ be an in-processing debiaser, which is a higher-order function that is an endomorphism over $\mathcal{F}$. **Loss functions.** Let $\ell : \mathcal{F}\times\mathcal{W} \to \mathbb{R}_0^+$ be a performance loss such as the hinge or binomial deviance, and $\phi_h : \mathbb{R}^2 \to= R_0^+$ be a loss function associated with the fairness definition h. **Metrics.** Let $\gamma : \mathcal{F}\times\mathcal{W}^{\mathbb{N}} \to [0, 1]$, $\gamma(f, \mathcal{D})$ be the accuracy of the classifier f on the data set $\mathcal{D}$, and $\tau_h : \mathcal{F}\times\mathcal{W}^{\mathbb{N}} \to [0, 1]$, $\tau_h(f, \mathcal{D})$ the fairness metric as defined in Definition 2 corresponding to the fairness definition h. When clear from context, the arguments f and $\mathcal{D}$ will be dropped for brevity.

1.1 Fairness definitions & metrics

Fairness metric	Equality statement
Equalized false omission rate (EFOR) [7]	$\Pr(Y = 1\|\hat{Y} = 0, S = s) = \Pr(Y = 1\|\hat{Y} = 0)$
Predictive parity (PP) [17]	$\Pr(Y = 1\|\hat{Y} = 1, S = s) = \Pr(Y = 1\|\hat{Y} = 1)$
Demographic parity (DP) [14]	$\Pr(\hat{Y} = 1\|S = s) = \Pr(\hat{Y} = 1)$
Equalized false negative rate (EFNR) [17]	$\Pr(\hat{Y} = 0\|Y = 1, S = s) = \Pr(\hat{Y} = 0\|Y = 1)$
Predictive equality (PE) [17]	$\Pr(\hat{Y} = 1\|Y = 0, S = s) = \Pr(\hat{Y} = 1\|Y = 0)$
Equality of opportunity (EOp) [25]	$\Pr(\hat{Y} = 1\|Y = 1, S = s) = \Pr(\hat{Y} = 1\|Y = 1)$
Equalized odds (EOd) [25]	EOp and PE

Table 1: Group fairness definitions used in this paper.

Many technical definitions of fairness exist and they have been reviewed elsewhere [38, 51, 7, 30]. We present only the definitions of fairness that we will study in this paper in Table 1. In addition to choosing a suitable fairness definition, we also have to choose some loss function, ϕ, to quantify the discrepancy from perfect fairness. One such function is the Calders-Verwer gap [14] $\Delta_{\text{DP}} = \Pr(\hat{Y}=1|S=1) - \Pr(\hat{Y}=1|S=0)$, which is simply the difference of the two sides of the equation that define demographic parity, and vanishes when perfect fairness exists. In addition to absolute differences, other metrics based on ratios, relative differences, or other more complicated losses have have been proposed. In this paper, we focus on symmetrized ratio metrics as defined in Definition 2.

1.2 Pre-, in- and post-processing methods for debiasing

Toolkits such as Aequitas [45], IBM AI Fairness 360 [6], Microsoft Fairlearn [9], and Amazon SageMaker Clarify [47] provide many debiasing algorithms. These algorithms are traditionally classified as pre-processing, in-processing and post-processing methods, which are depicted at the functional level in Figure 1. In this section, all primed quantities have been debiased. A *pre-processing debiaser* first transforms the input features $\mathcal{X}$ using some function $g_{\text{pre}} : \mathcal{X}\times\mathcal{S} \to \mathcal{X}$, then feeds the transformed features as input to an oblivious classifier $f : \mathcal{X} \to \mathcal{Y}$. The debiased classifier is then the composition $f' : \mathcal{X}\times\mathcal{S} \to \mathcal{Y}$, $f' = f \circ g_{\text{pre}}$. A *post-processing debiaser* takes the output of some oblivious classifier, $\hat{Y} \in \mathcal{Y}$, then transforms this output using some function $g_{\text{post}} : \mathcal{Y}\times\mathcal{S} \to \mathcal{Y}$, The debiased classifier is then the composition $f' : \mathcal{X}\times\mathcal{S} \to \mathcal{Y}$, $f' = g_{\text{post}} \circ (f \times \text{id}_{\mathcal{S}})$, where $\text{id}_{\mathcal{S}}$ is the identity function over protected class. Finally, an *in-processing debiaser* transforms some oblivious classifier f into an $\mathcal{S}$-aware but debiased classifier $f' = G(f)$, using the function-to-function mapping $G : (\mathcal{X} \to \mathcal{Y})\times\mathcal{W} \to (\mathcal{X}\times\mathcal{S} \to \mathcal{Y})$. In general, the resulting classifier cannot be written

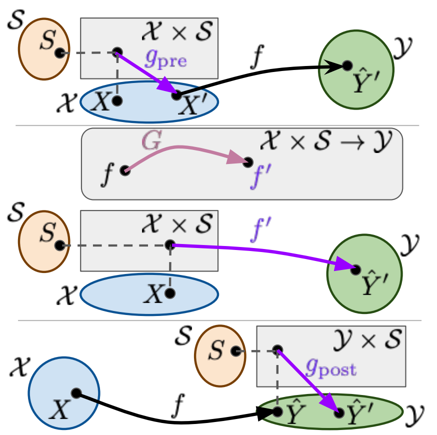


Figure 1: Overview of debiasing methods as described in Section 1.2. Top: pre-processing methods. Middle: in-processing methods. Bottom: post-processing methods.

as a function composition involving the original oblivious classifier f. Some in-processing debiasers like prejudice removal [29] further require that the debiased classifier f' be $\mathcal{S}$-oblivious, which is equivalent to the defining the debiased classifier $f'(X, S) = f(X)$ to be independent of S always. In other words, pre-processing debiasers transform the features $\mathcal{X}$, post-processing debiasers transform the predictions $\hat{\mathcal{Y}}$, and in-processing debiasers transform the classifiers f.

We conclude the introduction with two debiasing algorithms as illustrations of the general principle. **Reweighing** (RW) is a pre-processing debiaser introduced to enforce demographic parity (DP) [28]. Since DP is satisfied when Y and S are independent, reweighing assigns each data point i a weight $w_{\text{DP},i} = \Pr(\hat{Y}=y_i)\Pr(S=s_i)/\Pr(\hat{Y}=y_i, S=s_i)$, altering the measure associated with the sampled distribution of (Y, S) to match what would be expected from statistical independence. **Equalized odds** (EOd) is a post-processing debiaser [25, 40] that calculates probabilities $\Pr(\hat{Y}'|\hat{Y}, S)$ that the predictions $\hat{Y}$ should be flipped to yield the debiased predictions $\hat{Y}'$ that satisfy equalized odds fairness, while having $\hat{Y}'$ as close as possible to $\hat{Y}$.

2 Generalized debiasers

The reweighing pre-processor and equalized odds post-processsor are specialized to specific fairness definitions, demographic parity and equalized odds respectively. In this section, we show how these debiasers can be generalized to other fairness definitions.

2.1 Generalized reweighing for pre-processing

The reweighting pre-processor of Section 1.2 can be easily extended to some, but not all, other definitions of group fairness. For example, considering $\hat{Y} \perp\!\!\!\perp S|Y=1$ instead of $\hat{Y} \perp\!\!\!\perp S$ gives an immediate generalization of reweighing for equality of opportunity (EOp) instead of DP. As EOp fairness requires $\Pr(\hat{Y}=1|S=0, Y=1) = \Pr(\hat{Y}=1|S=0, Y=1)$, the corresponding reweighing scheme is simply $w_{\text{EOp},i} = \Pr(\hat{Y}=y_i)/\Pr(\hat{Y}=y_i|S=s_i, Y=1)$. However, there is no such reweighing scheme for equalized odds (EOd), which requires that both EOp and PE hold. Each equation demands its own reweighing scheme, with the first as before and the second as in $w_{\text{PE},i} = \Pr(\hat{Y}=y_i)/\Pr(\hat{Y}=y_i|S=s_i, Y=0)$, which will in general differ from the weights $w_{\text{EOp},i}$. Thus, reweighing as a method for exact debiasing works for neither composite fairness definitions that require multiple equality constraints, nor situations requiring multiple fairnesses to be satisfied simultaneously. It is therefore natural to consider the possibility of some interpolation scheme between different weighting schemes. We will revisit this idea later in Section 6.

2.2 Nonlinear programs for post-processing (NLinProg; NLP)

We now introduce **NLinProg** (NLP), a generalization of the equalized odds post-processor to allow for arbitrary combinations of group fairnesses to be debiased simultaneously.

Algorithm 1 The NLinProg post-processing debiaser

Input: Predictions $\hat{Y}$, protected class S, performance losses $\{\ell^{(i)}\}_i$, and fairness losses $\{\phi^{(i)}\}_i$.
Output: Debiased predictions $\hat{Y}'$.

```
1: Compute the solution $z = (\Pr(\hat{Y}'=y', \hat{Y}=y|S=s))_{y',y,s}$ to the PFOP (1).
2: For each prediction $\hat{Y}=y$ with protected class label $S=s$, choose a corresponding debiased
   prediction $\hat{Y}'=y'$ with probability $\Pr(\hat{Y}'=y'|\hat{Y}=y, S=s)$.
```

Definition 1. *The* ***performance–fairness optimality problem*** *(PFOP) is to determine the fairness-confusion tensor (FACT) $z = (TP_1, FN_1, FP_1, TN_1, TP_0, FN_0, FP_0, TN_0)/N$ [30] that solves:*

$$\arg\min_{z\in\Delta^7} \sum_i \mu_i \ell^{(i)}(z) + \sum_j \lambda_j \phi^{(j)}(z), \tag{1}$$

where $TP_0/N = \Pr(\hat{Y}'=1, \hat{Y}=1, S=0)$ is the normalized true positive entry for $S=0$, and similarly for the other entries of z, $\Delta^7 = \{z \in \mathbb{R}^8 : z \geq 0, \|z\|_1 = 1\}$ is the standard non-negative *simplex, $\ell^{(i)} : \Delta^7 \to \mathbb{R}_0^+$ is some performance loss with corresponding Lagrange multiplier μ_i, and $\phi^{(i)} : \Delta^7 \to \mathbb{R}_0^+$ is some fairness loss with corresponding Lagrange multiplier λ_j.*

We implement Algorithm 1 in the JuMP [22] framework for the Julia programming language [8], which uses Ipopt [52] for interior point optimization. The MIT-licensed open source implementation is available on GitHub.[1]

Unless otherwise specified, our subsequent experiments specialize to one accuracy loss $\ell^{(1)}(z) = 1 - \gamma(z)$, where $\gamma(z) = \sum_i (TP_i + TN_i)/N$ is the usual definition of accuracy, and fairness loss $\phi^{(1)}(z) = 1 - \tau_h(z)$, where τ_h is a quantity we will now define.

Definition 2. *For a FACT z, define $z_{|S=s} = (TP_s, FN_s, FP_s, TN_s)/N$ as the restriction of z to entries corresponding to $S = s$. Let $h : [0,1]^4 \to \mathbb{R}$ be a group fairness expressible as a constraint $h(z_{|S=1}) = h(z_{|S=0})$. Then, the* ***symmetrized fairness gap*** *for the fairness h at z is $\Delta_h(z) = |h(z_{|S=1}) - h(z_{|S=0})|$, and the* ***symmetrized ratio metric*** *for h evaluated at z is $\tau_h(z_{|S=0}, z_{|S=1}) = \min\left(h(z_{|S=1})/h(z_{|S=0}), h(z_{|S=0})/h(z_{|S=1})\right)$.*

It is easy to show that $\tau_h \in [0,1]$; we omit the proof of this simple fact. Furthermore, τ_h is symmetric in its arguments, which removes the need to assume that either class is generally privileged. Where clear from context, we will (with abuse of notation) also write the above as $\tau_h(z)$.

Example 1. *Demographic parity $\Pr(\hat{Y} = 1|S = 1) = \Pr(\hat{Y} = 1|S = 0)$ can be expressed as $h_{\text{DP}}(z_{|S=1}) = h_{\text{DP}}(z_{|S=0})$ with the function $h_{\text{DP}}(z_{|S=s}) = \Pr(\hat{Y} = s|S = s) = (TP_s + FP_s)/(TP_s + FP_s + FN_s + TN_s)$.*

We do not recommend NLinProg for general use—as we will see in Section 3, its performance is generally Pareto suboptimal, in that it yields neither the most fair classifiers nor the most accurate classifiers. However, for our experiments, NLinProg serves as a useful construct for investigating the general behavior of post-processing methods.

3 Empirical evaluation of debiasers

Methodology. We now evaluate the performance of three representative debiasers, RW, EOd (as described in Section 1.2), and NLP (Algorithm 1) on nine different debiasing experiments as stated in Table 2, representing different fairness criteria, data sets and debiasing strategies. We observe the phenomena in this section when running similar experiments using the Python toolkits Aequitas [45] and Fairness 360 [6], and have carefully reimplemented the algorithms in our own Julia implementation (provided in the Supplement) to verify that these effects are not the results of undiagnosed implementation bugs. We present results from our own implementations, which corroborate similar findings from the Python codes.

The classifier trained for each experiment is a random forest classifier estimated using the MIT-licensed `DecisionTree.jl` [44] Julia package, which implements the standard classification and regression trees (CART) [12] and random forest algorithms [11]. While hyperparameter tuning is an important part of developing fair real-world models [46, 39], we keep all hyperparameters at the same default values to facilitate comparison across these varied experiments, eliminating variation due to hyperparameter choice. Our evaluation criteria are the ratio of out-of-sample fairnesses τ/τ_0 for the debiased and original classifiers, with τ as defined in Definition 2, and the ratio of out-of-sample accuracies γ/γ_0 respectively. Unlike many previous studies, we focus on the *out-of-sample* behavior of the original and debiased classifiers, and estimate the generalization error by computing metrics across 100 different train–test splits computed from ten times ten-fold cross-validation (10 CV 10). Such extensive evaluation is necessary to reduce the error bars on the fairness metrics τ to determine if a debiaser had a statistically meaningful treatment effect; our experiments demonstrating such necessity are detailed in the Supplement.

Results. Figure 2 summarizes the results of our experiments. Within each subplot and point type, each point corresponds to the exact same classifier type, debiased the exact same way, but repeated over 100 different train–test splits arising from ten times ten-fold cross-validation (10 CV 10). The

[1]URL redacted for double-blind peer review.

	Data set	Protected class	Fairness metric	Source
A	Adult income	sex	PP	[21, 32]
B	German credit	marital_status	EFOR	[21]
C	Portuguese bank marketing	gender	EFOR	[21]
D	COMPAS	race	EFPR	[3]
E	Loan Defaults	sex	EFOR	[21]
F	Student Performance	sex	EFNR	[21]
G	Communities and crime	racepctblack	EFPR	[21]
H	Framingham Heart Study	male	EFOR	[20]
I	Medical Expenditure	race	EFOR	[6]

Table 2: List of experiments with data sets and associated fairness metrics used in our benchmarking study of Section 3.

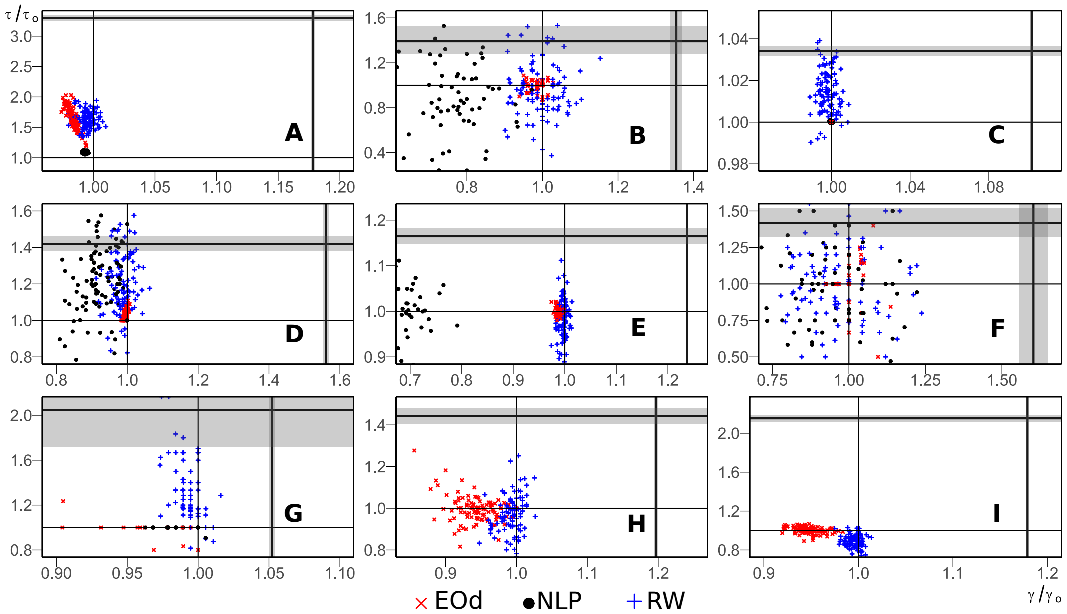


Figure 2: Plots of fairness ratios τ/τ_0 (vertical axes) against accuracy ratios γ/γ_0 (horizontal axes) for the experiments of Section 3 and Table 2 using random forest classifiers, showing that none of the reweighing (RW), equalized odds (EOd), and NLinProg (NLP) debiasers can consistently debias all the experiments.

only variation thus comes from the specific subset of data used for model training, and the test data for evaluation. The thick cross-hairs represent the ideal perfect fairness and accuracy, with the grey regions representing a one standard deviation spread across the folds. The narrow cross-hairs pinpoint the point where $\gamma = \gamma_0$ and $\tau = \tau_0$, i.e., where the debiaser had no treatment effect whatsoever.

Naïvely, we would expect that $\tau > \tau_0$ and $\gamma \approx \gamma_0$, i.e., that the fairness should improve while the accuracy stays roughly constant or perhaps decreases due to an implicit fairness–accuracy trade-off. Instead, we see that for Experiments A, E, H, and I, no debiaser was able to attain the target of maximal fairness. In fact, some experiments (like NLP in A or EOd in D) show essentially no change in the fairness and accuracy metrics at all. More worryingly, nearly all the experiments show large scatter in the out-of-sample fairness, with many points *below* the $\tau = \tau_0$ line. Our results therefore show that not only are debiasers unable to guarantee fairness out-of-sample, but even when it can do so for a particular train–test split, the effect can disappear entirely for different test data.

Experiments A, C, G, H and I also show evidence of an fairness–accuracy trade-off: as the fairness improves, the accuracy worsens, and the graphs generally trace out a negative slope. The satisfiability analysis of Kim et al. [30] shows that perfect accuracy and fairness can be attained in theory for all the experiments; however, we can understand this effect as arising from change in Bayes rate due to the additional fairness constraint imposed [30]. Nevertheless, we also see evidence of overfitting, not just in the variance of γ/γ_0, but also in many points with $\gamma > \gamma_0$, where debiasing *increased* the accuracy of the classifier, but not in a robust way. Our results agree with Friedler et al. [24], who showed that debiasing methods are prone to overfit on the training set, in that debiasing outcomes vary depending on the details of the train/test split, albeit without an explanation for this phenomenon. Our results are also consistent with [46], who showed that retuning hyperparameters is necessary to improve generalizability, also we do not investigate the effect of hyperparameter tuning in our work.

In summary, 1) the large variance in fairness metrics necessitate extensive uncertainty quantification to ascertain the treatment effect, 2) despite controlling for this variance, fairness can either improve or worsen after debiasing, and 3) accuracy usually decreases after debiasing, sometimes severely so. Below in Section 4, we provide a theoretical analysis of these phenomena in Theorem 1 in terms of bias-variance trade-off. In additional experiments in Section 6 and the Supplement, we also demonstrate the somewhat counter-intuitive result that a *partial* debiasing treatment can actually yield more fair classifiers in practice, which is the case for 12 out of the 27 combinations of experiment and debiaser.

4 Convergence of performance–fairness trade-offs

We now present a theoretical analysis of the phenomena we have observed above. To simplify our approach, we consider the penalized (or dual) version of machine learning problems involving fairness constraints. Our starting point is the scalarized optimization program $\lambda\ell + (1-\lambda)\phi$, with fairness loss $\phi : \mathbb{R}^2 \to \mathbb{R}_0^+$, for example, $\phi(x, y) = |x - y|$. The trade-off is parameterized by λ, interpolating linearly between considering only fairness ($\lambda = 0$) and only performance ($\lambda = 1$).

We want to know how the *empirical* trade-off, as measured on some test set $\mathcal{D}^\star$, converges to the *true* trade-off, as measured on the true underlying distribution $(Z, S) \sim \mathcal{P}$.

Definition 3. *Let $\ell, \mu : \mathcal{H}\times Z\times S \to \{0,1\}$ be indicator functions corresponding to the performance and fairness criteria such that when the desired criteria are satisfied, $\mathbb{E}_{(Z,S)\sim\mathcal{P}}(\ell(f, Z, S)) = 0$, and $\phi(\tilde{z}_0, \tilde{z}_1) = 0$, where $\tilde{z}_s = \mathbb{E}_{(Z_{|S=0})\sim\mathcal{P}}(\mu(f, Z, S))$. Then, the* ***population empirical risk*** *$L_\mathcal{P}$ for a population $\mathcal{P}$ is*

$$L_\mathcal{P}(f) = \lambda\mathbb{E}_{(Z,S)\sim\mathcal{P}}(\ell(f, Z, S)) + (1-\lambda)\phi\left(\tilde{z}_0, \tilde{z}_1\right). \tag{2}$$

An example of ℓ would be misclassification error $\ell = \mathbf{1}_{\{\hat{Y}\neq Y\}}$ (the complement of accuracy, $\mathbf{1}_{\{\hat{Y}=Y\}}$), while an example of μ would be predictive parity, $\mu = \mathbf{1}_{\{\hat{Y}=1\}}$, corresponding to the fairness constraint $\Pr(\hat{Y} = 1|S = 1) = \Pr(\hat{Y} = 1|S = 0)$, i.e., demographic parity. The fairness loss ϕ is related to the symmetrized fairness gap Δ_h defined in Definition 2, since we can take $\phi(z_{|S=0}, z_{|S=1}) = |h(z_{|S=0}) - h(z_{|S=1})| = \Delta_h(z)$.

Definition 4. *The* ***sample empirical risk*** *for a data set $\mathcal{D}$ is*

$$L_\mathcal{D}(f) = \lambda l^{(m)}(\mathcal{D}) + (1-\lambda)\phi\left(l_0^{m_0}(\mathcal{D}), l_1^{m_1}(\mathcal{D})\right), \tag{3}$$

where $l^{(m)}(\mathcal{D}) = \sum_{(z,s)\in\mathcal{D}} \ell(f, z, s)/m$ is the mean empirical performance loss, $l_s^{(m_s)}(\mathcal{D}) = \sum_{(z,s')\in\mathcal{D}:s'=s} \mu(f, z, s)/m_s$ is the mean empirical fairness loss for the subgroup $S = s$, $m_s = |\{(z, s') \in \mathcal{D} : s' = s\}|$ is the sample sizes for the group $S = s$, and $m = m_0 + m_1 = |\mathcal{D}|$.

We now derive the limiting distribution of $L_{\mathcal{D}^\star}(f)$ and show that it exhibits some form of bias–variance decomposition.

Theorem 1. *Let $f : \mathcal{X} \to \mathcal{Y}$ be a classification function and ℓ and μ be the indicator functions of Definition 3. Assume that we have observed m iid samples $\mathcal{D} = \{(Z_j, S_j) : (Z_j, S_j) \sim \mathcal{P}\}_{j=1}^m$ from a population distribution $\mathcal{P}$, the variance of $\ell(f, Z, S)$ is finite, the fairness penalty function ϕ is at least once-differentiable, and the variance of $\mu(f, Z, S)$ is finite. Then, the sample empirical loss converges asymptotically to the population empirical loss: $\sqrt{m}\left[L_\mathcal{D}(f) - L_\mathcal{P}(f)\right] \xrightarrow[m\to\infty]{} N\left(0, \mathbb{V}_{\lim}(f)\right)$, with limiting variance*

$$\begin{aligned}\mathbb{V}_{\lim}(f) = \lambda^2 \sum_{s\in\mathcal{S}} \pi_s(\sigma_s^\ell)^2 + \lambda^2 \sum_{s\neq s'} \pi_s\pi_{s'}\left(L_{\mathcal{P},s}(f) - L_{\mathcal{P},s'}(f)\right)^2 \\ +(1-\lambda)^2 \sum_s k_s^2 \frac{(\sigma_s^\mu)^2}{\pi_s} + 2\lambda(1-\lambda)\sum_s k_s \operatorname{Cov}_{(z,s')\in D_s}\left(\ell(f,z,s), \mu(f,z,s)\right),\end{aligned} \tag{4}$$

where Cov *is the covariance,* $s, s' \in \mathcal{S}$, $\mathcal{D}_s = \{(z, s) \in \mathcal{D} : s' = s\} \subseteq \mathcal{D}$ *is the subset of data with protected class membership* $S = s$, $\pi_s = Pr[S = s]$ *is the base rate of the* <u>*protected class*</u> $S = s$, $L_{\mathcal{P},s}(f) = \mathbb{E}_{(z,s)\in\mathcal{D}_s}(\ell(f, z, s))$ *is the sample expected loss* ℓ *over* $\mathcal{D}_s$, $(\sigma_s^\ell)^2 = \mathbb{V}_{(z,s')\in\mathcal{D}_s}(\ell(f, z, s))$ *is the sample variance of the loss* ℓ *over* $\mathcal{D}_s$, $M_{\mathcal{P},s}(f)$ *and* $(\sigma_s^\mu)^2$ *are the analogous mean and variance for the loss* μ*, and* $(k_0, k_1)^T = \nabla\phi\left(M_{\mathcal{P},0}(f), M_{\mathcal{P},1}(f)\right)$ *is the gradient of* ϕ *at the true value of the fairness function.*

This result can be proved with repeated use of the central limit theorem, the delta method and Slutsky's lemma. The full proof is included in the Supplement.

The first three terms in the limiting variance $\mathbb{V}_{\lim}(f)$ can be interpreted as 1) the intra-group variance, 2) the (statistical) bias that measures unfairness through the difference in loss for each group S, and 3) the variance stemming from the fairness penalty term. The last terms grow with $(1-\lambda)^2 k_i^2$, which intuitively captures how sensitivity to fairness constraints leads to increased variance. Interestingly, these terms are also inversely proportional to the base rates $\pi_s = \Pr(S = s)$, meaning that imbalance in the <u>protected class</u> increases the variance.

5 Empirical dependence on protected class imbalance

Theorem 1 predicts that the estimation variance depends on protected class imbalance, specifically, that the standard deviation of the estimated classifier $\sigma(\hat{f}) \sim 1/\Pr(S = 1)$ as $\Pr(S = 1)$ goes to zero. We should therefore expect a similar behavior for the fairness metric $\sigma(\tau) \sim 1/\Pr(S = 1)$ computed for the estimated classifer. We now confirm this dependency on a simple synthetic data generating process that allows us to vary the base rates in both outcome class $\Pr(Y)$ and protected class $\Pr(S)$. The details of the synthetic data and experimental setup are given in the Supplement. We use 10 times repeated 10-fold cross-validation on 20,000 data points and report the standard deviation of τ_{EFPR} across the replications.

Figure 3 shows how the standard deviation of the fairness metric changes with the base rate $\Pr(S = 1)$ for three different values of $\Pr(Y = 1)$. Each curve has the same qualitative shape consistent with an inverse dependence on $\Pr(S = 1)$. Our results support the theoretical analysis above that the variance in debiasing is strongly affected by class imbalances, both with respect to the class imbalance and the fraction of data points in the two protected attribute groups.

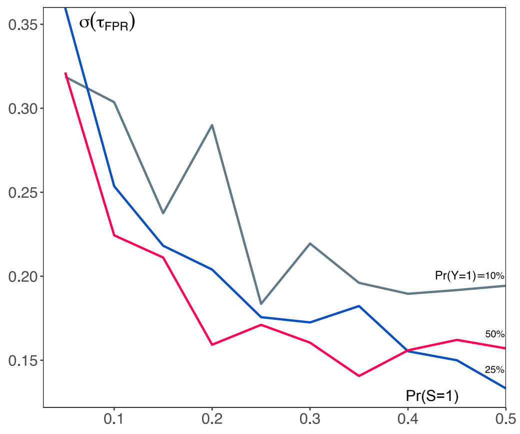


Figure 3: Standard deviations for τ_{EFPR} estimated via 10-fold CV across different fractions for protected class and positive class. Larger imbalances correspond to higher variance in the estimation of the fairness metric.

6 Partial debiasing

Theorem 1 implies that when considering the trade-off between performance and fairness, it is possible to construct a *variance-minimizing debiaser* that does not perfectly debias a model, but has better generalization properties. Minimizing the limiting variance (4) with respect to λ will in general not yield a full debiaser $\lambda = 1$, but rather some intermediate debiasing strength. This observation motivates our introduction of the notion of **partial debiasing** in this section. We will now describe two specific examples of partial debiasing.

Partial reweighing. The reweighing pre-processor of Section 1.2 can be easily generalized to yield a partial debiaser, simply by interpolating between the weight 1 (for $\lambda = 0$) and the weight $w_{h,i}$ for the fairness constraint h in Definition 2 (for $\lambda = 1$). The simplest such partial reweighing scheme is to simply perform linear interpolation, $w_i = (1-\lambda)1 + \lambda w_{h,i}$, although more exotic interpolation method could also be used.

Partial post-processing. Similarly, for post-processing methods like equalized odds (Section 1.2) and NLinProg (Section 2.2), we can define a partial debiasing scheme simply by interpolating the flipping probabilities $\Pr(\hat{Y}'|\hat{Y}=y, S=s)$ between 0 and their original values defined previously. Again for linear interpolation, this corresponds to replacing the flipping probabilities by $\lambda \Pr(\hat{Y}'|\hat{Y}, S)$.

As we show below, we find some surprising and nontrivial behaviors of this simple partial reweighing scheme, including the result that partial debiasing is in general preferable to full debiasing ($\lambda = 1$) to produce a low-variance debiased classifier.

6.1 Empirical evaluation of partial debiasing

We finish with another follow-up experiment, reporting out-of-bag metrics after 10-fold cross-validation and train an initial logistic regression model. We then debias this same model multiple times with respect to predictive parity (PP) fairness using three different partial debiasers (Section 6): partial equalized odds post-processing (EOd), partial reweighing (RW), and partial NLinProg (NLP) for accuracy and PP fairness. Figure 4 shows three different trajectories from parametrically increasing the debiasing strength from $\lambda = 0$ (no debiasing) to $\lambda = 1$ (full debiasing), starting from the initial model ($\lambda = 0$) at coordinates $(\gamma/\gamma_0, \tau/\tau_0) = (1, 1)$, again aggregated across 10CV10 folds.

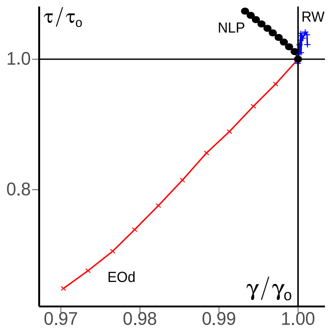


Figure 4: Accuracy–fairness plot of debiased models derived from a logistic regressor trained for the Adult income data set and debiased for PP fairness, showing parametric trajectories for increasing the debiasing strength λ from 0 to 1 the partial debiasers of Section 6.

As in our earlier experiments in Section 3, we see that the behavior of the different debiasers are markedly different. As before, it should be possible in theory to improve a classifier's fairness τ without compromising accuracy γ in this experiment. On the contrary, we observe that reweighing barely changes the metrics of the model, whereas EOd steadily leads to worsened PP unfairness and worse accuracy. The worsened fairness is to be expected, however, since we are debiasing with respect to a different metric that we are measuring. In contrast, debiasing and measuring the same metric of fairness in NLinProg leads to improved fairness τ, but at the expense of worsened accuracy γ. In this example, none of the debiased models come close to perfect fairness with metric $1/\tau_0 = 1.842$, implying that the training data and model family simply do not admit a perfectly fair classifier.

7 Conclusions and outlook

We have presented detailed empirical studies throughout the paper (especially Section 3) showing that classifiers treated with debiasing methods generally suffer from worse out-of-sample generalization behavior, so much so that the out-of-sample fairness can worsen relative to the original classifier. We need many test–train–validate splits to make a statistically significant determination of the treatment effect. As shown in the Supplement, the uncertainty in the fairness metric appears to be usually an order of magnitude larger than that for accuracy, which could reflect rare protected classes in Definition 2. We showed in Theorem 1 that this increased variance can be explained by bias–variance trade-off. To remove the statistical bias in the classifier that corresponds to discriminatory bias, we have to impose a fairness constraint, but satisfying that constraint increases the uncertainty of where the best decision boundary can be drawn, especially when the baseline model is already carefully estimated with attention paid to out-of-sample generalization error. Furthermore, we confirmed empirically in Section 5 that the estimation variance (4) is particularly severe when any of the protected classes is rare, i.e., when the base rate $\Pr(S)$ approaches zero. In practice, full debiasing is also not desirable if the performance of the debiased classifier degrades too much. We showed in Section 6 that the fine-grained control afforded by partial debiasing allows us to learn new classifiers that have desirable out-of-sample fairness properties.

The empirical results, while mostly negative, have motivated the theoretical analysis of Theorem 1, which gives us detailed insight into the origins of the large variance in classifiers. In particular, (4) states that the variance varies dramatically with protected class imbalance, which to our knowledge is a new result. Furthermore, (4) suggests that that partial debiasing can let us find a variance minimizing estimator that, while not applying the full debiasing treatment, can yield better fairness properties that generalize in practice. Since it is in general difficult to vary this trade-off parameter λ explicitly to find this estimator, finding practical ways to compute this minimal variance estimator seems like a promising research direction that could improve the practical utility of debiasing methods. Conversely, our results also show that fundamental limits exist to the ability to debias arbitrary models in a purely black box manner. Identifying underlying causal connections linking protected classes to features [54] may therefore be a more promising direction for successful mitigation of bias.

Our results signal caution to avoid the risk of fairwashing [1, 2], in the sense of believing that one is using a fair model resulting from some debiasing treatment, when in fact the model is overfit and does not generalize well out-of-sample [24]. Rather than blindly trusting that a debiased classifier is now fair, our results demonstrate that debiasing treatments need to be carefully tested in order to verify that the desired fairness properties hold in practice.

Acknowledgements

This work has been funded by the German Federal Ministry of Education and Research (BMBF) under Grant No. 01IS18036A. The authors of this work take full responsibilities for its content. We also thank our generous funding agencies IQVIA, UNIVERSITY of Auckland, Turing and tools practices, and Microsoft.

References

[1] U. Aïvodji, H. Arai, O. Fortineau, S. Gambs, S. Hara, and A. Tapp. Fairwashing: the risk of rationalization. In *Proceedings of the 36th International Conference on Machine Learning*, pages 161—-170, 2019. URL `http://arxiv.org/abs/1901.09749http://proceedings.mlr.press/v97/aivodji19a`.

[2] C. J. Anders, P. Pasliev, A.-K. Dombrowski, K.-R. Müller, and P. Kessel. Fairwashing explanations with off-manifold detergent. In *Proceedings of the 37th International Conference on Machine Learning*, pages 314–323, 2020. URL `http://arxiv.org/abs/2007.09969https://proceedings.mlr.press/v119/anders20a`.

[3] J. Angwin, J. Larson, S. Mattu, and L. Kichner. Machine bias, May 2016. URL `https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing`.

[4] C. Barabas, C. Doyle, J. Rubinovitz, and K. Dinakar. Studying up: Reorienting the study of algorithmic fairness around issues of power. In *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, FAT* '20, page 167–176, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450369367. doi: 10.1145/3351095.3372859. URL `https://doi.org/10.1145/3351095.3372859`.

[5] S. Barocas and A. Selbst. Big data's disparate impact. *California Law Review*, 104(1):671–729, 2016. doi: 10.15779/Z38BG31.

[6] R. K. E. Bellamy, K. Dey, M. Hind, S. C. Hoffman, S. Houde, K. Kannan, P. Lohia, J. Martino, S. Mehta, A. Mojsilović, S. Nagar, K. N. Ramamurthy, J. Richards, D. Saha, P. Sattigeri, M. Singh, K. R. Varshney, and Y. Zhang. AI Fairness 360: An extensible toolkit for detecting and mitigating algorithmic bias. *IBM Journal of Research and Development*, 63(4/5):4:1–15, 2019.

[7] R. Berk, H. Heidari, S. Jabbari, M. Kearns, and A. Roth. Fairness in criminal justice risk assessments: The state of the art. *Sociological Methods & Research*, Aug. 2018. doi: 10.1177/0049124118782533.

[8] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah. Julia: A fresh approach to numerical computing. *SIAM Review*, 59(1):65–98, 2017. doi: 10.1137/141000671.

[9] S. Bird, M. Dudík, R. Edgar, B. Horn, R. Lutz, V. Milan, M. Sameki, H. Wallach, and K. Walker. Fairlearn: A toolkit for assessing and improving fairness in AI. Technical Report MSR-TR-2020-32, Microsoft Research, Sept. 2020. URL `https://www.microsoft.com/en-us/research/publication/fairlearn-a-toolkit-for-assessing-and-improving-fairness-in-ai/`.

[10] S. Biswas and H. Rajan. *Do the Machine Learning Models on a Crowd Sourced Platform Exhibit Bias? An Empirical Study on Model Fairness*, pages 642–653. Association for Computing Machinery, New York, NY, USA, 2020. doi: 10.1145/3368089.3409704.

[11] L. Breiman. Random forests. *Machine Learning*, pages 5–32, 2001. doi: 10.1023/A:1010933404324.

[12] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Chapman & Hall/CRC, Boca Raton, FL, 1993.

[13] J. Buolamwini and T. Gebru. Gender shades: Intersectional accuracy disparities in commercial gender classification. *Proceedings of Machine Learning Research*, 81:77–91, 2018. URL `http://proceedings.mlr.press/v81/buolamwini18a.html`.

[14] T. Calders and S. Verwer. Three naive bayes approaches for discrimination-free classification. *Data Mining and Knowledge Discovery*, 21(2):277–292, 2010. doi: 10.1007/s10618-010-0190-x.

[15] J. Chen. Fair lending needs explainable models for responsible recommendation. In *Proceedings of the 2nd FATREC Workshop on Responsible Recommendation*, Sept. 2018.

[16] J. Chen, N. Kallus, X. Mao, G. Svacha, and M. Udell. Fairness under unawareness: Assessing disparity when protected class is unobserved. In *FAT* 2019 - Proceedings of the 2019 Conference on Fairness, Accountability, and Transparency*, pages 339–348, 2019. doi: 10.1145/3287560.3287594.

[17] A. Chouldechova. Fair prediction with disparate impact: A study of bias in recidivism prediction instruments. *Big Data*, 5(2):153–163, June 2017. doi: 10.1089/big.2016.0047.

[18] S. Corbett-Davies, E. Pierson, A. Feller, S. Goel, and A. Huq. Algorithmic decision making and the cost of fairness. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, page 797–806, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450348874. doi: 10.1145/3097983.3098095. URL `https://doi.org/10.1145/3097983.3098095`.

[19] A. Cotter, M. Gupta, H. Jiang, N. Srebro, K. Sridharan, S. Wang, B. Woodworth, and S. You. Training Well-Generalizing classifiers for fairness metrics and other Data-Dependent constraints. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 1397–1405. PMLR, 2019.

[20] L. Cupples, Q. Yang, S. Demissie, D. L. Copenhafer, and D. Levy. Description of the framingham heart study data for genetic analysis workshop 13. *BMC genetics*, 4 Suppl 1:S2, 02 2003. doi: 10.1186/1471-2156-4-S1-S2.

[21] D. Dua and C. Graff. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

[22] I. Dunning, J. Huchette, and M. Lubin. Jump: A modeling language for mathematical optimization. *SIAM Review*, 59(2):295–320, 2017. doi: 10.1137/15M1020575.

[23] A. Freeman. Racism in the credit card industry. *North Carolina Law Review*, 95(4):1071 – 1160, 2017. URL `https://ssrn.com/abstract=2976471`.

[24] S. A. Friedler, C. Scheidegger, S. Venkatasubramanian, S. Choudhary, E. P. Hamilton, and D. Roth. A comparative study of fairness-enhancing interventions in machine learning. In *Proceedings of the Conference on Fairness, Accountability, and Transparency - FAT* '19*, pages 329–338, New York, New York, USA, 2019. ACM Press. doi: 10.1145/3287560.3287589.

[25] M. Hardt, E. Price, and N. Srebro. Equality of opportunity in supervised learning. *Advances in Neural Information Processing Systems*, 29:3323–3331, Dec. 2016. doi: 10.5555/3157382.3157469. URL `https://papers.nips.cc/paper/6374-equality-of-opportunity-in-supervised-learning`.

[26] Information Commissioner's Office. Guidance on the ai auditing framework: Draft guidance for consultation (v1.0). Technical report, Information Commissioner's Office, Wilmslow, UK, Feb. 2020. URL `https://ico.org.uk/media/about-the-ico/consultations/2617219/guidance-on-the-ai-auditing-framework-draft-for-consultation.pdf`.

[27] N. Kallus, X. Mao, and A. Zhou. Assessing algorithmic fairness with unobserved protected class using data combination. In *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, page 110, New York, NY, USA, Jan. 2020. ACM. doi: 10.1145/3351095.3373154.

[28] F. Kamiran and T. Calders. Data preprocessing techniques for classification without discrimination. *Knowledge and Information Systems*, 33(1):1–33, 2012.

[29] T. Kamishima, S. Akaho, H. Asoh, and J. Sakuma. Fairness-aware classifier with prejudice remover regularizer. *Lecture Notes in Artificial Intelligence*, 7524(PART 2):35–50, 2012. doi: 10.1007/978-3-642-33486-3_3.

[30] J. S. Kim, J. Chen, and A. Talwalkar. Model-agnostic characterization of fairness trade-offs. In *Proceedings of the International Conference on Machine Learning*, volume 37, pages 9339–9349, 2020.

[31] J. Kleinberg, S. Mullainathan, and M. Raghavan. Inherent trade-offs in the fair determination of risk scores. In C. H. Papadimitriou, editor, *Proceedings of the 8th Innovations in Theoretical Computer Science Conference*, volume 67 of *Leibniz International Proceedings in Informatics (LIPIcs)*, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi: 10.4230/LIPIcs.ITCS.2017.43.

[32] R. Kohavi. Scaling up the accuracy of naive-bayes classifiers: a decision-tree hybrid. In *Proceedings of the 2nd. International Conference on Knowledge Discovery and Data Mining*, pages 202–207, Portland, Oregon, 1996. American Association for Artificial Intelligence.

[33] E. Kurshan, H. Shen, and J. Chen. Towards self-regulating AI: Challenges and opportunities of AI model governance in financial services. In *Proceedings of the 1st ACM International Conference on AI in Finance*, New York, NY, USA, 2020. ACM.

[34] L. T. Liu, S. Dean, E. Rolf, M. Simchowitz, and M. Hardt. Delayed impact of fair machine learning. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 6196–6200. AAAI Press, 2019.

[35] D. Mandal, S. Deng, S. Jana, J. Wing, and D. J. Hsu. Ensuring fairness beyond the training data. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 18445–18456. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper/2020/file/d6539d3b57159babf6a72e106beb45bd-Paper.pdf`.

[36] N. Mehrabi, F. Morstatter, N. Saxena, K. Lerman, and A. Galstyan. A survey on bias and fairness in machine learning, 2019.

[37] S. Mohamed, M.-T. Png, and W. Isaac. Decolonial ai: Decolonial theory as sociotechnical foresight in artificial intelligence. *Philosophy & Technology*, 405, 2020. doi: 10.1007/s13347-020-00405-8.

[38] A. Narayanan. Translation tutorial: 21 fairness definitions and their politics. In *Proceedings of the Conference on Fairness, Accountability and Transparency*, FAT* 18, New York, USA, 2018.

[39] V. Perrone, M. Donini, M. B. Zafar, R. Schmucker, K. Kenthapadi, and C. Archambeau. Fair Bayesian Optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence, Ethics and Society*, pages 1–15, jun 2020. URL `http://arxiv.org/abs/2006.05109`.

[40] G. Pleiss, M. Raghavan, F. Wu, J. Kleinberg, and K. Q. Weinberger. On fairness and calibration. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30, pages 5680–5689. Curran Associates, Red Hook, NY, 2017. URL `https://papers.nips.cc/paper/7151-on-fairness-and-calibration`.

[41] R. Richardson, J. M. Schultz, and K. Crawford. Dirty data, bad predictions: How civil rights violations impact police data, predictive policing systems, and justice. *New York University Law Review*, 94(2):192–233, 2019. URL `https://www.nyulawreview.org/online-features/dirty-data-bad-predictions-how-civil-rights-violations-impact-police-data-predictive-policing-systems-and-justice/`.

[42] K. T. Rodolfa, H. Lamba, and R. Ghani. Machine learning for public policy: Do we need to sacrifice accuracy to make models fair? *arXiv preprint arXiv:2012.02972*, 2020.

[43] C. Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215, 2019. doi: 10.1038/s42256-019-0048-x.

[44] B. Sadeghi. `DecisionTree.jl`, v0.10.10, Sept. 2020. URL `https://github.com/bensadeghi/DecisionTree.jl`.

[45] P. Saleiro, B. Kuester, A. Stevens, A. Anisfeld, L. Hinkson, J. London, and R. Ghani. Aequitas: A bias and fairness audit toolkit, 2018. URL `https://arxiv.org/abs/1811.05577`.

[46] S. Schelter, Y. He, J. Khilnani, and J. Stoyanovich. FairPrep: Promoting Data to a First-Class Citizen in Studies on Fairness-Enhancing Interventions. In *Proceedings of the 23nd International Conference on Extending Database Technology*, page 4, nov 2019. URL `http://arxiv.org/abs/1911.12587`.

[47] M. Sun and P. Yildaz. Understand ML model predictions and biases with Amazon SageMaker Clarify. In *AWS re:Invent*, 2020. URL `https://aws.amazon.com/sagemaker/clarify`.

[48] The Comptroller of the Currency, The Federal Reserve System, The Federal Deposit Insurance Corporation, The Consumer Financial Protection Bureau, and The National Credit Union Administration. Request for information and comment on financial institutions' use of artificial intelligence, including machine learning. *Federal Register*, pages 16837–16842, mar 2021.

[49] E. J. Topol. High-performance medicine: the convergence of human and artificial intelligence. *Nature Medicine*, 25(1):44–56, 2019. doi: 10.1038/s41591-018-0300-7.

[50] M. Turner and M. McBurnett. Predictive models with explanatory concepts: a general framework for explaining machine learning credit risk models that simultaneously increases predictive power. In *Proceedings of the 15th Credit Scoring and Credit Control Conference*, 2019. URL `https://crc.business-school.ed.ac.uk/wp-content/uploads/sites/55/2019/07/C12-Predictive-Models-with-Explanatory-Concepts-McBurnett.pdf`.

[51] S. Verma and J. Rubin. Fairness definitions explained. In *Proceedings of the International Conference on Software Engineering*, pages 1–7, New York, NY, USA, 2018. ACM. doi: 10.1145/3194770.3194776.

[52] A. Wächter and L. T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, 2006. doi: 10.1007/s10107-004-0559-y.

[53] A. Xiang and I. D. Raji. On the legal compatibility of fairness definitions. In *Workshop on Human-Centric Machine Learning at the 33rd Conference on Neural Information Processing Systems*, 2019.

[54] J. Zhang and E. Bareinboim. Fairness in decision-making: the causal explanation formula. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI'18)*, volume 32, pages 2037–2045, 2018. URL `https://ojs.aaai.org/index.php/AAAI/article/view/11564`.

5.2 Multi-objective counterfactual fairness

Contributed Article:
S. Dandl, F. Pfisterer, and B. Bischl. Multi-objective counterfactual fairness. In *GECCO '22: Proceedings of the Genetic and Evolutionary Computation Conference Companion*, page 328–331, Boston, United States of America, 2022. ACM

Declaration of contributions SD and FP contributed equally. The initial idea for the project initiated with FP based on previous work by SD [67]. FP wrote the initial manuscript together with SD and feedback by BB. FP and SD furthermore jointly implemented the proposed method and devised and executed the experiments. FP and SD jointly improved and revised the manuscript. BB provided guidance and feedback throughout the process.

Multi-Objective Counterfactual Fairness

Susanne Dandl*
LMU Munich
Munich, Germany

Florian Pfisterer*
LMU Munich
Munich, Germany

Bernd Bischl
LMU Munich
Munich, Germany

ABSTRACT

When machine learning is used to automate judgments, e.g. in areas like lending or crime prediction, incorrect decisions can lead to adverse effects for affected individuals. This occurs, e.g., if the data used to train these models is based on prior decisions that are unfairly skewed against specific subpopulations. If models should automate decision-making, they must account for these biases to prevent perpetuating or creating discriminatory practices. Counterfactual fairness audits models with respect to a notion of fairness that asks for equal outcomes between a decision made in the real world and a counterfactual world where the individual subject to a decision comes from a different protected demographic group. In this work, we propose a method to conduct such audits without access to the underlying causal structure of the data generating process by framing it as a multi-objective optimization task that can be efficiently solved using a genetic algorithm.

CCS CONCEPTS

• **Computing methodologies → Supervised learning by classification**; • **Mathematics of computing**;

KEYWORDS

machine learning, fairness, counterfactuals, multi-objective

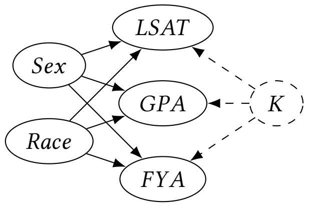
ACM Reference Format:
Susanne Dandl, Florian Pfisterer, and Bernd Bischl. 2022. Multi-Objective Counterfactual Fairness. In *Genetic and Evolutionary Computation Conference Companion (GECCO '22 Companion), July 9–13, 2022, Boston, MA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3520304.3528779

1 INTRODUCTION

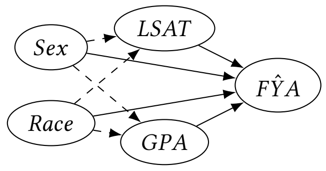
Machine learning (ML) is increasingly used to automate judgments in areas like lending, hiring, or predictive policing. Decisions made by such systems cannot only lead to adverse effects for affected individuals, but also shape future data that are collected (or not collected) [1], e.g., by not collecting data on individuals denied a loan. Such adverse effects are ethically or legally problematic when they disproportionately affect protected subgroups, e.g., based on race, gender, or sexual orientation. Several reasons lead to unfair predictions, such as a lack of representative data or differences in data quality between subgroups. We focus on a scenario where the labels used to train machine learning models are biased on prior decisions which are unfairly skewed against a specific subpopulation. If such biases exist in the data, models must take them into account in order to prevent such injustices.

Several contributions have addressed this topic and have argued that a causal perspective is required to address the problem [9, 16]. This has resulted in a variety of (causal) fairness notions [15, 16, 23] that can be used to audit fairness algorithms. Counterfactuals [20] provide a causal, interpretable perspective to answer *what-if* questions about alternative (counterfactual) worlds. From a perspective of fairness, this allows us to answer questions such as: *Would the model's prediction change if the person had been male instead of female?* This requires access to the underlying (causal) mechanism generating the data, e.g., in the form of a *directed acyclic graph* (DAG, c.f. [20]), which are often ambiguous, especially in the context of high dimensional data.

Introductory Example In order to provide some intuition, we use the law school example from [16]. The directed acyclic graph for the postulated data generating process is shown in Figure 1a. Sex, race as well as a latent variable *knowledge* (K) influence the result in the law school admission test (LSAT), GPA and the first-year average grade (FYA). Instantiating a counterfactual instance $\mathbf{x}^\star$ with, e.g., a changed variable *Sex* requires adapting the dependent variables *LSAT*, *GPA* and *FYA*. A ML model is now used to predict *FYA* from all other observed variables (Figure 1b). A fair model should now predict the same *FYA* regardless for $\mathbf{x}$ and $\mathbf{x}^\star$.



(a) DAG

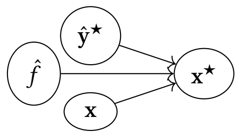


(b) Observational perspective

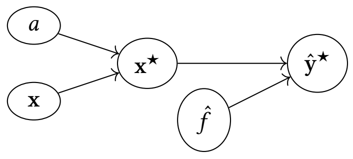
Figure 1: Law school example from [16].

Contributions: We propose a method to audit predictive models with respect to a fairness notion that relies on counterfactuals. Counterfactuals are found as solutions to a multi-objective optimization procedure, inspired by [7]. We argue that we can find realistic counterfactual examples by carefully crafting the objectives used for optimization. Due to the flexibility of the evolutionary algorithm used to tackle the resulting optimization problem, we can furthermore incorporate additional constraints in the optimization problem, allowing to attain more realistic and actionable counterfactuals. Unlike other methods, the multi-objective nature of our optimization problem allows us to return a Pareto-optimal set of diverse counterfactuals that can be used to assess fairness. Our

*Both authors contributed equally to this research.



(a) Counterfactual Explanations



(b) Counterfactual Fairness

Figure 2: Generating counterfactuals $\mathbf{x}^\star$ as explanations (CFE) (left) and for fairness (CFF) (right) for an observations $\mathbf{x}$ and predictor $\hat{f}$. The role of counterfactual prediction $\hat{y}^\star$ differs in both cases: While $\hat{y}^\star$ is incorporated into the generation of counterfactuals for CFEs, in CFF, the counterfactuals are first generated by striving for a different protected class a, and subsequently their counterfactual predictions are compared.

method does not require access to the underlying causal DAG and can therefore be used when such information is not available.

2 RELATED WORK

Fairness broadly asks that there is no *disproportionate* treatment between individuals depending on protected groups such as race, gender, or sexual orientation. A large body of work has previously studied differing notions of fairness [1, 18], often based on subgroup statistics in observational data [2, 4, 6, 13], while other notions of fairness argue to *treat similar persons similarly* [11] or argue for taking a causal perspective into account [5, 15, 16]. We follow the line of argumentation proposed in [16], which argues that the distribution over predictions should remain unchanged between the observed universe and a *counterfactual* universe in which an individual has different protected attributes. While [16] propose an algorithm that implements this definition, it requires access to the underlying DAG. One line of work implements notions similar to ours that do not require access, such as *FlipTest* [3], which uses a generative model approximating an optimal transport mapping to generate counterfactuals.

The notion of counterfactuals has been similarly used to improve *model interpretability*, answering which change in inputs would lead to a different model prediction [22]. These methods can generate potentially unrealistic out-of-distribution samples, which can jeopardize derived conclusions. For this reason, methods were proposed [7, 21] which focus on generating *plausible* counterfactuals. This is especially important in the context of algorithmic recourse. Karimi et al. [14] argue that explanations should be *actionable* but also *realistic* in the sense that they take into account the (causal) structure of the world from which they are obtained. This scenario differs from counterfactual fairness, since it aims at counterfactuals that lead to different model predictions. In contrast, counterfactual fairness notions observe the amount of change in a prediction from an instance to its counterfactual example. This difference is visualized in Figure 2.

Our method is heavily inspired by the MOC method described in [7], which was proposed in the context of finding multiple counterfactual explanations. In contrast, our method is used to find realistic counterfactual examples that allow auditing ML models with respect to counterfactual fairness for individual observations; when applied to multiple observations, we could also obtain a global assessment. We similarly formulate a multi-objective optimization problem that can be efficiently solved using evolutionary algorithms. In order for our counterfactuals to be realistic and actionable, we carefully craft objectives and mutation operators used in the search.

3 METHODOLOGY

Let $\hat{f}(\mathbf{x}) : \mathcal{X} \mapsto \mathbb{R}$ denote a model fitted to approximate the relationship between features $\mathbf{x}$ and a target variable of interest $\mathbf{y}$, which are i.i.d. samples from a data generating distribution $\mathbb{P}_{xy}$. We assume that our data contain feature(s) A defining the protected class and define $Z \equiv X \setminus A$ as the set of all other observable features. For a data point $\mathbf{x}$, we define a counterfactual observation as $\mathbf{x}^\star$ with prediction $\hat{y}^\star := \hat{f}(\mathbf{x}^\star)$. Counterfactuals that arise from intervention $A \leftarrow a$ could equivalently be denoted as $\mathbf{x}_{A\leftarrow a}$ [20]. For ease of exposition, we restrict ourselves to classification models that predict probabilities throughout the manuscript. Extensions to regression models are straightforward once prediction thresholds are specified.

3.1 Counterfactual Fairness

We first restate the definition of counterfactual fairness from [16]. It assumes a causal model (U, X, F), with U as a set of latent background variables not caused by any observed variables X, and F as a set of causal equations. $\hat{Y}$ denotes a predictor that contrary to $\hat{f}$ depends on X *and* U. The resulting $\hat{Y}$ for intervention $A \leftarrow a$ is denoted as $\hat{Y}_{A\leftarrow a}(U)$.

Definition 1 (Counterfactual fairness [16]). *Predictor $\hat{Y}$ is* **counterfactually fair** *if under arbitrary context $Z = \mathbf{z}$ and $A = a$,*

$$P(\hat{Y}_{A\leftarrow a}(U) = y \mid Z = \mathbf{z}, A = a) = P(\hat{Y}_{A\leftarrow a'}(U) = y \mid Z = \mathbf{z}, A = a),$$

for all y and for any value a' attainable by A.

This suggests that changing A while keeping features that are not causally reliant on A constant has no effect on the distribution of Y. The computation of U and $\hat{Y}_{A\leftarrow a}$ is complex and requires access to the underlying DAG. We therefore state a similar criterion below that is practically applicable without access to the DAG. Note that the counterfactual instance is not necessarily *deterministic*, and the desired counterfactual can stem from a distribution of counterfactual instances.

3.2 A Practical Instantiation

In practical scenarios without access to the DAG, there is little chance to recover U. More realistically, our model uses $\mathbf{x}$ to predict the outcome of interest. Instead, we can therefore ask that the equality in Definition 1 holds between a data point $\mathbf{x}$ and its counterfactual $\mathbf{x}^\star$. We now state a version of counterfactual fairness that can be practically applied to observational data:

Definition 2 (Counterfactual fairness in practice). *Predictor $\hat{Y}$ is* **counterfactually fair** *if under any context $Z = \mathbf{z}$ and $A = a$,*

$$P(\hat{f}(\mathbf{x}_{A\leftarrow a}) = y | Z = \mathbf{z}, A = a) = P(\hat{f}(\mathbf{x}_{A\leftarrow a'}) = y | Z = \mathbf{z}, A = a)$$

for all y and for any value a' attainable by A.

3.3 Generating Counterfactuals

The remaining task is now to generate counterfactuals $\mathbf{x}^\star := \mathbf{x}_{A\leftarrow a'}$ which should fulfill the following requirements: (1) the counterfactual should be **valid**, such that it has high likelihood w.r.t. the distribution of the desired protected class $P_{X_{A=a'}}$; (2) the counterfactual should be **close** to the original observation; (3) the counterfactual should be **plausible** such that it lies in a high-density region w.r.t. the full dataset. Similar to [7], we translate our customized requirements into the following optimization problem:

$$\min_{\mathbf{x}^\star} \mathbf{o}(\mathbf{x}^\star) := \min_{\mathbf{x}} \left(o_{valid}(\mathbf{x}^\star),\ o_{close}(\mathbf{x}^\star, \mathbf{x}), o_{plaus}(\mathbf{x}^\star, \mathbf{X}^{obs})\right)$$

with $\mathbf{o} : \mathcal{X} \rightarrow \mathbb{R}^3$ and $\mathbf{X}^{obs}$ being the observed data.

The first objective o_{valid} quantifies whether $\mathbf{x}^\star$ truly stems from the desired protected group a'. We operationalize it for minimization using an additional predictor $\hat{g}$ that is trained to predict whether a datapoint $\mathbf{x}^\star$ does not belong to the protected group a'.

$$o_{valid}(\mathbf{x}^\star) = \hat{g}(\mathbf{x}^\star)$$

The second and third objectives o_{close} and o_{plaus} are similar to the ones proposed by [7]. o_{close} quantifies the distance between the counterfactual $\mathbf{x}^\star$ and the original datapoint $\mathbf{x}$ using an *augmentation* of the Gower distance (see c.f. [7]).

The third objective o_{plaus} quantifies the weighted average Gower distance between $\mathbf{x}^\star$ and the k nearest observed data points $\mathbf{x}^{[1]}, ..., \mathbf{x}^{[k]} \in \mathbf{X}^{obs}$ as an empirical approximation of how likely $\mathbf{x}^\star$ originates from the distribution of $\mathcal{X}$:

$$o_{plaus}(\mathbf{x}^\star, \mathbf{X}^{obs}) = \sum_{i=1}^{k} w^{[i]} \frac{1}{p} \sum_{j=1}^{p} \delta_G(x_j^\star, x_j^{[i]}) \in [0, 1]$$

where $\sum_{i=1}^{k} w^{[i]} = 1$. We optimize counterfactuals using an NSGA-II [8] variant adapted to the scenario of generating counterfactual instances proposed by [7], including their described modifications. The algorithm uses nature-inspired methods such as selection, mutation and recombination to steer a randomly initialized population towards the optimal solution (see Appendix A for details). This yields a set of Pareto-optimal counterfactuals that can be subsequently used to evaluate algorithms with respect to our practical notion of counterfactual fairness. The Pareto set can be interpreted as a distribution over counterfactuals (as defined by the objectives), reflecting the fact that *real* counterfactuals can be stochastic due to stochasticity in the data generating process as well as uncertainty in the estimation of required quantities.

Since we seek counterfactuals with a high likelihood of coming from the distribution of the desired protected class $P_{X_{A=a'}}$, we base the fairness notions of Section 4 on samples with high values of o_{valid} letting the user define a lower threshold for o_{valid}. We assume that this Pareto-optimal and valid subset approximates the distribution over counterfactuals for a single data point $\mathbf{x}$.

Actionable Counterfactuals. By defining additional customized operators or objectives (e.g., sparsity constraints), our method can be further adapted to more closely reflect the real-world data generating processes. This includes carefully designed mutation operators that constrain the allowable changes to features: values for non-actionable features (e.g., age) could be frozen, or monotonicity constraints could be considered such that an increase in one feature leads to an increase or decrease in another feature [19]. Furthermore, we can accelerate the convergence to the Pareto front by initializing the first population of the NSGA-II with observations from $\mathbf{X}^{obs}$ with $A = a'$. These observations per definition should have low values both for o_{valid} and o_{plaus}.

3.4 Evaluating for Counterfactual Fairness

A counterfactual generation procedure $gen : \mathcal{X} \rightarrow \mathcal{X}^\star$ (such as the one proposed above) turns an instance $\mathbf{x}$ into a set of counterfactual instances $\mathbf{X}^\star$. We now define fairness criteria based on generated counterfactuals:

Definition 3 (Instance-wise counterfactual unfairness). *For a single individual* $\mathbf{x}$ *and a set of corresponding generated counterfactuals* $\mathbf{X}^\star$*, we define unfairness as:*

$$icuf(\mathbf{x}) = |\mathbb{E}_{\mathbf{x}^\star \sim gen(\mathbf{x})}[\hat{f}(\mathbf{x}) - \hat{f}(\mathbf{x}^\star)]|.$$

Computing the norm reflects the fact, that our notion does not differentiate between the direction of the unfairness (e.g., if $\hat{f}$ favors or disadvantages the individual).

Definition 4 (Global counterfactual unfairness). *For a distribution over datapoints* $\mathcal{X}$ *and a set of sets of corresponding generated counterfactuals* $\mathcal{X}^\star$*, we define a global notion of unfairness:*

$$gcuf(\mathcal{X}) = \mathbb{E}_{\mathbf{x} \sim \mathcal{X}}[icuf(\mathbf{x})].$$

Taking the expectation simultaneously reduces variance in the estimation and results in more robust estimates. Note that $\hat{f}$ for our purposes can be a predicted probability. By thresholding predictions, we can simultaneously obtain *FlipSets* – the set of points for which the classification switches between the original instance and the counterfactual – and subsequently create *transparency reports* [3].

4 EMPIRICAL EVALUATION

Our goal is to create realistic counterfactuals. We therefore use the data generating process (DGP) of the *law school dataset* from [16] to generate data and *true* counterfactuals $\mathbf{x}'$, while we present results for another dataset in the supplementary material. We describe experimental details in Appendix B.

RQ1: Does our method generate realistic counterfactuals?
We present a visual comparison using t-SNE embeddings in Figure 3. Generated counterfactuals are found in high-density regions of the data and close to instances of the desired class. The true counterfactual is surrounded by generated counterfactuals. The average minimum Gower distance between $\mathbf{x}^\star$ and $\mathbf{x}'$ is 0.069. We further quantify this in Table 1 by comparing our counterfactuals $\mathbf{x}^\star$ to two simple baselines: x^{nn}, the nearest neighbor of x with desired protected attribute a' and x^{rnd}, a random observation. Distances between generated counterfactuals are typically lower than random points, while distances between an instance and the true counterfactual are comparatively high.

RQ2: How does fairness reported by our method compare to simple baselines?
To investigate the faithfulness of our method and several baselines, we calculate their $gucf$ to the one of true counterfactuals. Individual values as well as further experiments are reported in the supplementary material. Table 2 reports $gcuf$ across several baselines and

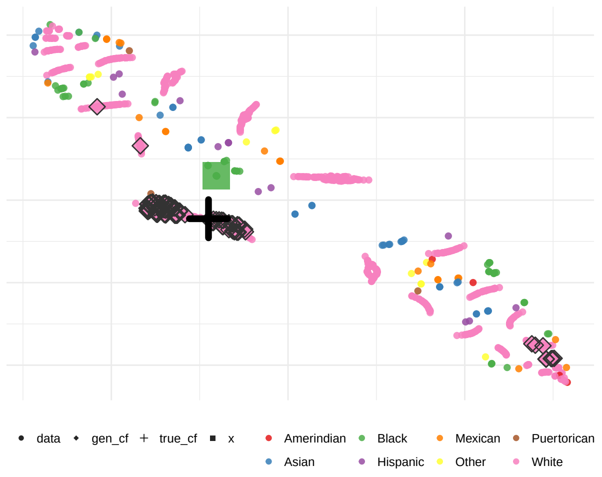


Figure 3: t-SNE plot for an instance of the law school DGP.

Table 1: Average Gower distances between x (original instance), $x^{\star}$ (generated counterfactual), x' (true counterfactual) and x^{rnd} (random point) and xnn (nearest neighbor).

$d(x, x')$	$d(x, x^{\star})$	$d(x, x^{rnd})$	$d(x, x^{nn})$
0.16	0.07	0.192	0.008

Table 2: Mean $gcuf$ measured using true counterfactuals and different generation methods: The proposed method (ours) and two baselines: $flip$, flipping the protected attribute $A = a'$ in X, and nn, the nearest neighbors with $A = a'$.

$gcuf_{true}$	$gcuf_{ours}$	$gcuf_{flip}$	$gcuf_{nn}$
0.277 ± .003	0.278 ± .003	0.265 ± .004	0.318 ± .004

$gcuf$ obtained using true counterfactuals. Reported values using $\mathbf{x}^{\star}$ are considerably closer to values estimated for true counterfactuals.

5 OUTLOOK

This manuscript proposes and evaluates a method for evaluating predictive models with respect to a counterfactual notion of individual and global fairness. Our method does not require access to the DAG generating the data, accounts for stochasticity by returning a Pareto-optimal set of counterfactuals, and is flexible enough for adoption to the needs of individual use cases. It is important to note that the validity of fairness auditing as proposed in our method heavily relies on the validity of generated counterfactuals, which is discussed in detail in Appendix C. In future work, we would like to improve the procedure used to find counterfactuals for a set of instances. The current procedure requires an inefficient loop across N observations that can hopefully be expedited by further tweaks to the optimization procedure. In a different line of work, we want to incorporate *path-based* notions of counterfactual fairness [5], which would allow for the definition of fair paths determined, e.g., due to principles such as business necessity (c.f. [12]).

ACKNOWLEDGMENTS

This work is funded by the German Federal Ministry of Education and Research (BMBF) under Grant No. 01IS18036A. The authors of this work take full responsibilities for its content.

REFERENCES

[1] Solon Barocas, Moritz Hardt, and Arvind Narayanan. 2019. *Fairness and Machine Learning*. fairmlbook.org. http://www.fairmlbook.org.

[2] Richard Berk, Hoda Heidari, Shahin Jabbari, Michael Kearns, and Aaron Roth. 2018. Fairness in Criminal Justice Risk Assessments: The State of the Art. *Sociological Methods & Research* (Aug. 2018), 42 pages. https://doi.org/10.1177/0049124118782533 arXiv:1703.09207

[3] Emily Black, Samuel Yeom, and Matt Fredrikson. 2020. Fliptest: fairness testing via optimal transport. In *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*. 111–121.

[4] Toon Calders and Sicco Verwer. 2010. Three naive Bayes approaches for discrimination-free classification. *Data Mining and Knowledge Discovery* 21, 2 (2010), 277–292. https://doi.org/10.1007/s10618-010-0190-x

[5] Silvia Chiappa. 2019. Path-specific counterfactual fairness. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 7801–7808.

[6] Alexandra Chouldechova. 2017. Fair Prediction with Disparate Impact: A Study of Bias in Recidivism Prediction Instruments. *Big Data* 5, 2 (June 2017), 153–163. https://doi.org/10.1089/big.2016.0047 arXiv:1703.00056

[7] Susanne Dandl, Christoph Molnar, Martin Binder, and Bernd Bischl. 2020. Multi-objective counterfactual explanations. In *International Conference on Parallel Problem Solving from Nature*. Springer, 448–469.

[8] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. 2002. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6, 2 (April 2002), 182–197. https://doi.org/10.1109/4235.996017

[9] Simon DeDeo. 2014. Wrong side of the tracks: Big data and protected categories. *arXiv preprint arXiv:1412.4643* (2014).

[10] Dheeru Dua and Casey Graff. 2017. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml

[11] Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Richard Zemel. 2012. Fairness through awareness. In *Proceedings of the 3rd innovations in theoretical computer science conference*. 214–226.

[12] Susan S Grover. 1995. The business necessity defense in disparate impact discrimination cases. *Ga. L. Rev.* 30 (1995), 387.

[13] Moritz Hardt, Eric Price, and Nathan Srebro. 2016. Equality of opportunity in supervised learning. *Advances in Neural Information Processing Systems* 29 (Dec. 2016), 3323–3331. https://doi.org/10.5555/3157382.3157469 arXiv:1610.02413

[14] Amir-Hossein Karimi, Bernhard Schölkopf, and Isabel Valera. 2021. Algorithmic recourse: from counterfactual explanations to interventions. In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*. 353–362.

[15] Niki Kilbertus, Mateo Rojas-Carulla, Giambattista Parascandolo, Moritz Hardt, Dominik Janzing, and Bernhard Schölkopf. 2017. Avoiding discrimination through causal reasoning. *arXiv preprint arXiv:1706.02744* (2017).

[16] Matt J Kusner, Joshua R Loftus, Chris Russell, and Ricardo Silva. 2017. Counterfactual fairness. *arXiv preprint arXiv:1703.06856* (2017).

[17] Rui Li, Michael T.M. Emmerich, Jeroen Eggermont, Thomas Bäck, M. Schütz, J. Dijkstra, and J. H.C. Reiber. 2013. Mixed Integer Evolution Strategies for Parameter Optimization. *Evolutionary Computation* 21, 1 (2013), 29–64.

[18] Ninareh Mehrabi, Fred Morstatter, Nripsuta Saxena, Kristina Lerman, and Aram Galstyan. 2021. A Survey on Bias and Fairness in Machine Learning. 54, 6, Article 115 (jul 2021), 35 pages. https://doi.org/10.1145/3457607

[19] Ramaravind K. Mothilal, Amit Sharma, and Chenhao Tan. 2020. Explaining Machine Learning Classifiers through Diverse Counterfactual Explanations. In *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency* (Barcelona, Spain) *(FAT* '20)*. Association for Computing Machinery, New York, NY, USA, 607–617. https://doi.org/10.1145/3351095.3372850

[20] Judea Pearl. 2009. Causal inference in statistics: An overview. *Statistics surveys* 3 (2009), 96–146.

[21] Rafael Poyiadzi, Kacper Sokol, Raul Santos-Rodriguez, Tijl De Bie, and Peter Flach. 2020. *FACE: Feasible and Actionable Counterfactual Explanations*. Association for Computing Machinery, New York, NY, USA, 344–350. https://doi.org/10.1145/3375627.3375850

[22] Sandra Wachter, Brent Daniel Mittelstadt, and Chris Russell. 2018. Counterfactual explanations without opening the black box: automated decisions and the GDPR. *Harvard Journal of Law and Technology* 31, 2 (2018), 841–887.

[23] Junzhe Zhang and Elias Bareinboim. 2018. Equality of Opportunity in Classification: A Causal Approach. In *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.), Vol. 31. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2018/file/ff1418e8cc993fe8abcfe3ce2003e5c5-Paper.pdf

A NSGA-II

NSGA-II [8] first initializes a random set of candidates (in our case counterfactual instances) which are evaluated by the proposed objectives. The best candidates are recombined in pairs and then slightly mutated to generate new candidates. Old and new candidates are ranked according to their objective values using non-dominated sorting and crowding distance sorting. The first aims at optimality, the second at diversity of the objective values. Based on this ranking, the best candidates are selected for the next generation. In subsequent generations, recombination, mutation and selection are repeated based on the updated population. In the end, the Pareto optimal set over all candidates is returned. Compared to the originally proposed NSGA-II, the method by Dandl et al. [7] uses mutation and recombination methods [17] to cover mixed (discrete and continuous) search spaces, and a crowding distance sorting that additionally considers diversity in the feature space.

B EXPERIMENTAL DETAILS

The goal of the experimental evaluation is two-fold: Since fairness metrics $icuf$ (Definition 3) and $gcuf$ (Definition 4) rely on the assumption that generated counterfactuals are realistic, we investigate this assumption in downstream experiments based on the *adult* dataset [10]. Simultaneously, our ultimate goal is to check for *instance-wise* or *global* unfairness, therefore, we also need to ascertain that our numeric estimates of unfairness correspond to the real unfairness. The latter can only be observed in scenarios where true counterfactuals are observable – which is not the case for the *adult* dataset. Therefore, we investigate our goals in a simulation scenario based on the law school example described in the introduction. The code to reproduce all experiments is available in a GitHub repository: https://github.com/pfistfl/counterfactuals/tree/moccf/paper/experiments. Optimization is generally run for ≤ 30 generations of the adapted NSGA-II algorithm. Generating counterfactuals for a single instance generally takes around 15 seconds for 30 generations.

Quality of generated counterfactuals

We generate the counterfactual for a given instance $\mathbf{x}$ and use t-SNE embeddings to visualize the generated counterfactuals $\mathbf{x}^\star \in \mathbf{X}^\star$. We visually judge the quality of generated counterfactuals using the following criteria:

- $\mathbf{x}^\star$ should lie in high-density regions of the data.
- $\mathbf{x}^\star$ should lie in high-density regions for samples of $\mathbf{X}$ with the desired protected status.
- $\mathbf{x}^\star$ should be close to the original instance $\mathbf{x}$.

Adult. We trained a random forest model on the first 1000 samples of the *adult* dataset [10]. As a preprocessing step, we combined categories of the protected attribute race with few observations such that we receive three categories (*White*, *Black* and *Other*). For an instance with race *Black*, we generated counterfactuals $\mathbf{x}_{A \leftarrow White}$. Figure 5 of the Pareto front reveals that the three objectives contradict each other, e.g., counterfactuals with low values in o_{valid} or o_{close} have higher values in o_{plaus}. The t-SNE embeddings in Figure 4 show that generated counterfactuals are found in high-density regions of the data and close to instances of the desired class.

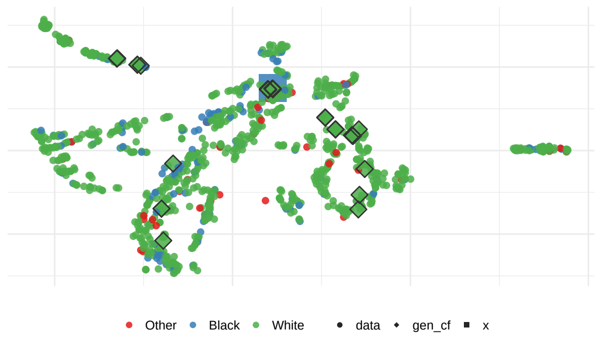


Figure 4: t-SNE plot for the adult dataset after 175 generations.

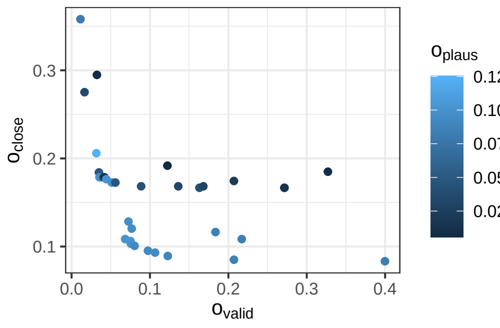
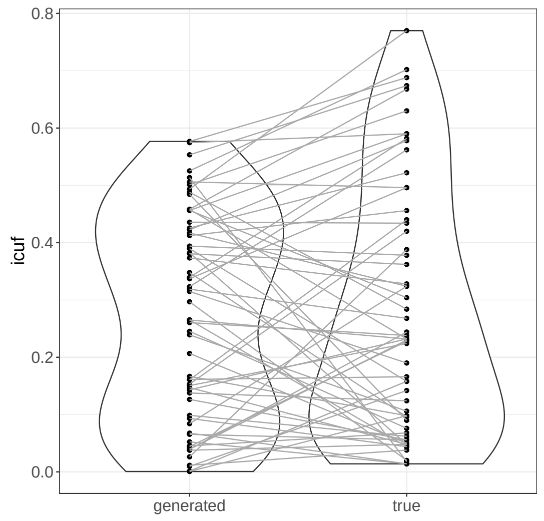


Figure 5: Plot of the Pareto front for the adult dataset after 175 generations.

Law School. We draw 1000 samples from the data generating process as described in [16] and detailed in the introduction. We then investigate the counterfactuals $\mathbf{x}' := \mathbf{x}_{A \leftarrow White}$ for all instances in X with race *Black*. We furthermore use the FYA variable in order to estimate a variable $PASS$ (indicating whether a student will pass), where $PASS_{(i)} \sim Ber(logit(FYA_{(i)}))$ for each respective instance i. Given access to the *true* counterfactual $\mathbf{x}'$, we can furthermore assess how close $\mathbf{x}^\star \in \mathbf{X}^\star$ lie to $\mathbf{x}'$ for example given the Gower distance. Results reported in Figure 3 are for a single instance, while distances reported in Table 1 are averaged across all instances with label *Black*. We did not include the protected attribute for calculating Gower distances.

Individual and global unfairness

We investigate global and individual level unfairness based on the law school example described in the introduction. We use the same experimental setup as described above. Since we have access to the data generating process in this simulated scenario, we can generate the true counterfactuals as well as counterfactuals generated using



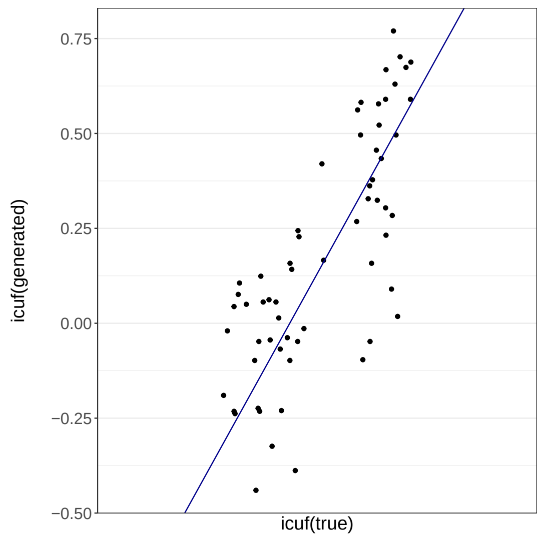


Figure 6: *Upper*: Comparison of *icuf* between generated counterfactuals (left) and true counterfactuals (right) for the law school example. The global *gcuf* is 0.268 **and** 0.319 **respectively. *Lower*: Scatterplot of *icuf* for generated counterfactuals ($\mathbf{x}^\star$) and true counterfactuals ($\mathbf{x}'$) for the law school example.**

the proposed method. The resulting $icuf$ and $gcuf$ for both true counterfactuals (right) and generated counterfactuals (left) are reported in Figure 6. While $icuf$ is slightly underestimated, the global estimate of model unfairness (0.268) is reasonably close to the true one (0.319).

C ASSUMPTIONS AND VALIDITY OF GENERATED COUNTERFACTUALS

The goal of this work is to propose an alternative method for fairness auditing of machine learning models. In contrast to existing methods for observational data (cf. [13]), our method hopes to generate *causally valid* counterfactuals. In the absence of an unambiguous DAG, there can be no guarantees that any generated counterfactual actually stems from the true distribution of counterfactuals – at best we can hope that we generate sufficiently similar datapoints given the specified objectives. Thus, we argue that our method (as well as other methods proposed in this context) should never be used in isolation, but as one additional perspective to detect potential biases in data. It is similarly important to consider fairness in its broader context, i.e., the actual outcomes that decisions based on ML models produce and their long-term effects, e.g., in the context of feedback loops. Furthermore, the question of whether a technical intervention in favor of possible other solutions is necessary for a given context needs to be thoroughly considered.

CHAPTER 6

CONTRIBUTIONS - BENCHMARKS & SOFTWARE

6.1 Benchmarking time series classification – Functional data vs machine learning approaches

Contributed Article:
F. Pfisterer, L. Beggel, X. Sun, F. Scheipl, and B. Bischl. Benchmarking time series classification – functional data vs machine learning approaches, 2019, arXiv:1911.07511

Declaration of contributions BB initiated the project, with help on the first architecture design and data flow implementation by XS and LB, while FP and BB refactored and implemented the current version of the software architecture. LB independently provided the repository of data sets for the benchmark and gathered the results of the benchmark experiments of [12]. LB implemented the first version of several classification algorithms for mlrFDA, followed by a refactoring of the code by FP. XS implemented the first version of several functional feature extraction algorithms used in mlrFDA, including bsignal, Dynamic Time Warping (multiple reference input), multiresolution (proposed by BB with major code snippet), FP refactored these codes and added several new feature extraction methods. XS implemented the first version of classification and regression of FDboost, FGAM, with the help of FS, followed by refactor by FP. LB implemented the first version of Fourier and Wavelets feature extraction; FP and XS refactored the code. XS implemented the first version of the Benchmark code and conducted a first batch of experiments, with LB analyzing the results. FP refactored the benchmark code and rerun the experiment and collected the experiment results independently.

Benchmarking time series classification - Functional data vs machine learning approaches

Florian Pfisterer, Xudong Sun[1], Laura Beggel[1], Fabian Scheipl, Bernd Bischl

Department of Statistics, Ludwig-Maximilians-Universität München
Ludwigstr. 33,
80539, München, Germany

Abstract

Time series classification problems have drawn increasing attention in the machine learning and statistical community. Closely related is the field of functional data analysis (FDA): it refers to the range of problems that deal with the analysis of data that is continuously indexed over some domain. While often employing different methods, both fields strive to answer similar questions, a common example being classification or regression problems with functional covariates. We study methods from functional data analysis, such as functional generalized additive models, as well as functionality to concatenate (functional) feature extraction or basis representations with traditional machine learning algorithms like support vector machines or classification trees. In order to assess the methods and implementations, we run a benchmark on a wide variety of representative (time series) data sets, with in-depth analysis of empirical results, and strive to provide a reference ranking for which method(s) to use for non-expert practitioners. Additionally we provide a software framework in R for functional data analysis for supervised learning, including machine learning and more linear approaches from statistics. This allows convenient access, and in connection with the machine-learning toolbox *mlr*, those methods can now also be tuned and benchmarked.

Keywords: Functional Data Analysis, Time Series, Classification, Regression

[*]source code available at `https://github.com/mlr-org/mlr`
[*]Corresponding author: florian.pfisterer@stat.uni-muenchen.de
[1]Equal Contribution

1. Introduction

The analysis of functional data is becoming more and more important in many areas of application such as medicine, economics, or geology (cf. Ullah and Finch [1], Wang et al. [2]), where this type of data occurs naturally. In industry, functional data are often a by-product of continuous monitoring of production processes, yielding great potential for data mining tasks. A common type of functional data are time series, as time series can often be considered as discretized functions over time.

Many researchers publish software implementations of their algorithms, therefore simplifying the access to already established methods. Even though such a readily available, broad range of methods to choose from is desirable in general, it also makes it harder for non-expert users to decide which method to apply to a problem at hand and to figure out how to optimize their performance. As a result, there is an increasing demand for automated model selection and parameter tuning.

Furthermore, the functionality of available pipeline steps ranges from simple data structures for functional data, to feature extraction methods and packages offering direct modeling procedures for regression and classification. Users are again faced with a multiplicity of software implementations to choose from and, in many instances, combining several implementations may be required. This can be difficult and time-consuming, since the various implementations utilize a multiplicity of different workflows which the user needs to become familiar with and synchronize in order to correctly carry out the desired analysis.

There is a wide variety of packages for functional data analysis in R [3] available that provide functionality for analyzing functional data. Examples range from the **fda** [4] package which includes object types for functional data and allows for smoothing and simple regression, to, e.g., boosted additive regression models for functional data in **FDboost** [5]. For an extensive overview, see the CRAN task view [6].

Many of those packages are designed to provide algorithmic solutions for one specific problem, and each of them requires the user to become familiar with its user interface. Some of the packages, however, such as **fda.usc** [7] or **refund** [8] are not designed for only one specific analysis task, but combine several approaches. Nevertheless, these packages do not offer unified frameworks or consistent user interfaces for their various methods, and most of the packages can still only be applied separately.

A crucial advantage of providing several algorithms in one package with a unified and principled user interface is that it becomes much easier to compare the provided methods with the intention to find the best solution for a problem at hand. But to determine the best alternative, one still has to be able to compare the methods at their best performance on the considered data, which requires hyperparameter search and, more preferably, efficient tuning methods.

While the different underlying packages are often difficult and sometimes even impossible to extend to new methods, custom implementations and extensions can be easily included in the accompanying software.

We want to stress that the focus of this paper does not lie in proposing new algorithms for functional data analysis. Its added value lies in a large comparison of algorithms while providing a unified and easily accessible interface for combining statistical methods for functional data with the broad range of functions provided by **mlr**, most importantly benchmarking and tuning. Additionally, the often overlooked possibility of extracting non-functional features from functional data is integrated, which enables the user to apply classical machine learning algorithms such as *support vector machines* [9] to functional data problems.
In a benchmark study similar to Bagnall et al. [10] and Fawaz et al. [11], we explore the performance of implemented methods, and try to answer the following questions:

1. Can functional data problems be solved with classical machine learning methods ignoring the functional structure of the data as well as with more elaborate methods designed for this type of data? Bagnall et al. [10] measure the performance of some non-functional-data-specific algorithms such as the rotation forest [12], but this does not yield a complete picture.
2. Guidance on the wide range of available algorithms is often hard to obtain. We aim to make some recommendations in order to simplify the choice of learning algorithm.
3. Do statistical methods explicitly tailored to the analysis of functional data [e.g. **FDboost**, 5] perform well on classical time series tasks? No benchmark results for these methods, which provide interpretable results, are currently available.
4. Many methods that represent functional data in a non-functional domain have been proposed and are also often applied in practice. Examples for this include either hand crafted features [cf. 13], summary statistics [14], or generally applicable methods such as wavelet decomposition [15].
5. Hyperparameter optimization is a very important step in many machine learning applications. In our benchmark, we aim to quantify the impact of hyperparameter optimization for a set of given algorithms on several data sets.

Contributions. As contributions of this paper, we aim to answer the questions posed above. Additionally, we provide a toolbox for the analysis of functional data. It implements several methods for feature extraction and directly modeling functional data, including a thorough benchmark of those algorithms. This toolbox also allows for full or partial replication of the conducted benchmark comparison.

2. Related Work

In the remainder of the paper, we focus on comparing algorithms from the functional data analysis and the machine learning domain. Functional data analysis traditionally values interpretable results and valid statistical inference over prediction quality. Therefore functional data algorithms are often not compared with respect to their predictive performance in literature. We aim to close this gap. On the other hand, machine learning algorithms often do not yield interpretable results. While we consider both aspects to be important, we want to focus on predictive performance in this paper.

2.1. Feature extraction and classical machine learning methods

In this work, we differentiate between machine learning algorithms that can directly be applied to functional data, and algorithms intended for scalar features, which we call *classical* machine learning methods.
A popular approach when dealing with functional data is to reduce the problem to a non-functional task by extracting relevant non-functional features [1]. Applying classical machine learning methods after extracting meaningful features can then lead to competitive results [cf. 16, e.g.] or at least provide baselines, which are in general not covered by functional data frameworks. In our framework, such functionality is easily available by combining feature extraction, e.g., based on extracting heuristic properties [cf. **tsfeatures**; 14] or wavelet coefficients [17, 15] and analyzing these derived scalar features with classical machine learning tools provided by **mlr**.
Based on some existing functionality of the listed packages, we adapt different feature extraction methods. Along with different algorithms already proposed in literature, we propose two new custom methods, *DTWKernel* and *MultiRes-Features*:

> **tsfeatures** [14] extracts scalar features, such as auto-correlation functions, entropy and other heuristics from a time series.
>
> **fourier** transforms data from the time domain into the frequency domain using the *fast fourier transform* [18]. Extracted features are either phase or amplitude coefficients.
>
> **bsignal** B-Spline representations from package *FDboost* [5] are used as feature extractors. Given the knots vector and effective degree of freedom, we extract the design matrix for the functional data using *mboost*.
>
> **wavelets** [15] applies a discrete wavelet transform to time series or functional data, e.g., with Haar or Daubechies wavelets. The extracted features are wavelet coefficients at several resolution levels.
>
> **PCA** projects the data on their principal component vectors. Only a subset of the principal component scores representing a given proportion of signal variance is retained.

DTWKernel computes the dynamic time warping distances of functional or time series data to (a set of) reference data. We implement *dynamic time warping* (DTW) based feature extraction. This method computes the dynamic time warping distance of each observed function to a (user-specified) set of reference curves. The distances of each observation to the reference curves is then extracted as a vector-valued feature. The reference curves can either be supplied by the user, e.g., they could be several typical functions for the respective classes, or they can be obtained from the training data. In order to compute *dynamic time warping distances*, we use a fast dynamic time warping [19] implementation from package **rucrdtw** [20].

MultiResFeatures extracts features, such as the mean at different levels of resolution (zoom-in steps). Inspired by the image pyramid and wavelet methods, we implement a feature extraction method, *multi-resolution feature extraction* where we extract features like mean and variance computed over specified windows of varying widths. Starting from the full sequence, the sequence is repeatedly divided into smaller pieces, where at each resolution level, a scalar value is extracted. All extracted features are concatenated to form the final feature vector.

2.2. Methods for functional data

Without feature extraction, direct functional data modeling (both classification and regression) methods incorporated in our package span two families: The first family of semi-parametric approaches includes FGAM [8], FDboost [5], and the functional generalized linear model [FGLM; 21], which are all structured additive models. Those methods use (tensor product) spline basis functions or functional principal components (FPCs) [22] to represent effects fitted in a generalized additive model. While FGAM and FGLM use the iterated weighted least square (IWLS) method to generate maximum likelihood estimates, FDboost uses component-wise gradient boosting [23] to optimize the parameters. Additionally, the estimated parameters can be penalized. A general formula for this family of methods is $\zeta(Y|X = x) = h(x) = \sum_{j=1}^{J} h_j(x)$, where ζ represents a functional of the conditional response distribution (e.g., an expectation or a quantile), x is a vector of (functional) covariates and $h_j(x)$ are partial additive effects of subsets of x in basis function representation, cf. Greven and Scheipl [24] for a general introduction.

The second family of methods are non-parametric methods as introduced in Ferraty and Vieu [25], e.g., based on (semi-)metrics which quantify local or global differences or distances across curves. For example, the distance between two instances could be defined by the L_2 distance of two curves $d(x_i(t), x_j(t)) = \sqrt{\int (x_i(t) - x_j(t))^2 dt}$. Kernel functions are used to average over the training instances and weigh their respective contributions based on the value of their distance semi-metric to the predicted instance. Functional k-nearest neighbors algorithms can also be defined based on such semi-metrics. Implementations can be found in packages **fda.usc** [7] and **classiFunc** [26].

2.3. Toolboxes for functional data analysis

The package **fda** [4] contains several object types for functional data and allows for smoothing and regression for functional data. Analogously, the R-package **fda.usc** [7] contains several classification algorithms that can be used with functional data. In Python, **scikit-fda** [27] offers both representation of and (pre-)processing methods for functional data, but only a very small set of machine learning methods for classification or regression problems is implemented at the time of writing.
As a byproduct of the Time-Series Classification Bake-off [10], a wide variety of algorithms were implemented and made available. But this implementation emphasizes the benchmark over providing a data analysis toolbox for users, and is therefore not easily usable for inexperienced users.

2.4. Benchmarks

The recently published benchmark analysis **Time-Series Classification Bake-off** by Bagnall et al. [10] provides an overview of the performance of 18 state-of-the-art algorithms for time series classification. They re-implement (in Java) and compare 18 algorithms designed especially for time series classification on 85 benchmark time series data sets from Bagnall et al. [28]. In their analysis, they also include results from several standard machine learning algorithms. They note that the *rotation forest* [12] and *random forest* [29] are competitive with their time series classification baseline [1-nearest neighbor with dynamic time warping distance; 30]. Their results show that ensemble methods such as *collection of transformation ensembles* [COTE; 31] perform best, but for the price of considerable runtime.
Deep learning methods applied to time series classification tasks have also shown competitive prediction power. For example, [11] provide a comprehensive review of state-of-the-art methods. The authors compared both generative models and discriminative models, including *fully connected neural networks, convolutional neural networks, auto-encoders* and *echo state networks*, whereas only discriminative end-to-end approaches were incorporated in the benchmark study.
The benchmark study conducted in this work does not aim to replicate or compete with earlier studies like [10], but instead tries to extend their results.

3. Functional Data

In contrast to non-functional data analysis, where the measurement of a single observation is a vector of scalar components whose entries represent values of the separate multidimensional features, functional data analysis treats and analyses the features themselves as functions over their domain. By learning to represent the underlying function, the carried out analysis is not just restricted to the measured discrete values but it is possible to sample from (and analyze) the entire domain space.
In this work, we focus on pairs of features and corresponding labels (x, y) for supervised learning. In contrast to non-functional data analysis, where the

measurement of a single observation is a vector of scalar components, functional features are function-valued over their domain. The features $x = (x_1, ..., x_p)$ can thus also be a function, i.e., $x_j = g_j(t)$, $g : T \to \mathbb{R}$. In practice, functional data comes in the form of observed values $g_j(t), t \in \{1, ..., L\}$, where each t corresponds to a discrete point on the continuum. Those observed values stem from an underlying function f evaluated over a set of points. A frequent type of functional data is time series data, i.e., measurements of a process measured at discrete time-points.
For example, in some electrical engineering applications, signals are obtained over time at a certain sampling rate, but other domains are possible as well. Spectroscopic data, for example, are functional data recorded over certain parts of the electromagnetic spectrum. One such example is depicted in Figure 1. It shows spectroscopy data of fossil fuels [32] where the measured signal represents reflected energies in the ultraviolet-visible (UV-VIS) and the near infrared spectrum (NIR). In the plot, different colors correspond to different instances. This is a typical example of a scalar-on-function regression problem, where the inputs are a collection of spectroscopic curves for a fuel, and the prediction target is the heating value of the fossil fuel.
In Figure 2, we display two functional classification scenarios. The goal in those scenarios is to distinguish the class type of the curve, which can also be understood as a function-on-scalar problem. Figure 2a shows the vertical position of an actor's hand while either drawing a toy-gun and aiming at a target, or just imitating the motion with the blank hand. This position is measured over time. The two different types of classes of the curves can be distinguished by the color scheme.
Figure 2b shows a data set built for distinguishing images of beetles from images of flies based on their outlines. While following the outline, the distance to the center of the object is measured which is then used for classification purposes. The latter data sets are available from [28].
The interested reader is referred to Ramsay [21] and Kokoszka and Reimherr [35] for more in-depth introductions to this topic.

4. Functional Data Analysis with mlrFDA

Along with the benchmark, we implement the software **mlrFDA**, which extends the popular machine learning framework **mlr** for the analysis of functional data. As the implemented functionality is an extension of the **mlr** package, all of the functionality available in **mlr** transfers to the newly added methods for functional data analysis. We include a brief overview of the implemented functionality in Appendix A.1. A more in-detail overview and tutorial on **mlr** can be found in the mlr tutorial [36].
mlr provides a unified framework for machine learning methods in R, currently supporting *tasks* from 4 main problem types: (multilabel-)classification, regression, cluster analysis, and survival analysis. For each problem type, many algorithms (called *learners*) are integrated. This yields an extensive set of modeling options with a unified, simple interface. Moreover, advanced techniques such

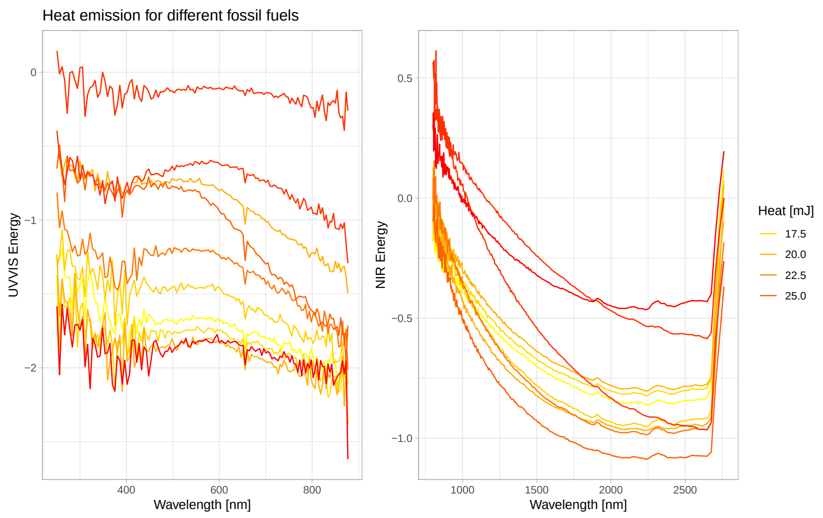


Figure 1: Scalar-on-function regression: Spectral data for fossil fuels [32]

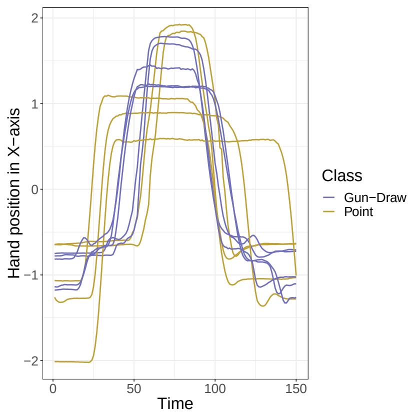
as hyperparameter tuning, preprocessing and feature selection are also part of the package. An additional focus lies on extensibility, allowing the user to integrate their own algorithms, performance measures and preprocessing methods. As **mlrMBO** seamlessly integrates into the new software, many different tuning procedures can be readily adapted by the user. Tuning of hyperparameters is usually not integrated in software packages for functional data analysis and thus would require the user to write additional, non-trivial code that handles (nested) resampling, evaluation and optimization methods.

mlrFDA contains several functional data algorithms from several R packages, e.g., **fda.usc**, **refund** or **FDboost**. The algorithms' functionality, however, remains unchanged, only their user interface is standardized for use with **mlr**. For detailed insights into the respective algorithms, full documentation is available in the respective packages.

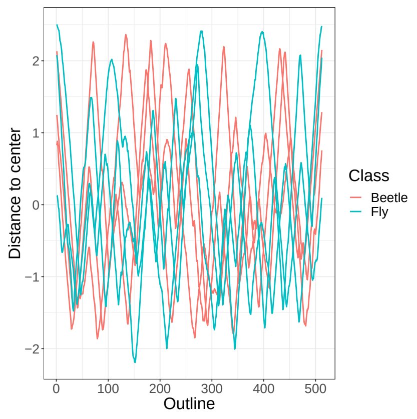
Since our toolbox is built on **mlr**'s extensible class system, our framework is easily extensible to other methods that have not yet been integrated, and the user can include his or her own methods which do not necessarily need to be available as a packaged implementation. Additionally, **mlrFDA** inherits **mlr**'s functionality for performance evaluation and benchmarking, along with extensive and advanced (hyperparameter) tuning. This makes our platform very attractive for evaluating which algorithm fits best to a problem at hand, and even allows for large benchmark studies.

5. Benchmark Experiment

In order to enable a comparison of the different approaches, an extensive benchmark study is conducted. This paper does not aim to replicate or reproduce



(a) Gunpoint



(b) BeetleFly

Figure 2: Excerpts from two time series classification data sets. (a): Gunpoint data [33], (b): BeetleFly Data [34].

results obtained by Bagnall et al. [10] or Fawaz et al. [11]. Instead we focus on providing a benchmark complementary to previous benchmarks. This is done because *i*) the experiments require large amounts of computational resources, and *ii*) the added value of an exact replication of the experiments (with open source code) is comparatively small. Nonetheless, we aim for results that can be compared, and thus extend the results obtained by Bagnall et al. [10] by staying close to their setup. The experiments were carried out on a high performance computing cluster, supported by the Leibniz Rechenzentrum Munich. Individual runs were allowed up to 2.2 GB of RAM and 4 hours run-time for each evaluation. We want to stress that this benchmark compares *implementations*, which does not always necessarily correspond to the performance of the corresponding theoretical *algorithm*. Additionally, methods for functional data analysis are traditionally more focused on valid statistical inference and interpretable results, which does not necessarily coincide with high predictive performance.

5.1. Benchmark Setup

A benchmark experiment is defined by four important characteristics: The *data sets* algorithms are tested on, the *algorithms* to be evaluated, the *measures* used for evaluating predictive performance, and a *resampling strategy* used for generating train and test splits of the data. A comprehensive overview of the conducted benchmark setup can be obtained from Table 1.

These characteristics are briefly described subsequently before providing and discussing the results. We use a subset of 51 data sets from the popular UCR archive [28] in order to enable a comparison of results in [10] with the additional methods described in this paper. The data sets stem from various application types such as ECG measurements, sensor data, or image outlines, therefore

Data sets	51 Data sets, see table B.7
Algorithms Machine Learning:	Function (Package) - `glmnet` (**glmnet**) - `rpart` (**rpart**) - `ksvm`* (**kernlab**) - `ranger`* (**ranger**) - `xgboost`* (**xgboost**)
Functional Data	- `classif.knn`(**fda.usc**) - `classif.glm` (**fda.usc**) - `classif.np` (**fda.usc**) - `classif.kernel`(**fda.usc**) - `FDboost` (**FDboost**) - `fgam` (**refund**) - `knn with dtw` (**classiFunc**)
Feature Extraction + ML	- feature extraction: see table A.6 - in combination with ML algorithms marked with a *.
Measures	mean misclassification error, training time
Resampling	20-fold stratified sub-sampling; class balances and train/test set size as in [10].
Tuning	100 iterations of Bayesian optimization (3-fold inner CV). Corresponding hyperparameter-ranges can be obtained from tables 3 and 5.

Table 1: Benchmark experiment setup

having varying training set sizes or measurement lengths. For more detailed information about the data sets, see Bagnall et al. [28].

We selected data using the following criteria: In order to reduce the computational resources we did *i*) not run data sets that have multiple versions, *ii*) exclude data sets with less then 3 examples in each class *iii*) remove data sets with more than 10000 instances or time series longer than 750 measurements. As some of the classifiers only work with multi-class targets via *1-vs-all* classification, we *iv*) additionally excluded data sets with more then 40 classes. In essence, we benchmark small and medium sized data sets with a moderate amount of different classes.

We add 7 new algorithms and 6 feature extraction methods which can be combined with arbitrary machine learning methods for scalar features (c.f. Table 1). Additionally we test 5 classical machine learning methods, in order to obtain a broader perspective on expected performance if the functional nature of the data is ignored. As we benchmark default settings as well as tuned algorithms, in total 80 different algorithms are evaluated across all data sets. When combining feature extraction and machine learning methods, we fuse the learning algorithm and the preprocessing, thus treating them as a pipeline where data is internally transformed before applying the learner. This allows us to jointly tune the hyperparameters of learning algorithm and preprocessing method. The respective defaults and parameter ranges can be obtained from Table 3 (feature extractors) and Table 5 (learning algorithms). More detailed description of the

hyperparameters can be obtained from the respective packages documentation. In order to generate train/test splits, and thus obtain an unbiased estimate of the algorithm's performance, we use stratified sub-sampling. We use 20 different train/test splits for each data set in order to reduce variance and report the average. For tuned models, we use use nested cross-validation [37] to ensure unbiased estimates, where the outer loop is again subsampling with 20 splits, and the inner resampling for tuning is a 3-fold (stratified) cross-validation. All compared 80 algorithms are presented exactly the same index sets for the 20 train-test outer subsampling splits.
Mean misclassification error (MMCE) is chosen as a measure of predictive performance in order to stay consistent with Bagnall et al. [10]. Other measures, such as area under the curve (AUC) require predicted probabilities and do not trivially extend to multi-class settings.
While Bagnall et al. [10] tune all algorithms across a carefully handcrafted grid, we use *Bayesian optimization* [38]. In order to stay comparable, we analogously fix the amount of tuning iterations to 100.
We use **mlrMBO** [39] in order to perform Bayesian optimization of the hyperparameters of the respective algorithm. Additionally, in order to scale the method to a larger amount of data sets and machines, the R-package **batchtools** (Bischl et al. [40], Lang et al. [41]) is used. This enables running benchmark experiments on high-performance clusters. For the benchmark experiment, a job is defined as re-sampling of a single algorithm (or tuning thereof) on a single version of a data set. This allows for parallelization to an arbitrary number of CPU's, while at the same time guaranteeing reproducibility. The code for the benchmark is available from `https://github.com/compstat-lmu/2019_fda_benchmark` for reproducibility.

5.2. Results

This Section tries to answer the questions posed in section 1. We evaluate *i*) various machine learning algorithms in combination with feature extraction, *ii*) classical time series classification approaches, *iii*) the effect of tuning hyperparameters for several methods, and *iv*) try to give recommendations with respect to which algorithm(s) to choose for new classification problems.
Algorithms evaluated in this benchmark have been divided into three groups: Algorithms specifically tailored to functional data, *classical* machine learning algorithms without feature extraction and *classical* machine learning algorithms in combination with feature extraction.

5.2.1. Algorithms for functional data

Performances of algorithms specifically tailored to functional data analysis can be obtained from Figure 3. The *k-nearest neighbors* algorithm from package **classiFunc** [26] in combination with dynamic time warping [19] distance seems to perform best across data sets. It is also considered a *strong baseline* in Bagnall et al. [10].

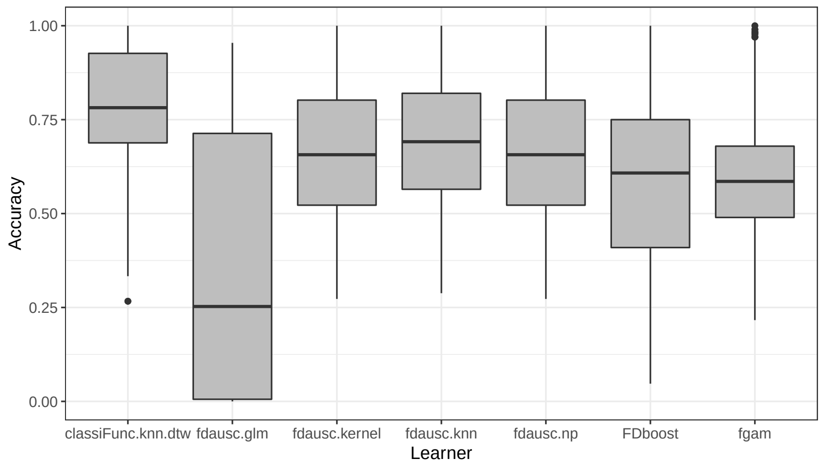


Figure 3: Performances for functional data analysis algorithms in default settings (untuned) across all 51 data sets.

5.2.2. *Machine Learning algorithms with feature extraction*

Performances of different machine learning algorithms in combination with feature extraction with and without tuning can be obtained from Figure 4.

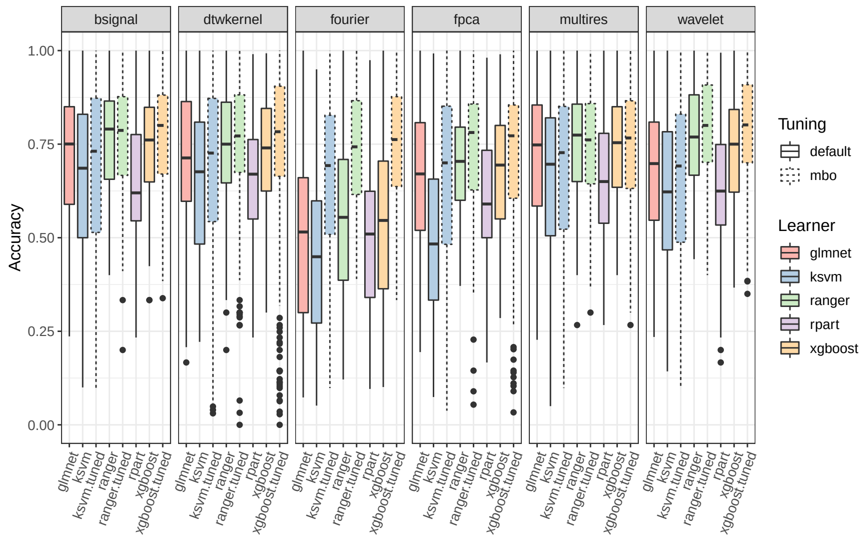


Figure 4: Results for feature extraction-based machine learning algorithms with default and tuned (MBO) hyperparameters across 51 data sets. Hyperparameters are tuned jointly for learner and feature extraction method.

We conclude that feature extraction using splines (bsignal) and wavelets as well as extracting dynamic time warping distances works well when combined with conventional machine learning algorithms, even at their default hyper-parameters. Among the learners, random forests, especially in combination with *bsignal* show quite advantageous performance. In addition, we find an obvious improvement from hyper-parameter tuning for the Fourier feature extraction. In terms of learners, random forest and gradient boosted tree learners (`xgboost`) perform better than support vector machines.

5.2.3. Machine Learning algorithms without feature extraction

Additionally, we aim to provide some insight with regards to the performance of machine learning algorithms that ignore the functional nature of our data. Figure 5 provides an overview over the performance of different machine learning algorithms that are often used together with traditional tabular data. Performances in this figure are obtained from algorithms directly applied to the functional data without any additional feature extraction. The widely used gradient boosting (`xgboost`) and random forest (`ranger`) implementations seem to work reasonably well for functional data even without additional feature extraction.

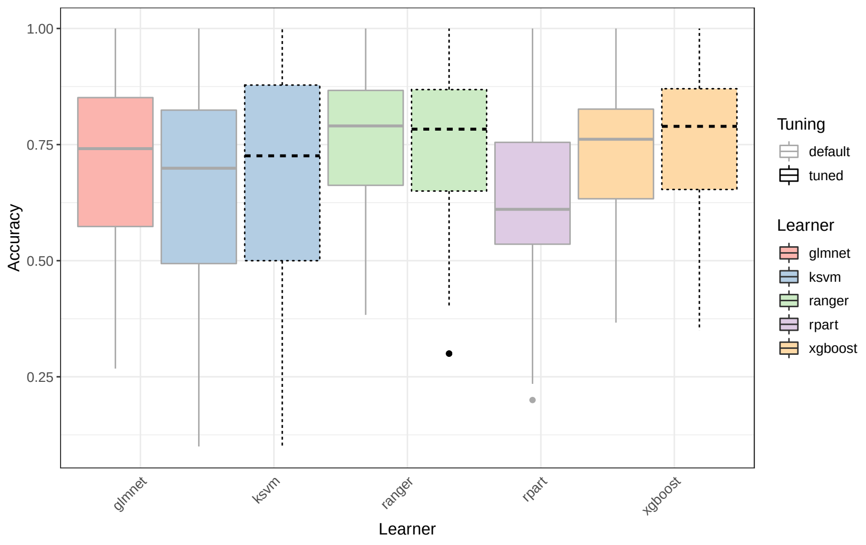


Figure 5: Performance of non-functional machine learning algorithms across 51 data sets applied directly to functional data with and without tuning.

5.2.4. The effect of tuning hyperparameters

From our experiments, we conclude, that tuning hyperparameters of machine learning algorithms in general has a non-negligible effect on the performance. Using Bayesian optimization in order to tune algorithm hyperparameters on average yielded an absolute increase in accuracy of 5.4% across data sets and learners.

Figure 6 displays the aggregated time over all data sets, taking into account the time required for hyperparameter tuning. All experiments have been run on equivalent hardware on high-performance computing infrastructure. Due to fluctuations in server load, this does not allow for an exact comparison with respect to computation time, but we hope to achieve comparable results as we repeatedly evaluate on sub-samples. Note that we restrict the *tuning* to 3 algorithms where tuning traditionally leads to higher performances.[2]

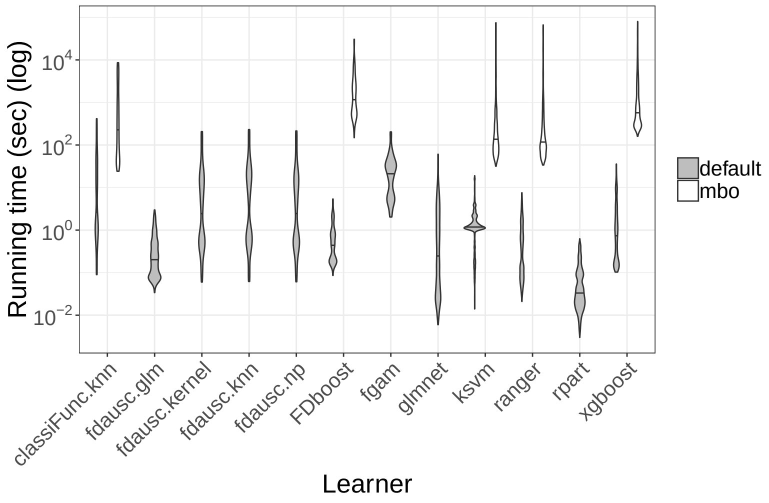


Figure 6: Comparison of running time for the different learner classes with default and tuned hyperparameters across 51 data sets. A log transformation on the running time in seconds is applied, and the mean running time is visualized for each stratification as a horizontal line within the violin plot.

5.2.5. Top 10 Algorithms and recommendations

Table 2 showcases the top 10 algorithms from the benchmark in terms of average rank in predictive accuracy across data sets. With this list, we aim to provide some initial understanding of the performance of different algorithms and feature extraction methods. Note that this list by no means reflects performance on future data sets, but might serve as an indicator, of which algorithms one might want to try first given computational constraints.

We observe that wavelet extraction in combination with either ranger or xgboost seems to be very strong. They obtain an average rank of 12.90 and 14.45 (out of 80) respectively. Dynamic time warping distances for k-nearest neighbors indeed seems to be a strong baseline, even without tuning. Another strong feature extraction method seems to be the extraction of B-spline features. Using the 10 algorithms above allows us to obtain an accuracy within 5% of the maximum on 49 of the 51 data sets.

[2]Additionally, we find significantly improved performance for tuned FDboost in Figure 3

Table 2: Top 10 algorithms by average rank across all data sets. Percent Accuracy describes the fraction of the maximal accuracy reached for each task.

Algorithm Setting	Accuracy %	Average Rank
ranger_wavelet_tuned	0.92	12.90
xgboost_wavelet_tuned	0.92	14.45
ranger_bsignal_tuned	0.90	15.02
knn_dtw_tuned	0.92	15.22
ranger_none_default	0.90	15.59
ranger_bsignal_default	0.89	15.71
ranger_wavelet_default	0.90	16.33
knn_dtw_default	0.92	16.43
xgboost_bsignal_tuned	0.90	17.57
ranger_none_tuned	0.89	18.49

If the only criterion for model selection is predictive performance, (tuned) machine learning models in combination with feature extraction is a competitive baseline. This class of methods achieves within 95% of the optimal performance on 47 out of 51 data sets, while they include the best performing classifier in 35 cases.

5.2.6. Comparison to classical time series classification

Even though the main purpose of this paper is not a direct comparison with the results from [10], we can use our results to show that applying functional data approaches and classical machine learning approaches together with feature extraction can still improve classification accuracy compared to current state-of-the-art time series classification methods.

In the experiments we conducted, the methods described in this paper improved accuracy on 9 out of the 51 data sets which is displayed in Figure 7. The 9 data sets and the corresponding best learner are displayed in Table 4. For each data set, only the best reached accuracy for both sets of algorithms is displayed.

Additionally, we evaluate how our learners rank in comparison to the individual bake-off algorithms from [10]. The algorithm which performs best on a data set obtains the rank 1. The mean rank of the individual learners over all 49 data sets (we take the intersection of the data sets from our benchmark and the ones from [10]). The average sorted ranks for the top 50% algorithms are displayed in Figure 8. We observe that the ensemble methods get the top ranks, which is no surprise, as for instance the COTE algorithm [42] internally combines several classifiers from 4 different time series domains.

However, compared to the classical time series algorithms from [10] with the ensemble methods removed, our functional data algorithms obtain an overall good rank in accuracy performance, interleaved with the algorithms from [10]. Note that the benchmarks are not exactly comparable due to minor differences in the benchmark setup, and we instead only include their reported results.

id	type	values	def.	trafo
bsignal				
bsignal.knots	int	{3,...,500}	10	-
bsignal.df	int	{1,...,10}	3	-
multires				
res.level	int	{2,...,5}	-	-
shift	num	[0.01,1]	-	-
pca				
rank.	int	{1,...,30}	-	-
wavelets				
filter	chr	d4,d8,d20,la8,la20,bl14,bl20,c6,c24	-	-
boundary	chr	periodic,reflection	-	-
fourier				
trafo.coeff	chr	phase,amplitude	-	-
dtwkernel				
ref.method	chr	random,all	random	-
n.refs	num	[0,1]	-	-
dtwwindow	num	[0,1]	-	-

Table 3: Parameter spaces and default settings for feature extraction methods.

6. Summary and Outlook

In this work, we provide a benchmark along with a software implementation that integrates the functionality of a diverse set of R-packages into a single user interface and API. Both contributions come with a multiplicity of benefits:

- The user is not required to learn and deal with the vast complexity of the different interfaces the underlying packages expose.
- All of the existing functionality (e.g., preprocessing, resampling, performance measures, tuning, parallelization) of the **mlr** ecosystem can now be used in conjunction with already existing algorithms for functional data.
- We expose functionality that allows us to work with functional data using *traditional* machine learning methods via feature extraction methods.
- Integration of additional preprocessing methods or models is (fairly) trivial and automatically benefits from the full **mlr** ecosystem.

In order to obtain a broader overview of the performance of the integrated methods, we perform a large benchmark study. This allows users to get an initial overview of potential performances of the different algorithms. Specifically,

- We open up new perspectives for time series classification tasks by incorporating methods from functional data analysis, as well as feature transformations combined with conventional machine learning models.

Name	Algorithm_Setting	Accuracy
Beef	xgboost_wavelet_tuned	0.83
ChlorineConcentration	ksvm_none_tuned	0.91
DistalPhalanxOutlineAgeGroup	ranger_none_default	0.83
DistalPhalanxOutlineCorrect	ranger_dtwkernel_default	0.83
DistalPhalanxTW	ranger_bsignal_default	0.76
Earthquakes	FDboost_none_default	0.80
Ham	xgboost_wavelet_tuned	0.84
InsectWingbeatSound	ranger_wavelet_default	0.65
SonyAIBORobotSurface1	ksvm_wavelet_default	0.94

Table 4: Data sets together with corresponding **mlrFDA** learner and accuracy for which our learners were able to improve accuracy in the conducted experiments.

- Based on the large scale benchmark, we conclude that many learners have competitive performance (Figure 7) and additionally offer the interpretability of many functional data analysis methods. Our toolbox serves as a strong complement and alternative to other time series classification software.

- The presented benchmark study uses state-of-the-art Bayesian optimization for hyperparameter optimization, which results in significant improvements over models that are not tuned. This kind of hyperparameter tuning is easy to do with **mlrFDA**. Tuning, albeit heavily influencing performance is often not investigated. Our benchmark closes this gap in existing literature.

- We find that extracting vector valued features and feeding them to a conventional machine learning model can often form competitive learners.

- The pareto-optimal set in terms of performance on each data set contains 23 different algorithm−feature-extraction combinations. Our toolbox *i*) offers the same API for all methods and *ii*) allows to automatically search over this space, and thus allows users to obtain optimal models without knowing all underlying methods.

Concerning the questions we proposed at the beginning of the paper, we draw the following conclusions:

- Tuning only a subset of the presented learners and feature extractions, i.e., the methods listed in Table 2, is sufficient to achieve good performances on almost all data sets in our benchmark.

- A simple random forest without any preprocessing can also be a reasonable baseline for time series data. It achieves an average rank of 15.59 (top 4) in our benchmark.

parameter	type	values	default	trafo
ksvm				
C	num	[-15,15]	-	2^x
sigma	num	[-15,10]	-	2^x
ranger				
mtry.power	num	[0,1]	-	p^x
min.node.size	num	[0,0.99]	-	2^$(log2(n) * x)$
sample.fraction	num	[0.1,1]	-	-
xgboost				
nrounds	int	{1,...,5000}	100	-
eta	num	[-10,0]	-	2^x
subsample	num	[0.1,1]	-	-
booster	chr	gbtree,gblinear	-	-
max_depth	int	{1,...,15}	-	-
min_child_weight	num	[0,7]	-	2^x
colsample_bytree	num	[0,1]	-	-
colsample_bylevel	num	[0,1]	-	-
lambda	num	[-10,10]	-	2^x
alpha	num	[-10,10]	-	2^x
FDboost				
mstop	int	{1,...,5000}	100	-
nu	num	[0,1]	0.01	-
df	num	[1,5]	4	-
knots	int	{5,...,100}	10	-
degree	int	{1,...,4}	3	-

Table 5: Parameter spaces and defaults used for tuning machine learning and functional data algorithms. In case no default is provided, package defaults are used. Additional information can be found in the respective packages documentation.

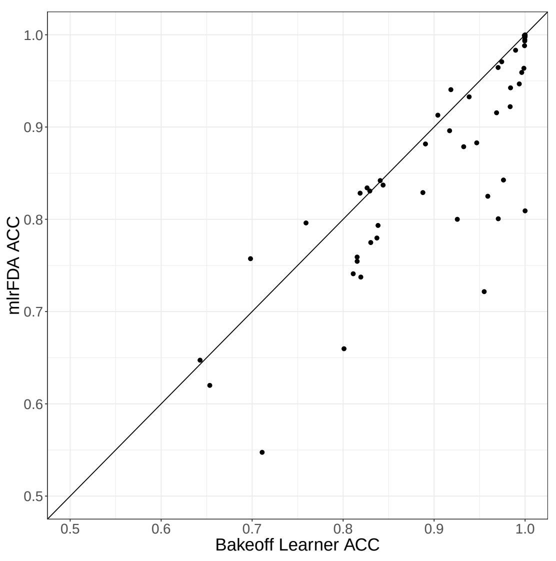


Figure 7: Comparing accuracy between our **mlrFDA** learners and the classical time series classification algorithms in [10]. For each data set, only the best accuracy for each of the two benchmarks is shown. We observe that for 9 of the evaluated data sets the classification performance can directly be improved solely by applying our **mlrFDA** learners, while we perform on par with the classical time series classification algorithms (when rounding to 3 decimal digits) on two data sets.

- Most algorithms for functional data (e.g., `FDboost`) do not perform well in our benchmark study. As those algorithms are fully interpretable and offer statistically valid coefficients, they can still be useful in some applications, and should thus not be ruled out.
- Feature extraction techniques, such as b-spline representations (*bsignal*) and wavelet extraction work well in conjunction with machine learning techniques for vector valued features such as `xgboost` and `random forest`.
- Tuning leads to an average reduction in absolute MMCE of 3.59% (ranger), 5.69% (xgboost), 7.78% (ksvm) (across feature extraction techniques) and 11% (FDboost). This holds for all feature extraction techniques, where improvements range from 1.12% *multires* to 20.3% *fourier*.

In future work we will continue to expand the available toolbox along with a benchmark of new methods, and provide the R community a wider range of methods that can be used for the analysis of functional data. This includes not only integrating many already available packages, and as a result to enable preprocessing operations such as smoothing (e.g., **fda** [4]) and alignment (e.g., **fdasrvf** [43] or **tidyfun** [44]), but also to explore and integrate advanced imputation methods for functional data. Further work will also extend the current implementation to support data that is measured on unequal or irregular grids. Additionally, we aim to implement some of the current state-of-the art machine learning models from the time series classification bake-off [10], such as the *Collective of Transformation-Based Ensembles* (COTE) [31]. This enables researchers to use and compare with current state-of-the-art methods.

Acknowledgements

This work has been funded by the German Federal Ministry of Education and Research (BMBF) under Grant No. 01IS18036A. The authors of this work take full responsibilities for its content.

References

[1] S. Ullah, C. F. Finch, Applications of functional data analysis: A systematic review, BMC Medical Research Methodology 13 (2013) 43. URL: https://doi.org/10.1186/1471-2288-13-43. doi:10.1186/1471-2288-13-43.

[2] J.-L. Wang, J.-M. Chiou, H.-G. Müller, Functional data analysis, Annual Review of Statistics and Its Application 3 (2016) 257–295. URL: https://doi.org/10.1146/annurev-statistics-041715-033624. doi:10.1146/annurev-statistics-041715-033624.

[3] R Core Team, R: A Language and Environment for Statistical Computing, R Foundation for Statistical Computing, Vienna, Austria, 2014. URL: http://www.R-project.org/.

[4] J. O. Ramsay, H. Wickham, S. Graves, G. Hooker, fda: Functional Data Analysis, 2018. URL: https://CRAN.R-project.org/package=fda, r package version 2.4.8.

[5] S. Brockhaus, D. Ruegamer, FDboost: Boosting Functional Regression Models, 2018.

[6] F. Scheipl, Cran task view - functional data analysis, 2018. URL: https://cran.r-project.org/web/views/FunctionalData.html.

[7] M. Febrero-Bande, M. Oviedo de la Fuente, Statistical computing in functional data analysis: The r package fda.usc, Journal of Statistical Software 51 (2012) 1–28. URL: http://www.jstatsoft.org/v51/i04/.

[8] J. Goldsmith, F. Scheipl, L. Huang, J. Wrobel, J. Gellar, J. Harezlak, M. W. McLean, B. Swihart, L. Xiao, C. Crainiceanu, P. T. Reiss, refund: Regression with Functional Data, 2018. URL: https://CRAN.R-project.org/package=refund, r package version 0.1-17.

[9] C. Cortes, V. Vapnik, Support-vector networks, Mach. Learn. 20 (1995) 273–297. URL: https://doi.org/10.1023/A:1022627411411. doi:10.1023/A:1022627411411.

[10] A. Bagnall, J. Lines, A. Bostrom, J. Large, E. Keogh, The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances, Data Mining and Knowledge Discovery 31 (2017) 606–660.

[11] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, P.-A. Muller, Deep learning for time series classification: a review, Data Mining and Knowledge Discovery (2019) 1–47.

[12] J. J. Rodriguez, L. I. Kuncheva, C. J. Alonso, Rotation forest: A new classifier ensemble method, IEEE Transactions on Pattern Analysis and Machine Intelligence 28 (2006) 1619–1630. doi:10.1109/TPAMI.2006.211.

[13] C. Stachl, M. Bühner, Show me how you drive and i'll tell you who you are. recognizing gender using automotive driving parameters, Procedia Manufacturing 3 (2015) 5587 – 5594. URL: http://www.sciencedirect.com/science/article/pii/S2351978915007441. doi:https://doi.org/10.1016/j.promfg.2015.07.743, 6th International Conference on Applied Human Factors and Ergonomics (AHFE 2015) and the Affiliated Conferences, AHFE 2015.

[14] R. Hyndman, E. Wang, Y. Kang, T. Talagala, Y. Yang, tsfeatures: Time Series Feature Extraction, 2018. URL: https://github.com/robjhyndman/tsfeatures/, r package version 0.1.

[15] E. Aldrich, wavelets: A package of functions for computing wavelet filters, wavelet transforms and multiresolution analyses, 2013. URL: https://CRAN.R-project.org/package=wavelets, r package version 0.3-0.

[16] J. Goldsmith, F. Scheipl, Estimator selection and combination in scalar-on-function regression, Computational Statistics & Data Analysis 70 (2014) 362–372.

[17] S. Mallat, A theory for multiresolution signal decomposition: The wavelet representation, IEEE Trans. Pattern Anal. Mach. Intell. 11 (1989) 674–693.

[18] E. O. Brigham, R. E. Morrow, The fast fourier transform, IEEE Spectrum 4 (1967) 63–70. doi:10.1109/MSPEC.1967.5217220.

[19] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, E. Keogh, Searching and mining trillions of time series subsequences under dynamic time warping, in: Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2012, pp. 262–270. URL: http://doi.org/10.1145/2339530.2339576. doi:10.1145/2339530.2339576.

[20] P. Boersch-Supan, rucrdtw: Fast time series subsequence search in r, The Journal of Open Source Software 1 (2016) 1–2. URL: http://doi.org/10.21105/joss.00100. doi:10.21105/joss.00100.

[21] J. Ramsay, Functional data analysis, Wiley Online Library, 2006.

[22] A. Srivastava, E. P. Klassen, Functional and Shape Data Analysis, Springer, 2016.

[23] T. Hothorn, P. Bühlmann, T. Kneib, M. Schmid, B. Hofner, Model-based boosting 2.0, Journal of Machine Learning Research 11 (2010) 2109–2113.

[24] S. Greven, F. Scheipl, A general framework for functional regression modelling, Statistical Modelling 17 (2017) 1–35.

[25] F. Ferraty, P. Vieu, Nonparametric functional data analysis: theory and practice, Springer Science & Business Media, 2006.

[26] T. Maierhofer, F. Pfisterer, classiFunc: Classification of Functional Data, 2018. URL: `https://CRAN.R-project.org/package=classiFunc`, r package version 0.1.1.

[27] Grupo de Aprendizaje Automatico - Universidad Autonoma de Madrid, scikit-fda: Functional Data Analysis in Python, 2019. URL: `https://fda.readthedocs.io`.

[28] A. Bagnall, J. Lines, W. Wickers, E. Keogh, The UEA & UCR time series classification repository, 2017. `www.timeseriesclassification.com`.

[29] L. Breiman, Random forests, Mach. Learn. 45 (2001) 5–32. URL: `https://doi.org/10.1023/A:1010933404324`. doi:`10.1023/A:1010933404324`.

[30] P. Tormene, T. Giorgino, S. Quaglini, M. Stefanelli, Matching incomplete time series with dynamic time warping: An algorithm and an application to post-stroke rehabilitation, Artificial Intelligence in Medicine 45 (2008) 11–34. doi:`10.1016/j.artmed.2008.11.007`.

[31] A. Bagnall, J. Lines, J. Hills, A. Bostrom, Time-series classification with cote: The collective of transformation-based ensembles, IEEE Transactions on Knowledge and Data Engineering 27 (2015) 2522–2535. doi:`10.1109/TKDE.2015.2416723`.

[32] K. Fuchs, F. Scheipl, S. Greven, Penalized scalar-on-functions regression with interaction term, Computational Statistics & Data Analysis 81 (2015) 38–51. doi:`10.1016/j.csda.2014.07.001`.

[33] C. A. Ratanamahatana, E. J. Keogh, Three myths about dynamic time warping data mining., in: H. Kargupta, J. Srivastava, C. Kamath, A. Goodman (Eds.), SDM, SIAM, 2005, pp. 506–510.

[34] J. Hills, J. Lines, E. Baranauskas, J. Mapp, A. Bagnall, Classification of time series by shapelet transformation, Data Min. Knowl. Discov. 28 (2014) 851–881. URL: `http://dx.doi.org/10.1007/s10618-013-0322-1`. doi:`10.1007/s10618-013-0322-1`.

[35] P. Kokoszka, M. Reimherr, Introduction to functional data analysis, CRC Press, 2017.

[36] J. Schiffner, B. Bischl, M. Lang, J. Richter, Z. M. Jones, P. Probst, F. Pfisterer, M. Gallo, D. Kirchhoff, T. Kühn, et al., mlr tutorial, arXiv preprint arXiv:1609.06146 (2016).

[37] B. Bischl, O. Mersmann, H. Trautmann, C. Weihs, Resampling methods for meta-model validation with recommendations for evolutionary computation, Evolutionary Computation 20 (2012) 249–275. URL: `https://doi.org/10.1162/EVCO_a_00069`. doi:`10.1162/EVCO_a_00069`, pMID: 22339368.

[38] J. Snoek, H. Larochelle, R. P. Adams, Practical bayesian optimization of machine learning algorithms, in: F. Pereira, C. J. C. Burges, L. Bottou, K. Q. Weinberger (Eds.), Advances in Neural Information Processing Systems 25, Curran Associates, Inc., 2012, pp. 2951–2959. URL: http://papers.nips.cc/paper/4522-practical-bayesian-optimization-of-machine-learning-algorithms.pdf.

[39] B. Bischl, J. Richter, J. Bossek, D. Horn, J. Thomas, M. Lang, mlrMBO: A Modular Framework for Model-Based Optimization of Expensive Black-Box Functions, 2017. URL: http://arxiv.org/abs/1703.03373.

[40] B. Bischl, M. Lang, O. Mersmann, J. Rahnenführer, C. Weihs, BatchJobs and BatchExperiments: Abstraction mechanisms for using R in batch environments, Journal of Statistical Software 64 (2015) 1–25. URL: http://www.jstatsoft.org/v64/i11/.

[41] M. Lang, B. Bischl, D. Surmann, batchtools: Tools for r to work on batch systems, The Journal of Open Source Software 2 (2017). URL: https://doi.org/10.21105/joss.00135. doi:10.21105/joss.00135.

[42] A. Bagnall, J. Lines, J. Hills, A. Bostrom, Time-series classification with cote: the collective of transformation-based ensembles, IEEE Transactions on Knowledge and Data Engineering 27 (2015) 2522–2535.

[43] J. D. Tucker, fdasrvf: Elastic Functional Data Analysis, 2016. URL: https://CRAN.R-project.org/package=fdasrvf, r package version 1.6.0.

[44] F. Scheipl, J. Goldsmith, tidyfun, https://github.com/fabian-s/tidyfun, 2019.

[45] L. Jin, Q. Niu, Y. Jiang, H. Xian, Y. Qin, M. Xu, Driver sleepiness detection system based on eye movements variables, Advances in Mechanical Engineering 5 (2013) 648431. URL: https://doi.org/10.1155/2013/648431. doi:10.1155/2013/648431.

[46] M. Murugappan, M. Rizon, R. Nagarajan, S. Yaacob, Eeg feature extraction for classifying emotions using fcm and fkm, in: Proceedings of the 7th WSEAS International Conference on Applied Computer and Applied Computational Science, ACACOS'08, World Scientific and Engineering Academy and Society (WSEAS), Stevens Point, Wisconsin, USA, 2008, pp. 299–304. URL: http://dl.acm.org/citation.cfm?id=1415743.1415793.

[47] S. Soltani, On the use of the wavelet decomposition for time series prediction, Neurocomputing 48 (2002) 267 – 277. URL: http://www.sciencedirect.com/science/article/pii/S0925231201006488. doi:https://doi.org/10.1016/S0925-2312(01)00648-8.

6.2 Regularized target encoding outperforms traditional methods in supervised machine learning with high cardinality features

Contributed Article:
F. Pargent, F. Pfisterer, J. Thomas, and B. Bischl. Regularized target encoding outperforms traditional methods in supervised machine learning with high cardinality features. *Computational Statistics*, pages 1–22, 2022

Declaration of contributions The contribution originated from a master's thesis by FPa that investigated the effect of different categorical encoding techniques supervised by JT and BB. This resulted in a first draft that extended the analysis of the original thesis, which was mainly authored by FPa with help from FPf who substantially revised the manuscript. JT and BB provided helpful feedback, reviewed and edited the final manuscript.

Computational Statistics
https://doi.org/10.1007/s00180-022-01207-6

ORIGINAL PAPER

Regularized target encoding outperforms traditional methods in supervised machine learning with high cardinality features

Florian Pargent[1] · Florian Pfisterer[2] · Janek Thomas[2] · Bernd Bischl[2]

Received: 30 April 2021 / Accepted: 7 February 2022

Abstract
Since most machine learning (ML) algorithms are designed for numerical inputs, efficiently encoding categorical variables is a crucial aspect in data analysis. A common problem are high cardinality features, i.e. unordered categorical predictor variables with a high number of levels. We study techniques that yield numeric representations of categorical variables which can then be used in subsequent ML applications. We focus on the impact of these techniques on a subsequent algorithm's predictive performance, and—if possible—derive best practices on when to use which technique. We conducted a large-scale benchmark experiment, where we compared different encoding strategies together with five ML algorithms (lasso, random forest, gradient boosting, *k*-nearest neighbors, support vector machine) using datasets from regression, binary- and multiclass–classification settings. In our study, regularized versions of target encoding (i.e. using target predictions based on the feature levels in the training set as a new numerical feature) consistently provided the best results. Traditionally widely used encodings that make unreasonable assumptions to map levels to integers (e.g. integer encoding) or to reduce the number of levels (possibly based on target information, e.g. leaf encoding) before creating binary indicator variables (one-hot or dummy encoding) were not as effective in comparison.

Keywords Supervised machine learning · Benchmark · High-cardinality categorical features · Target encoding · Dummy encoding · Generalized linear mixed models

Florian Pargent
florian.pargent@psy.lmu.de

[1] Department of Psychology, Psychological Methods and Assessment, LMU Munich, Leopoldstraße 13, 80802 Munich, Germany

[2] Department of Statistics, Statistical Learning and Data Science, LMU Munich, Ludwigstraße 33, 80539 Munich, Germany

1 Introduction

While increasing sample size is usually considered the most important step to improve the predictive performance of a machine learning (ML) model, using effective feature engineering comes as a close second. One remaining challenge is how to handle high cardinality features—categorical predictor variables with a high number of different levels but without any natural ordering. While categorical variables with only a small number of possible levels can often be efficiently dealt with using standard techniques such as one-hot encoding, this approach becomes inefficient as the number of levels increases. Despite this inefficiency, simpler strategies are often favored in practice because other methods are either not known, implementations are missing or because of a lack of trust due to missing validation studies. Although domain knowledge can sometimes be used to reduce the number of theoretically relevant levels, finding strategies that work well on a large variety of problems is highly important for many applications as well as in automated ML (Feurer et al. 2015; Thomas et al 2018; Thornton et al. 2013). Optimally, strategies should be model-agnostic because benchmarking encoding methods together with ML algorithms from different classes is often necessary for applications. While a variety of strategies exist, there are very few benchmarks that can be used to decide which technique is expected to yield good predictive performance. Furthermore, there has recently been increasing attention on scientific benchmark studies that compare different methods to provide a clearer picture in light of a large number of methods available to practitioners (Bommert et al. 2020; Fernández-Delgado et al. 2014), as they can provide at least partial answers to such questions. The goal of this study is to provide an overview of existing approaches for encoding categorical predictor variables and to study their effect on a model's predictive performance. Following calls in the computational statistics community for neutral benchmark studies (Boulesteix et al. 2017), which do not introduce a new method, thus reducing the risk of cherry picking methods (Dehghani et al. 2021) and reporting over-optimistic performance (Nießl et al. 2021), we present a carefully designed experimental setting to discern the effect of encoding strategies and their interaction with different ML algorithms.

1.1 Notation

We consider the classical setting of supervised learning from an $i.i.d.$ tabular dataset $\mathcal{D}$ of size N sampled from a joint distribution $\mathcal{P}(\boldsymbol{x}, y)$ of a set of features $\boldsymbol{x}$ and an associated target variable y. Here, $\boldsymbol{x}$ consists of a mix of numeric (real-valued or integer-valued) features and categorical features, the latter of which we seek to transform feature-wise to numeric features using a categorical encoding technique. Let x be a single unordered categorical feature from a feature space $\mathcal{X}$ with cardinality $card(\mathcal{X}) \leq card(\mathbb{N})$. It holds either $y \in \mathbb{R}$ (regression), $y \in \mathcal{C}$ from a finite class space $\mathcal{C} = \{c_1, \ldots, c_C\}$ with $C = 2$ (binary classification) or $C > 2$ (multiclass classification). We always assume to observe all C classes in our training sample, however we might only observe a subset $\mathcal{L}^{train} \subseteq \mathcal{X}$ of a feature's available L levels, $\mathcal{L}^{train} = \{l_1, \ldots, l_L\}$ for categorical features. We denote the observed frequency of

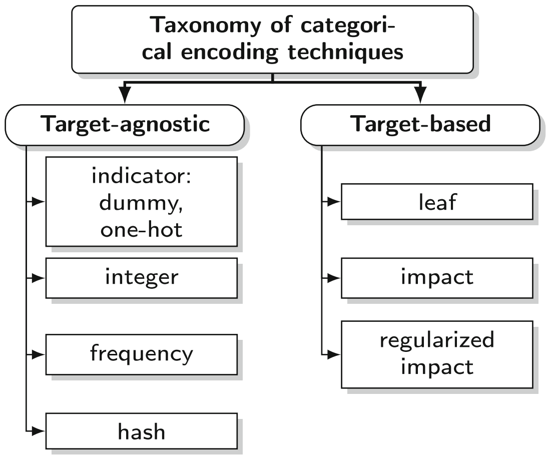


Fig. 1 Taxonomy of common categorical variable encoding techniques

class c in the training set with N_c and the observed frequency of a level l in the training set with N_l. We investigate categorical encoding techniques to transform each nominal feature x^{train} into numerical features $\hat{x}^{train}$ which are then used for training. If clear from the context, we use $\hat{x}_l$ as the encoded value for an observation with level l. Although datasets might contain multiple high cardinality features, we encode each feature separately but with the same strategy.

1.2 Related work

We broadly categorize feature encoding techniques into *target-agnostic* methods and *target-based* methods (Micci-Barreca 2001). Figure 1 contains a taxonomy of our considered encoding methods. *Target-agnostic* methods do not rely on any information about the target variable and can therefore also be used in unsupervised settings. Simple strategies from this domain e.g. one-hot or dummy encoding are widely used—in the scientific literature (Hancock and Khoshgoftaar 2020; Kuhn and Johnson 2019) but also on Kaggle[1] to embed variables for classical ML algorithms as well as (deep) neural networks. Such indicator methods map each level of a categorical variable to a set of dichotomous features encoding the presence or absence of a particular level. An obvious drawback of indicator encoding is that it adds one additional feature per level of a categorical variable. When indicator encoding leads to an unreasonable number of features, levels are often mapped to integer values with random order (integer encoding). Alternatively, the "hashing trick" (Weinberger et al. 2009) can be used to randomly collapse feature levels into a smaller number of indicator variables (Kuhn and Johnson 2019), or levels can be encoded by using the observed frequency of a given level in the dataset (frequency encoding).

Target-based methods try to incorporate information about the target values associated with a given level. Early strategies aimed to reduce the number of levels by methods like hierarchical clustering or decision trees based on statistics of the target variable, although this has been rarely described in the scientific literature (for a brief mention, see Micci-Barreca 2001). The basic idea of more advanced methods

[1] https://www.kaggle.com.

called target, impact, mean, or likelihood encoding is to use the training set to make a simple prediction of the target for each level of the categorical feature, and to use the prediction as the numerical feature value $\hat{x}_l$ for the respective level. An early formal description of this strategy is (Micci-Barreca 2001). In simple target encoding for regression problems, the mean target value in the training set from all observations with a certain feature level is used to encode that level for all observations: $\hat{x}_l = \frac{\sum_{i:x_i^{train}=l} y_i^{train}}{N_l}$. Simple target encoding often does not perform well with rare levels, where it tends to overfit to the training data and fails to generalize well for new observations. In the extreme case of a categorical feature with unique values (e.g. some hashed ID variable) studied in Prokhorenkova et al. (2018), the mean target for each level of this feature in simple target encoding is similar to the true target value of a single observation. Based on the encoded feature, all observations can be predicted perfectly in the training set, even if the original variable did not contain any useful information. ML models would place a high priority on such an encoded feature during training but would perform badly on test data. To avoid this, practitioners often use regularized target encoding with a smoothing parameter that shrinks those effects towards the global mean (Micci-Barreca 2001). An alternative strategy is to combine target encoding with cross-validation (CV) techniques (Prokhorenkova et al. 2018).

1.3 Categorical encoding benchmarks

Several small-scale studies have been previously conducted. Those works did not yield conclusive results due to narrower scopes or not considering high cardinality variables. One benchmark (6 datasets) on encoding high cardinality features (cardinality between 103 and 9095) in combination with gradient boosting has been published on the Kaggle forums (Prokopev 2018). Different versions of target encoding are compared with indicator, integer, and frequency encoding. They recommend combining the smoothed version of target encoding with 4- or 5-fold CV, never using simple target encoding and using indicator encoding only for small datasets. Interestingly, frequency encoding did perform well in many cases. Coors (2018) performed a benchmark (12 datasets) on encoding high cardinality features (maximum number of levels per dataset between 10 and 25,847) when developing the automatic gradient boosting (autoxgboost) library (Thomas et al 2018). They compared different variants of target encoding with integer and indicator encoding. In their benchmarks, target encoding only improved over target-agnostic methods on 2 datasets, while it led to worse results on 4 datasets. For a smaller number of levels, indicator and integer encoding yielded similar results. As those studies only consider gradient boosting and a limited amount of datasets, it is unclear whether results generalize to other datasets and ML algorithms. A recent study (15 datasets) found good performance of target encoding, but they only investigated categorical features in regression settings (Seca and Mendes-Moreira 2021). Several other publications studied encoding text data based on similarity (Cerda et al. 2018; Cerda and Varoquaux 2020) and employed indicator or target encoding as baselines. Another line of work studies variable encodings employed within specific ML models. Wright and König (2019) study treatments for categorical variables in random forests together with dummy and integer encoding (18 datasets, cardinality between

3 and 38), concluding that indicator and integer encoding perform subpar in comparison to methods that re-order levels according to the target variable. Prokhorenkova et al. (2018) compared their new CatBoost variant of target encoding to smoothed target encoding without CV, hold-out, and leave-one-out CV on 8 datasets. While the CatBoost method performed best, hold-out came second and target encoding without CV performed worst. An overview of encoding techniques tailored towards neural networks (e.g. the widely adopted *entity embeddings* by Guo and Berkhahn 2016) is provided in Hancock and Khoshgoftaar (2020). They present a survey of indicator- and embedding-based methods but no benchmark study. In contrast, our work is the first to focus on high cardinality variables and techniques that are agnostic to the subsequent ML method. We study this problem on a larger variety of datasets and settings.

The main goal of our study is to assess the impact different categorical encoding techniques have on subsequent models' predictive performance. As the optimal encoding might differ depending on the ML algorithm, we consider various state-of-the-art algorithms, *regularized linear models* (LASSO), *random forests* (RF), *gradient tree boosting* (GB), *k-nearest neighbors* (KNN), and *support vector machines* (SVM). To find default settings for high cardinality features, we analyze a variety of datasets with different characteristics including regression, binary classification, and multiclass classification problems. Because our methods vary in runtime and complexity, we are interested in whether more complex methods are to be preferred, or if simpler approaches suffice. We also study the relationship between a feature's cardinality and the choice of encoding technique by varying the minimum number of levels above which features are transformed.

1.4 Contributions

We survey a broad set of categorical encoding techniques and conduct a comprehensive benchmark study with a focus on high cardinality features. We carefully design a benchmark scenario as well as a preprocessing scheme allowing us to study 7 different encoding techniques in conjunction with 5 commonly used ML algorithms across 24 diverse datasets, both from a classification and a regression regime. We give a detailed description of our study design to highlight important considerations for studying high cardinality features. Our results provide an overview of the performance of various approaches heavily used in the literature. After a discussion of results concerning predictive performance, we provide further analyses seeking to inform practitioners which methods to apply. This includes an important discussion of runtimes.

2 Encodings

Pseudocode for all encoders is presented in the Supplementary Material. An important detail is how encoding techniques treat new levels during the prediction phase.

2.1 Integer encoding

The simplest strategy for categorical features is integer encoding (also called ordinal encoding). Observed levels from the training set are mapped to the integers 1 to L. Although new levels could be mapped to $L + 1$ or 0, model predictions would be arbitrary as the integer order does not carry information. Thus, we encode new levels as missing values and use mode imputation to obtain the integer which matches the most frequent level in the training set. Integer encoding should only be an acceptable strategy for tree-based models, which can separate all original levels with repeated splits.

2.2 Frequency encoding

Frequency encoding maps each level to its observed frequency in the training set ($\hat{x}_l = N_l$). This assumes a functional relationship between the frequency of a level and the target. It implicitly reduces the number of levels, and the subsequent model can best differentiate between levels with dissimilar frequencies. This approach is heavily used in natural language processing to encode token or n-gram counts. We encode new levels with a frequency of 1.

2.3 Indicator encoding

We use indicator encoding as an umbrella term for two common strategies to encode categorical features with a small to moderate number of levels: one-hot and dummy encoding. One-hot encoding transforms the original feature into L binary indicator columns, each representing one original level. An observation is coded with 1 for the indicator column representing its level ($x_i^{train} = l$) and 0 for all other indicators. Dummy encoding results in only $L - 1$ indicator columns. A reference feature level is chosen that is encoded with 0 in all indicator columns. For one-hot encoding, the zero vector can be used to encode new levels which were not observed during training. For dummy encoding, it is not useful to collapse new levels to the often arbitrary reference category; in our case the first level in alphabetical order. We replace new levels in the prediction phase with the most frequent level in the training set. As constructing all indicator variables is practically infeasible for high cardinality variables, we limit their number by collapsing rare levels beyond a varying threshold to a single *other* category before encoding.

2.4 Hash encoding

Hash Encoding can be used to compute indicator variables based on a hash function (Weinberger et al. 2009). The basic idea is to transform each feature level l into an integer $hash(l) \in \mathbb{N}$, based on its label. This integer is then transformed into an indicator representation, with 1 in indicator column number $(hash(l) \mod hash.size) + 1$ and 0 in all remaining columns (Kuhn and Johnson 2019). Some levels are hashed to the same indicator representation. The smaller the $hash.size$, the higher the number of collapsed levels. The number of indicators is often effectively lower than $hash.size$, as some indicators can be constant in the training set (we remove those columns for both training and prediction). Although we could jointly hash multiple features, we hash each feature separately to improve comparability with the other encoders.

2.5 Leaf encoding

Leaf encoding fits a decision tree on the training set to predict the target based on the categorical feature. Each level is encoded by the number of the terminal node, in which an observation with the respective level ends up. In that way, leaf encoding combines feature levels with similar target values. We use the rpart package in R (Therneau and Atkinson 2018) which grows CARTs with categorical feature support that can be pruned based on internal performance estimates from 10-fold CV. Thus, our leaf encoder automatically uses an "optimal" number of new levels. To speed up the computation for multiclass classification, our implementation uses the ordering approach presented in Wright and König (2019). New levels are encoded with the arbitrary number of the terminal node with most observations during training. Encoded values are treated as a new categorical feature and encoded by one-hot encoding. Our leaf encoder can be thought of as a simplification of the approach suggested by Grąbczewski and Jankowski (2003).

2.6 Impact encoding

An early formal description of a so-called impact, target or James-Stein encoder was provided by Micci-Barreca (2001). The basic idea is to encode each feature level with the conditional target mean (regression) or the conditional relative frequency of one or more target classes (classification). The impact encoder for classification uses a logit link and transforms the original feature into C numeric features, each representing one target class. A smoothing parameter ϵ is introduced to avoid division by zero. This parameter could be used to further regularize towards the unconditional mean. We choose a small $\epsilon = 0.0001$ as we want to compare simple target encoding with the regularized encoder introduced next. Weights of evidence encoding from the credit scoring classification literature (Hand and Henley 1997) is almost identical to impact encoding, but without regularization or centering.

2.7 GLMM encoding

Smoothed target encoding (Micci-Barreca 2001) can be interpreted as a simple (generalized) linear mixed model (glmm) in which the target is predicted by a random intercept for each feature level in addition to a fixed global intercept. This connection is described in Kuhn and Johnson (2019). To achieve regularized impact encoding, we implemented glmm encoders for regression, binary, and multiclass classification. The encoded value for each level is based on the spherical conditional mode estimates. In regression, the conditional modes are similar to the mean target value for each level, weighted by the relative observed frequency of that level in the training set (Bates 2020). The estimate of the fixed intercept can be used during the prediction phase to encode new feature levels not observed in the training set. In multiclass classification, we fit C one vs. rest glmms resulting in one encoded feature per class. An important advantage of using a glmm over impact encoding with a smoothing parameter is that a reasonable amount of regularization is determined automatically and tuning the complete ML pipeline is not necessary.

Overfitting can be further avoided through combination with cross-validation (CV) to train the encoder on independent observations without limiting the data to train the ML model. We provide an implementation which combines target encoding based on glmms with CV. During the training phase, we partition the data using CV into $n.folds$ and fit a glmm on each resulting training set. For each observation, there is exactly one glmm that did not use that observation for model fitting and can be safely used for encoding. Note that the $n.folds$ CV models (for $n.folds > 1$) are only used during the training phase. In the prediction phase, feature values are always encoded by a single glmm fitted to the complete training set. We study this method in three different settings: without CV (noCV), with $5-$ (5CV) and with $10-$ fold CV (10CV). In our study, we use the lmer (regression) and glmer (classification) functions from the lme4 package in R (Bates et al. 2015) as an efficient way to fit glmms.

2.8 Control conditions

We include three control conditions to better understand the effectiveness of the investigated encoders: The performance of a featureless learner (**FL** condition) was estimated as a conservative baseline for each dataset. In regression problems, FL predicts the mean of the target variable in the training set for each observation in the test set. In classification problems, the most frequent class of the target within the training set is predicted. For each dataset, we also consider a RF without encoding (**none** condition), to compare the use of encoding methods with a natural categorical splitting approach. The ranger (Wright et al. 2017) implementation provides efficient categorical feature support by ordering levels once before starting the tree growing algorithm (Wright and König 2019). In the **remove** high cardinality features control condition, we omit features with a high number of levels above some threshold and use one-hot encoding (without collapsing rare levels) for the remaining features. This condition reflects on whether including high cardinality features does indeed improve predictive performance. Otherwise, the best encoding might just provide the least impairment compared to not including any high cardinality features. We include an overview of available implementations in widely used ML frameworks for R and python in the Supplementary Material.

3 Benchmark setup

3.1 Datasets

A table showing a detailed summary of all benchmark datasets can be found in the Supplementary Material. We specifically investigate datasets that contain categorical variables with a large number of levels, including many well-known datasets from previous studies (Cerda et al. 2018; Coors 2018; Kuhn and Johnson 2019; Prokhorenkova et al. 2018). All datasets can be downloaded from the `OpenML` platform Vanschoren et al. (2013) based on the name or the displayed OmlId. The datasets include 8 regression, 10 binary classification, and 6 multiclass classification problems (between 3 and 12 classes). To assess the imbalance in categorical variables we computed the normalized entropy for each categorical variable. The maximum normalized entropy is 1, which corresponds to a uniform distribution, while a lower number indicates a larger imbalance. Sample sizes range between 736 and 1224158 observations. The total number of features ranges between 5 and 208. Datasets contain between 1 and 20 categorical features with more than 10 levels, with the maximum number of levels for a feature ranging between 14 and 30,114. Missing values are present in about half of the datasets.

3.2 High cardinality threshold

It is often assumed that advanced encoding methods are only advantageous for variables with a high number of levels, while simple indicator encoding is more appropriate for a small number of levels. To reflect this in our benchmark, a high cardinality threshold (HCT) parameter with values of 10, 25, and 125 was introduced determining varying configurations for the different encoders. For indicator encoding, the $HCT - 1$ most frequent levels are encoded together with a single collapsed category for the remaining levels. For integer, frequency, hash, leaf, impact, and glmm encoders, only features with more than HCT levels in the training set were encoded with the respective strategy, while the remaining categorical features were one-hot encoded. HCT is used as the hash size in hash encoding. In the remove control condition, features with more than HCT levels in the training set are removed from the feature set.[2]

3.3 Machine learning pipeline and algorithms

In total, we investigate 5 ML algorithms. We kept tuning their hyperparameters to a minimum because we were interested in the effect of the encoding techniques instead of a comparison of ML algorithms. LASSOs were fitted with `glmnet` (Friedman

[2] Based on the number of categorical features and levels per feature, some HCT settings were removed from the benchmark for some combinations of dataset X encoder. This ensured that encoders always affect at least one feature and that the remove condition always removes at least one feature. If settings where an encoder would lead to identical encoding strategies for all features of a dataset, we only kept the condition with the smallest HCT value.

et al. 2010), internally tuning the regularization using 5-fold CV. RFs with 500 trees were trained using `ranger` (Wright et al. 2017) without tuning, because RFs can be expected to give reasonable results with default settings (Probst et al. 2019). GB models were trained using `xgboost` (Chen et al. 2018), setting the learning rate to 0.01 and determining the number of iterations using early stopping on a 20% holdout set. KNN was taken from package `kknn` (Schliep and Hechenbichler 2016) standardizing features and using a constant $k = 15$ for the number of nearest neighbors, together with an information gain filter (Brown et al. 2012) to limit the number of features to 25. SVMs with radial basis function kernel were trained with `liquidSVM` (Steinwart and Thomann 2017). The bandwidth and regularization parameters were internally tuned using 5-fold CV. We used a one-vs-all approach for multiclass-settings.

The ML pipeline outlined below was used for all experimental conditions. It was carefully designed to ensure consistent results for extreme conditions (e.g. some levels only existing in the training data).

Imputation I Create a new factor level for missing values in categorical features with more than two categories. Impute missing values in binary features using the mode and missing values in numerical features using the mean feature value in the training data.

Encoding Transform the complete categorical features by the respective encoder. In the *no encoding* condition, the encoder simply passes on its input. The leaf and remove conditions still return categorical features, while the remaining encoders return only numerical features. Encoders only affect categorical variables above the specified HCT value.

Imputation II To handle new levels observed during prediction, impute missing values obtained during encoding.

Drop constants Drop features that are constant during training. As none of the original datasets includes constant columns, this step only removes constant features that are produced by the encoders or the CV splitting procedure.

Final one-hot encoding Transform all remaining categorical features via one-hot encoding (skipped for no encoding condition).

Learner Use the transformed data from each training set to fit the respective ML algorithm. In the prediction phase, transformed feature values (based on the trained encoder) for new observations in each test set are fed into the trained model to compute predictions.

3.4 Performance evaluation

We perform all analyses in the open-source statistical software R (R Core 2021). To enable a fair and reliable comparison in our study, we implemented all encoding methods on top of the `mlrCPO` package (Binder 2018). The pre-processing, as well as the final ML algorithm described in Sect. 3.3, were trained and resampled using the `mlr` framework (Bischl et al. 2016) together with the `batchtools` package (Lang et al. 2017) to scale the benchmark analysis to HPC compute infrastructure.

All materials for this study, including reproducible code for this manuscript and result objects can be downloaded from our **online repository**. [3]

Throughout our experiments, we use 5-fold CV to obtain estimates of predictive performance. Depending on the target variable, we report root mean squared error (RMSE) for regression, area under the curve (AUC) for binary classification, and its extension AUNU (Ferri et al. 2009) for multiclass problems.

In our benchmark study, each encoding method listed in Sect. 2 is combined with all ML algorithms (c.f. Sect. 3.3) across high cardinality thresholds (HCT) 10, 25, 125. While we only report results for the best HCT setting for each ML algorithm $\times$ dataset combination in Sect. 4, we study the effect of the HCT parameter in more detail in Sect. 4.5.

4 Benchmark results

Our main question is which encoding methods generally work well across various datasets. We report results for regression and classification datasets separately, as the associated metrics differ in scale.

For 6 datasets, some conditions with the SVM led to unexpected crashes due to memory problems or numerical errors. We completely removed those datasets for the SVM when computing ranks or other statistics that compare encodings across datasets.

4.1 Encoder performance

Mean performance estimates along with minimum and maximum performance are reported for all datasets in Figs. 2 (regression), 3 (binary classification), and 4 (multiclass classification). To reduce the complexity induced by the hyperparameter HCT we only display the parameter condition with the best performance for each combination of dataset $\times$ encoding $\times$ ML algorithm. The y-axis differs for all datasets and is reversed for the RMSE for better visual comparison. For some datasets, the remove condition performed very similar to the other encodings (e.g. *ames-housing*, *porto-seguro*), suggesting that categorical features were not informative. Performance in some CV folds was below the FL learner for *flight-delay-usa-dec-2017* ($RMSE_{FL} = 48.81$) and *nyc-taxi-green-dec-2016* ($RMSE_{FL} = 2.22$).

On datasets with substantive performance differences, target encoding with the glmm encoder was generally **most** effective. The **worst** encoder differed by ML algorithm and dataset. For datasets *Click_prediction_small*, *KDDCup09_upselling*, *kick*, and *okcupid-stem* some encoder $\times$ ML algorithm conditions performed worse than simply removing high cardinality features.

4.2 Meta rankings and dataset clustering

To further analyze our results, we used statistical inference methods inspired by the benchmark community in computational statistics (e.g., Hothorn et al. 2005). First,

[3] https://github.com/slds-lmu/paper_2021_categorical_feature_encodings.

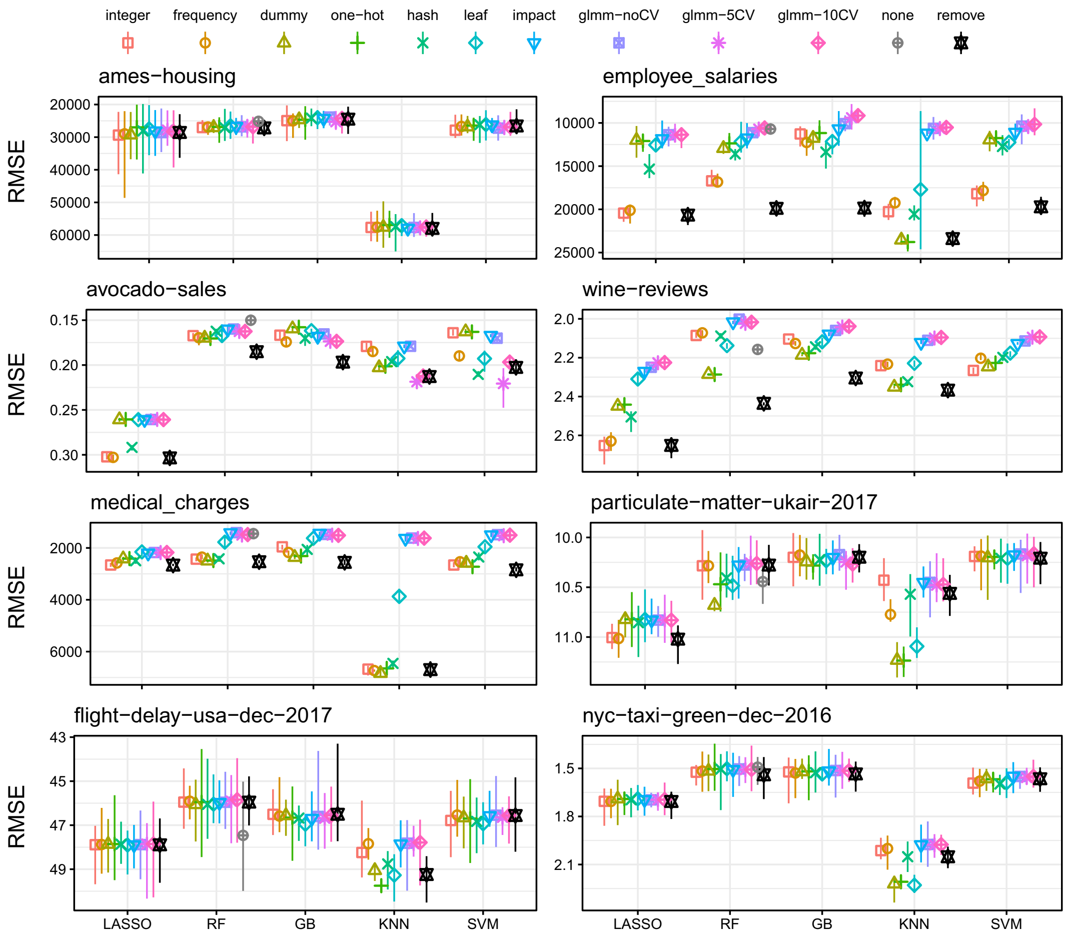


Fig. 2 Performance estimates from 5-CV for regression (mean, min, max). For each combination, only the best HCT condition is displayed. Note the reversed y-axis to ease visual interpretation

we present meta rankings for each ML algorithm in Fig. 5: We defined an encoder relation within each dataset, based on corrected resample t-tests (Nadeau and Bengio 2003). An encoding was defined to beat another encoding if the one-sided p-value of the t-test was $< .05$. This allowed us to compute a weak-order consensus ranking R defined by the optimization problem:

$$\underset{R \in C}{arg\ min} \sum_{b=1}^{B} d(R_b, R)$$

where d is the symmetric difference distance and R_b is the relation for dataset b (Hornik and Meyer 2007; Meyer and Hornik 2018). The symmetric difference between two relations is the number of cases one encoding beats another encoding in one relation but not in the other one.

Although the presented solutions of the optimization problem are not unique, rankings were highly stable for the high and low ranks. Meta rankings seem to be highly consistent with the individual patterns of encoder performances reflected in Figs. 2 to 4. Looking at meta-rankings, approaches based on GLMM's in combination with

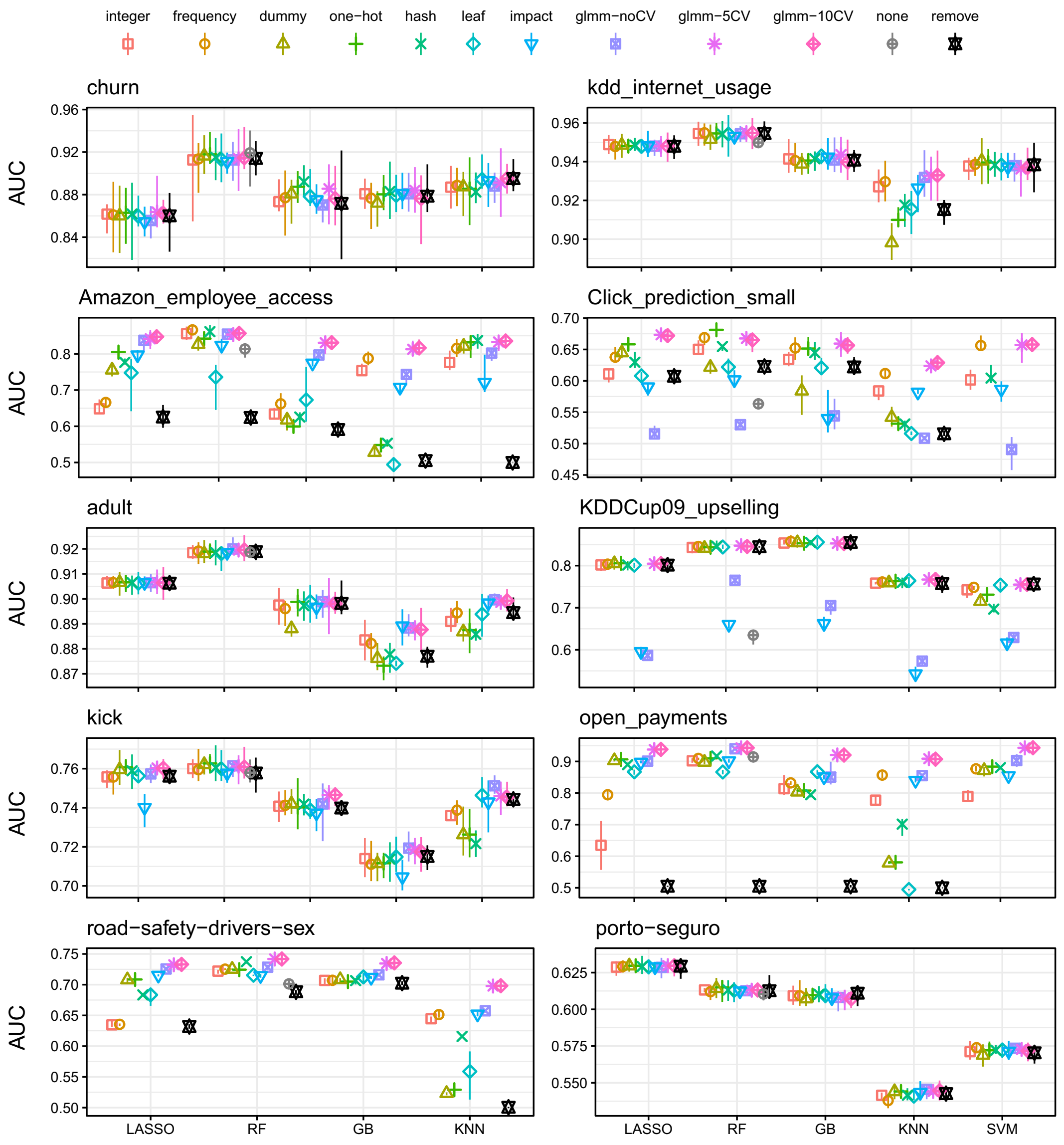


Fig. 3 Performance estimates from 5-CV for binary classification (mean, min and max). For each combination, only the best HCT condition is displayed

cross-validation outperform all other approaches across all algorithms. A further interesting detail omitted in Fig. 5 for clarity is that the *none* encoding strategy for the RF was beaten by all strategies except for the *remove* condition. This implies, that even if the algorithm provides a mechanic for treating categorical variables, it might often be optimal to use a different strategy instead.

In a second exploratory analysis, we tried to find clusters of datasets based on systematic patterns of encoder performance (independent of the employed ML algorithm). We computed a partial-order consensus relation across ML algorithms for each dataset and hierarchically clustered the dataset consensus relations using the symmetric difference distance in combination with the complete linkage agglomeration method. The resulting dendrogram is displayed in Fig. 6. Although the cluster structure is somewhat ambiguous, roughly three clusters can be described: The first 9 datasets

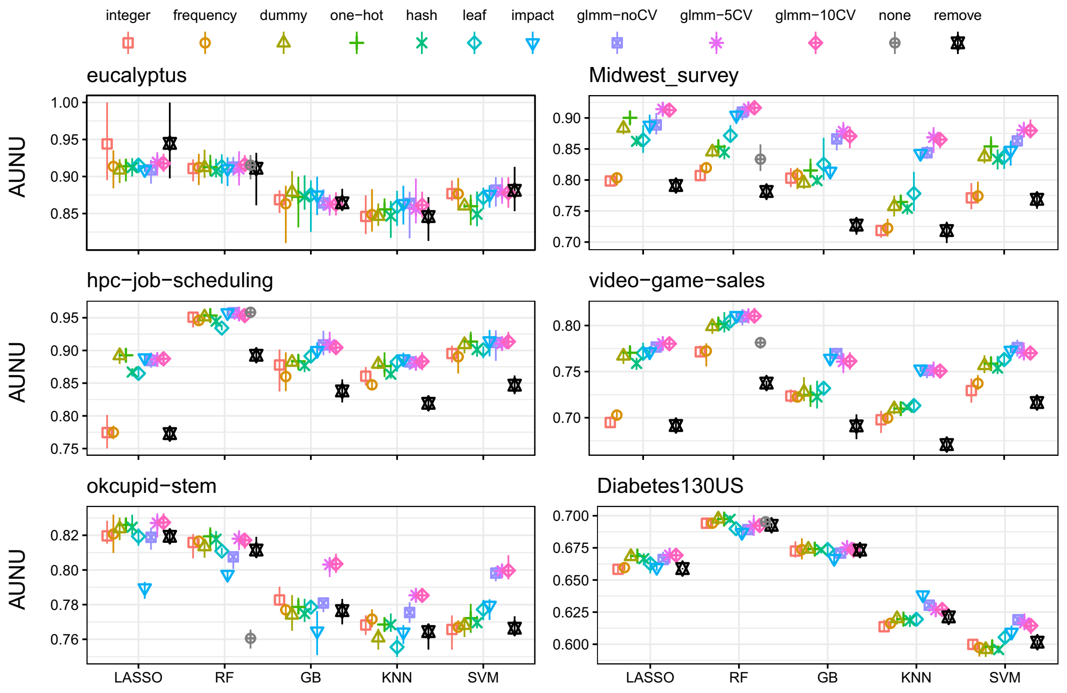
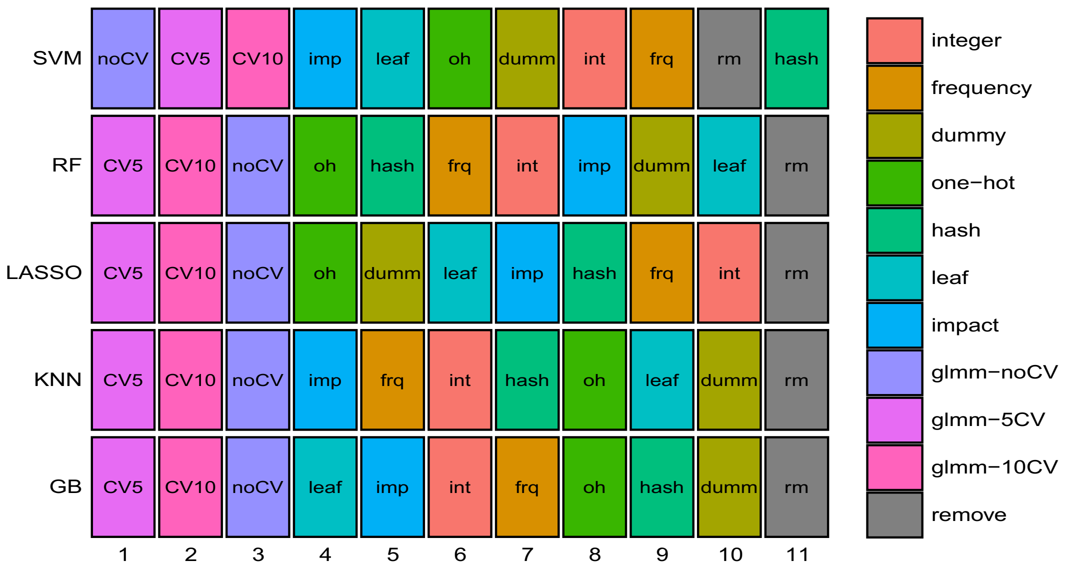


Fig. 4 Performance estimates from 5-CV for multiclass classification (mean, min and max). For each combination, only the best HCT condition is displayed



	1	2	3	4	5	6	7	8	9	10	11
SVM	noCV	CV5	CV10	imp	leaf	oh	dumm	int	frq	rm	hash
RF	CV5	CV10	noCV	oh	hash	frq	int	imp	dumm	leaf	rm
LASSO	CV5	CV10	noCV	oh	dumm	leaf	imp	hash	frq	int	rm
KNN	CV5	CV10	noCV	imp	frq	int	hash	oh	leaf	dumm	rm
GB	CV5	CV10	noCV	leaf	imp	int	frq	oh	hash	dumm	rm

🟥	integer
🟧	frequency
🟫	dummy
🟩	one−hot
🟩	hash
🟦	leaf
🟦	impact
🟪	glmm−noCV
🟪	glmm−5CV
🟪	glmm−10CV
⬛	remove

Fig. 5 Consensus rankings across all datasets for each algorithm. Lower ranks indicate better performance. The rank of the *none* control condition of the RF (rank 11 of 12) was omitted from the figure

from the top of the dendrogram are characterized by a medium to high number of levels, low performance of the remove condition (indicating the importance of high cardinality features) and clear performance advantage of target encoders. For the next 13 datasets which contain the smallest number of levels, traditional encodings can compete with target encoding. This biggest cluster also includes 9 datasets with zero distances, in which no significant performance differences could be observed between

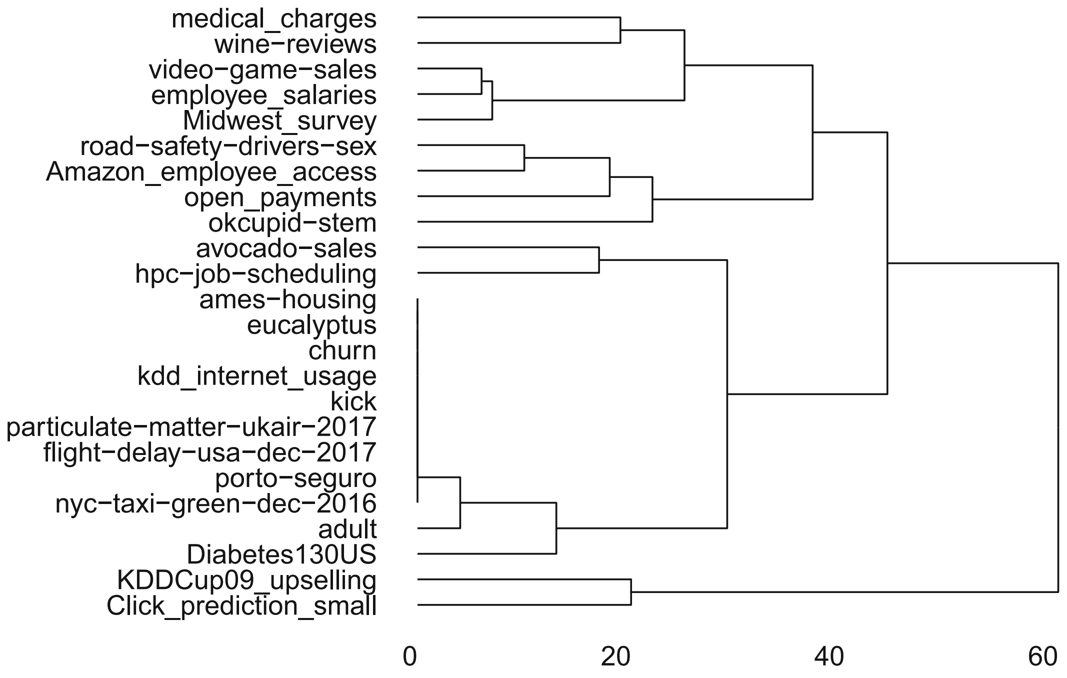


Fig. 6 Hierarchical cluster analysis of benchmark datasets. The symmetric difference distance between two datasets reflects differences in performance patterns between encodings (independent of the employed ML algorithm)

any encoding conditions (nor with the remove condition, indicating that high cardinality features are less informative). The last two datasets with the highest number of levels formed a separate cluster, in which target encoding without strong regularization (impact, glmm-noCV) showed severe overfitting. Note that clusters were not determined by problem type, again suggesting that encoder rankings are somewhat similar for regression and classification settings.[4]

4.3 Summary of encoder performance

Regularized target encoding was superior or at least competitive on all datasets. We could not observe a setting in which regularized target encoding was convincingly beaten by target agnostic methods. Especially effective was glmm encoding with 5-fold-CV, which ranked first place for all ML algorithms except SVM. Performance often did not improve with glmm-10CV, suggesting that 5 folds might be a good regularization default in practice. In line with earlier research (Micci-Barreca 2001; Prokhorenkova et al. 2018), target encoding with regularization (glmm) performed better or equally well in comparison with the unregularized impact encoder. When glmm performed better, impact encoding not only performed worse than glmm with CV but was sometimes also inferior compared to other encoders.

[4] We tried to corroborate this observation by comparing meta-rankings between problem types. Unfortunately, problem type specific consensus relations did not converge (probably due to the reduced number of datasets) and were uninterpretable.

The following observations were also interesting: Surprisingly, integer encoding did not perform well with GB and target-based encoders (especially the glmm encoder) seem to be preferable. For LASSO, previous studies have suggested that indicator encoding works well, even with a very high number of levels (Cerda et al. 2018). Although the glmm encoders ranked first in our benchmark for the LASSO, it was the only algorithm where the indicator encoders achieved the next best ranking. Note that for computational reasons we limited the maximum amount of indicator variables per feature to 125 in our experimental design. The HCT = 125 setting performed best for LASSO with indicator encoding in a large number of datasets, indicating that performance might have further improved with higher values. Both KNN and SVM rely on numerical distances in feature space and tend to perform poorly in the presence of high dimensionality. Thus, we expected that target encoding should work well here as it transforms categories into a single, smooth numerical feature. This was backed by our benchmark results. For some datasets (*medical_charges*, *road-safety-drivers-sex*, and *Midwest_survey*) KNN (without tuning of the optimal number of nearest neighbors) could compete with the more sophisticated ML algorithms when combined with target encoding, but performed poorly with other encoders. Although consistent with the big picture, SVM results have to be considered with care, as some experimental conditions resulted in unexpected computational errors. In RF, a widely used strategy to deal with categorical features is to order levels by average target statistics for a given level. For a small number of levels, this approach has been reported superior to indicator and integer encoding (Wright and König 2019), while we observed poor performance for datasets with a larger number of levels.

When looking for a simple default encoding, indicator encoding in combination with collapsing small levels seems a robust alternative, although the glmm encoders performed better. We found that one-hot encoding usually gave a slightly better performance than dummy encoding, which has also been observed by Chiquet et al. (2016); Tutz and Gertheiss (2016) (p. 254). Our results suggest that one-hot encoding is the better standard compared to dummy encoding when applying nonlinear regularized models like RF, GB or SVM with (high cardinal) categorical features. We provide a more detailed comparison for indicator encoding in the Supplementary Material.

4.4 Runtime analysis

To determine whether traditional encodings are preferred when facing limited computational resources, we further analyzed runtimes of the whole analysis pipeline for different encoders and ML algorithms. Aggregated results are shown in Table 1. To enable a meaningful comparison, we report runtime as the fraction of a full pipeline's strategy compared to the one-hot encoding condition and then further aggregate across datasets using the median. Again, we only report the **best** HCT setting. Absolute runtimes are hard to interpret and aggregate because runtime distributions across datasets are heavily skewed as proportionally larger runtimes are observed for big datasets. While a pipeline's runtime can be dominated by the encoder for small datasets, the training of the consecutive ML algorithm is dominating for large datasets, which might render differences during encoding irrelevant. Therefore we aim to report what

Table 1 Proportional increase in runtime compared to one-hot encoding

Encoding	LASSO	RF	GB	KNN	SVM
Integer	$0.4_{0}^{1.4}$	$0.59_{0}^{1.1}$	$0.67_{0}^{0.9}$	$0.85_{0.4}^{1.5}$	$1.01_{0.6}^{1.6}$
Frequency	$0.34_{0}^{2.1}$	$0.54_{0}^{1.7}$	$0.54_{0}^{1.5}$	$0.79_{0.2}^{1.1}$	$1.11_{0.5}^{1.7}$
Dummy	$1.13_{0.8}^{2.8}$	$0.91_{0.5}^{1.7}$	$0.97_{0.4}^{5.7}$	$1.05_{0.6}^{1.6}$	$1.02_{0.7}^{1.5}$
One-hot	1_{1}^{1}	1_{1}^{1}	1_{1}^{1}	1_{1}^{1}	1_{1}^{1}
Hash	$0.87_{0.2}^{2.6}$	$1.03_{0.4}^{3.3}$	$0.98_{0.4}^{7.3}$	$0.93_{0.5}^{2.2}$	$1.16_{0.5}^{2}$
Leaf	$0.46_{0}^{1.7}$	$0.67_{0}^{2.3}$	$0.85_{0}^{9.1}$	$0.97_{0.5}^{2.2}$	$1.01_{0.6}^{1.9}$
Impact	$0.5_{0.1}^{1.9}$	$0.6_{0.1}^{1.5}$	$0.82_{0}^{120.5}$	$1.11_{0.3}^{24}$	$1.06_{0.7}^{1.8}$
glmm-noCV	$0.58_{0}^{2.8}$	$0.57_{0.1}^{3.8}$	1.65_{0}^{12}	$1.09_{0.3}^{10.3}$	$1.1_{0.7}^{2.1}$
glmm-5CV	$0.83_{0.1}^{2.2}$	$0.66_{0.1}^{15.3}$	$7.21_{0}^{69.3}$	$2.77_{0.7}^{50.1}$	$1.41_{0.6}^{4.6}$
glmm-10CV	$1.07_{0.1}^{4.2}$	$0.96_{0.1}^{28.4}$	$12.71_{0}^{127.6}$	$5.37_{0.8}^{97.9}$	$1.55_{0.5}^{4.6}$
None		$0.2_{0}^{1.4}$			
Remove	$0.4_{0}^{1.7}$	$0.48_{0}^{1.1}$	$0.58_{0}^{1.3}$	$0.82_{0.2}^{1.2}$	$1.09_{0.6}^{2}$

Median $_{\text{min}}^{\text{max}}$ across datasets of the proportional increase in runtime from 5-CV, when comparing the respective encoder with one-hot encoding. Only the best HCT conditions are reported

is important in practice, the time differences for training the **full pipeline**. The results clearly show that regularized target encoding does not consistently yield slower runtimes compared to simple strategies like indicator encoding. Supposedly, the more efficient representations produced by target encoding lead to faster runtimes of subsequent ML algorithms. This suggests that a possible runtime vs. predictive performance trade-off might also be in favour of target encoding, especially for large datasets where a high number of indicator variables increases computational load. We did observe a substantial increase in runtimes when using regularized glmm with GB (with little tuning), where runtimes are relatively short in comparison to the time required to fit categorical encoders. In other settings (LASSO, RF), the glmm encoders have been observed to be even faster than indicator encoding.

4.5 Analysing high cardinality thresholds

Until now, we ignored the HCT parameter by reporting only the condition with the best performance. An interesting question is whether target encoding is only useful for features with a very high number of levels or also for fewer levels, where most practitioners would routinely use indicator encoding. We tested this using HCT thresholds of 10, 25 and 125, but the results are not easy to interpret. In general, the optimal threshold seemed to strongly depend on the dataset, but we also observed some weak patterns: LASSO improved for larger HCT, indicating that its internal regularization can efficiently deal with the sparseness induced by indicator encoding. In comparison, other methods generally yielded better performance if features above the very low HCT of 10 were encoded using one of the target encoding strategies. Differences in regularization for the target based encoders seemed not to prefer different HCT values.

5 Discussion

In our benchmark, we compared encoding strategies for high cardinality categorical features on a variety of regression, binary, and multiclass classification datasets with different ML algorithms. Regularized target encoding was superior across most datasets and ML algorithms. Although the performance of other encoding strategies was comparable in some conditions, target encoding was never outperformed. In general, our results suggest that regularized target encoding based on glmms with 5-fold CV (glmm-5CV) works well for all kinds of algorithms and should be a reasonable default strategy. It sometimes leads to slightly longer runtimes (in comparison to indicator encoding), but especially for larger datasets this is often offset by the more efficient representation produced by target encoding. The glmm encoder has a clear advantage over target encoding with a smoothing hyperparameter (Micci-Barreca 2001), as costly tuning the whole ML pipeline with different smoothing values is not required.

What constitutes a “high” cardinality problem is a difficult question that might not only depend on the number of levels but also on other characteristics of the dataset and its features. Supposedly, the number of datasets in our study was too small to discover consistent patterns between encoder performance and dataset characteristics. Target encoding features with only 10 levels seemed to be effective in a substantive number of conditions, but for other datasets, higher HCT values performed better. Thus, some form of hyperparameter tuning seems necessary to decide the level threshold for target encoding at this point, as no suitable defaults seem to be available. Note that we compared different HCT values but then used the same encoding strategy for all affected features alongside one-hot encoding for the remaining ones. A further improvement could be to decide whether to use target encoding on a feature by feature basis. ML pipelines could introduce a categorical hyperparameter for each feature that represents which encoding is used. To make this complicated meta optimization problem feasible, only a small number of encoders can be included. Our study can help to decide which encoders could be safely omitted from consideration.

5.1 Limitations

Several design decisions were necessary to make answering our research questions computationally feasible: We used minimal tuning for our ML algorithms, as we were not interested in comparing their performance against each other. This probably led to suboptimal performance for the GB, KNN, and SVM learners. When interpreting our results, we assume that the encoder rankings are comparable when more extensive tuning is used. This is plausible because the meta rankings between algorithms were highly stable. We only used 5-fold-CV without repetitions to estimate predictive performance, but due to a small variance between folds, we could confidently detect performance differences of encoders for many datasets. Because we are interested in model-agnostic methods that can be combined with any supervised ML algorithm, we did not investigate recent model-specific strategies (Guo and Berkhahn 2016; Prokhorenkova et al. 2018). We did not include deep neural networks in our benchmark because they are still rarely applied in the statistical learning community and more dominant in computer science. We do not differentiate between nominal and ordinal categorical features: Most publicly available datasets simply do not contain any high cardinal ordinal features. For many applications, ordinal feature information is not available as metadata which makes it less relevant for some applications like automatic ML. Thus, our benchmark does not include specific ordinal encoding methods like Helmert and Polynomial contrasts (Chambers and Hastie 1992) or ordinal penalization methods (Tutz and Gertheiss 2016). We only investigate traditional categorical variables and do not extend our analysis to multi-categorical variables or text-strings (Cerda et al. 2018; Cerda and Varoquaux 2020), where other encoding techniques can harvest additional information. We only investigate univariate encodings, i.e. we always encode each variable separately. Levels with comparable main effects can not be distinguished based on the transformed feature, which prevents the consecutive ML algorithm from learning interactions for specific levels. Methods that jointly encode features and therefore leverage correlational structures between features are an interesting avenue for future research: Recent approaches use target-based encoding strategies to learn a vector-valued representation of a level, similar to neural network embeddings (Guo and Berkhahn 2016; Rodríguez et al. 2018). Those should be compared to classical approaches from the optimal scaling literature (e.g. De De Leeuw et al. 1976; Young et al. 1976) or the subsequent *aspect* framework (Mair and Leeuw 2010), which are unfortunately unfamiliar to many ML researchers.

5.2 Conclusion

This benchmark study compared the predictive performance of a variety of strategies to encode categorical features with a high number of unordered levels for different supervised ML algorithms. Regularized versions of target encoding, which uses predictions of the target variable as numeric feature values, performed better than traditional strategies like integer or indicator encoding. Most effective, with a consistently superior performance across ML algorithms and datasets, was a target encoder which combines simple generalized linear mixed models with cross-validation and does not require hyperparameter tuning. GLMMs are a major workhorse in applied statistics but not well understood and often neglected by the ML community. Refining target encoders that use statistical models for more efficient regularization and studying their theoretical properties could be a valuable research topic for the computational statistics community.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s00180-022-01207-6.

Funding Open Access funding enabled and organized by Projekt DEAL. This work has been funded by the German Federal Ministry of Education and Research (BMBF) under Grant No. 01IS18036A and by the Bavarian Ministry of Economic Affairs, Regional Development and Energy through the Center for Analytics–Data–Applications (ADA-Center) within the framework of, BAYERN DIGITAL II“ (20-3410-2-9-8).

Data Availability All benchmark datasets are publicly available at https://www.openml.org/.

Code Availability All analysis code is publicly available at https://github.com/slds-lmu/paper_2021_categorical_feature_encodings.

Declaration

Conflicts of interest The authors declare no conflict of interest.

References

Bates D (2020) Computational methods for mixed models. Vignette for lme4. https://cran.r-project.org/web/packages/lme4/vignettes/Theory.pdf

Bates D, Mächler M, Bolker B, Walker S (2015) Fitting linear mixed-effects models using lme4. J Stat Softw 67:1–48. https://doi.org/10.18637/jss.v067.i01

Binder M (2018) mlrCPO: Composable preprocessing operators and pipelines for machine learning. R package version 0.3.4-2. https://github.com/mlr-org/mlrCPO

Bischl B, Lang M, Kotthoff L, Schiffner J, Richter J, Studerus E, Casalicchio G, Jones ZM (2016) mlr: machine learning in r. J Mach Learn Res 17:1–5

Bommert A, Sun X, Bischl B, Rahnenführer J, Lang M (2020) Benchmark for filter methods for feature selection in high-dimensional classification data. Comput Stat Data Anal. https://doi.org/10.1016/j.csda.2019.106839

Boulesteix A-L, Binder H, Abrahamowicz M, Sauerbrei W et al (2017) On the necessity and design of studies comparing statistical methods. Biomet J Biomet Zeitschrift 60:216–218. https://doi.org/10.1002/bimj.201700129

Brown G, Pocock A, Ming-Jie Z, Luján M (2012) Conditional likelihood maximisation: a unifying framework for information theoretic feature selection. J Mach Learn Res 13:27–66

Cerda P, Varoquaux G (2020) Encoding high-cardinality string categorical variables. IEEE Trans Knowl Data Eng. https://doi.org/10.1109/TKDE.2020.2992529

Cerda P, Varoquaux G, Kégl B (2018) Similarity encoding for learning with dirty categorical variables. Mach Learn 107:1477–1494. https://doi.org/10.1007/s10994-018-5724-2

Chambers J, Hastie T (1992) Statistical models. Chapter 2 of statistical models in S, 1st edn. Routledge. https://doi.org/10.1201/9780203738535

Chen T, He T, Benesty M, Khotilovich V,Tang Y, Cho H, Chen K, Mitchell R, Cano I, Zhou T, Li M, Xie J, Lin M, Geng Y, Li Y (2018) Xgboost: Extreme gradient boosting. R package version 0.71.2. https://CRAN.Rproject.org/package=xgboost

Chiquet J, Grandvalet Y, Rigaill G (2016) On coding effects in regularized categorical regression. Stat Modell 16:228–237. https://doi.org/10.1177/1471082X16644998

Coors S (2018) Automatic gradient boosting (Master'sthesis). LMU Munich. https://epub.ub.uni-muenchen.de/59108/1/MA_Coors.pdf

De Leeuw J, Young FW, Takane Y (1976) Additive structure in qualitative data: an alternating least squares method with optimal scaling features. Psychometrika 41:471–503

Dehghani M, Tay Y, Gritsenko AA, Zhao Z, Houlsby N, Diaz F, Metzler D, Vinyals O (2021) The benchmark lottery. arXiv preprint arXiv:2107.07002

Fernández-Delgado M, Cernadas E, Barro S, Amorim D (2014) Do we need hundreds of classifiers to solve real world classification problems? J Mach Learn Res 15:3133–3181

Ferri C, Hernández-Orallo J, Modroiu R (2009) An experimental comparison of performance measures for classification. Pattern Recogn Lett 30:27–38. https://doi.org/10.1016/j.patrec.2008.08.010

Feurer M, Klein A, Eggensperger K, Springenberg J, Blum M, Hutter F (2015) Efficient and robust automated machine learning. In: Cortes C, Lawrence ND, Lee DD, Sugiyama M, Garnett R (eds) Advances in neural information processing systems 28. Curran Associates Inc, New York, pp 2962–2970

Friedman J, Hastie T, Tibshirani R (2010) Regularization paths for generalized linear models via coordinate descent. J Stat Softw 33:1–22. https://doi.org/10.18637/jss.v033.i01

Gelman A, Hill J (2006) Data analysis using regression and multilevel/hierarchical models. Cambridge University Press, Cambridge

Grąbczewski K, Jankowski N (2003) Transformations of symbolic data for continuous data oriented models. In: Kaynak O, Alpaydin E, Oja E, Xu L (eds) Artificial neural networks and neural information processing – ICANN/ICONIP 2003. Springer, Berlin, Heidelberg, pp 359–366

Guo C, Berkhahn F (2016) Entity embeddings of categorical variables. arXiv preprint arXiv:1604.06737

Hancock JT, Khoshgoftaar TM (2020) Survey on categorical data for neural networks. J Big Data 7:1–41. https://doi.org/10.1186/s40537-020-00305-w

Hand DJ, Henley WE (1997) Statistical classification methods in consumer credit scoring: a review. J R Stat Soc A Stat Soc 160:523–541. https://doi.org/10.1111/j.1467-985X.1997.00078.x

Hornik K, Meyer D (2007) Deriving consensus rankings from benchmarking experiments, In: Advances in data analysis. Springer, pp 163–170. https://doi.org/10.1007/978-3-540-70981-7_19

Hothorn T, Leisch F, Zeileis A, Hornik K (2005) The design and analysis of benchmark experiments. J Comput Graph Stat 14:675–699. https://doi.org/10.1198/106186005X59630

Kuhn M, Johnson K (2019) Feature engineering and selection: a practical approach for predictive models. Hall/CRC, Chapman

Lang M, Bischl B, Surmann D (2017) Batchtools: tools for r to work on batch systems. J Open Source Softw. https://doi.org/10.21105/joss.00135

Mair P, de Leeuw J (2010) A general framework for multivariate analysis with optimal scaling: the r package aspect. J Stat Softw 32:1–23. https://doi.org/10.18637/jss.v032.i09

Meyer D, Hornik K (2018) Relations: data structures and algorithms for relations

Micci-Barreca D (2001) A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems. SIGKDD Explor Newsl 3:27–32. https://doi.org/10.1145/507533.507538

Nadeau C, Bengio Y (2003) Inference for the generalization error. Mach Learn 52:239–281. https://doi.org/10.1023/A:1024068626366

Nießl C, Herrmann M, Wiedemann C,Casalicchio G, Boulesteix A-L (2021) Over-optimism in benchmark studies and the multiplicity of design and analysis options when interpreting their results. WIREs Data Mining and Knowledge Discovery, e1441. https://doi.org/10.1002/widm.1441

Probst P, Wright MN, Boulesteix A-L (2019) Hyperparameters and tuning strategies for random forest. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery. https://doi.org/10.1002/widm.1301

Prokhorenkova L, Gusev G, Vorobev A, Dorogush AV, Gulin A (2018) CatBoost: Unbiased boosting with categorical features, in: Bengio S, Wallach H, Larochelle H, Grauman K, Cesa-Bianchi N, Garnett R (Eds.), Advances in Neural Information Processing Systems 31. Curran Associates, Inc., pp. 6638–6648

Prokopev V (2018) Mean (likelihood) encodings: a comprehensive study. Kaggle Forums

R Core Team (2021) R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria

Rodríguez p, Bautista MA, Gonzàlez J, Escalera S (2018) Beyond one-hot encoding: lower dimensional target embedding. Image Vis Comput 75:21–31. https://doi.org/10.1016/j.imavis.2018.04.004

Schliep K, Hechenbichler K (2016) Kknn: Weighted k-nearest neighbors R package version 1.3.1. https://CRAN.R-project.org/package=kknn

Seca D, Mendes-Moreira J (2021) Benchmark of encoders of nominal features for regression. In: Rocha Á, Adeli H, Dzemyda G, Moreira F, Ramalho Correia AM (eds) Trends and applications in information systems and technologies. Springer International Publishing, Cham, pp 146–155

Steinwart I, Thomann P (2017) liquidSVM: A fast and versatile SVM package. arXiv: 1702:06899

Therneau T, Atkinson B (2018) Rpart: recursive partitioning and regression trees. R package version 4.1-13. https://CRAN.R-project.org/package=rpart

Thomas J, Coors S, Bischl B (2018) Automatic gradient boosting. arXiv preprint arXiv:1807.03873

Thornton C, Hutter F, Hoos HH, Leyton-Brown K (2013) Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms, In: Proceedings of the 19th ACM SIGKDD international conference on knowledge discovery and data mining, KDD '13. ACM, New York, NY, USA, pp 847–855. https://doi.org/10.1145/2487575.2487629

Tutz G, Gertheiss J (2016) Rejoinder: Regularized regression for categorical data. Stat Model 16:249–260. https://doi.org/10.1177/1471082X16652780

Vanschoren J, van Rijn N, Bischl B, Torgo L (2013) OpenML: networked science in machine learning. SIGKDD Explor 15:49–60. https://doi.org/10.1145/2641190.2641198

Weinberger KQ, Dasgupta A, Langford J, Smola AJ, Attenberg J (2009) Feature hashin for large scale multitask learning. In: Proceedings of the 26th Annual International Conference on Machine Learning (ICML '09). Association for Computing Machinery, New York, NY, USA, 1113–1120. https://doi.org/10.1145/1553374.1553516

Wright MN, König IR (2019) Splitting on categorical predictors in random forests. PeerJ 7. https://doi.org/10.7717/peerj.6339

Wright MN, Ziegler A (2017) Ranger: a fast implementation of random forests for high dimensional data in C++ and R. J Stat Softw 77:1–17. https://doi.org/10.18637/jss.v077.i01

Young FW, De Leeuw J, Takane Y (1976) Regression with qualitative and quantitative variables: an alternating least squares method with optimal scaling features. Psychometrika 41:505–529. https://doi.org/10.1007/BF02296972

6.3 Evaluating Domain Generalization for Survival Analysis in Clinical Studies

Contributed Article:
F. Pfisterer, C. Harbron, G. Jansen, and T. Xu. Evaluating domain generalization for survival analysis in clinical studies. In G. Flores, G. H. Chen, T. Pollard, J. C. Ho, and T. Naumann, editors, *Proceedings of the Conference on Health, Inference, and Learning*, volume 174 of *Proceedings of Machine Learning Research*, pages 32–47. PMLR, 07–08 Apr 2022

Declaration of contributions The main part of the project was conducted while FP was interning with Fa. Hofmann-La Roche AG. The initial project idea was contributed by TX. FP collected and implemented relevant methods and conducted the core benchmark with additional input and ideas by TX and GJ. FP and TX jointly created the manuscript with additional feedback and helpful comments by GJ and CH.

Evaluating Domain Generalization for Survival Analysis in Clinical Studies

Florian Pfisterer florian.pfisterer@stat.uni-muenchen.de
F.Hoffmann-La Roche AG, Switzerland; Ludwig-Maximilians-Universität München, Germany

Chris Harbron chris.harbron@roche.com
Roche Products Ltd, United Kingdom

Gunther Jansen gunther.jansen@roche.com
F.Hoffmann-La Roche AG, Switzerland

Tao Xu tao.xu.tx1@roche.com
F.Hoffmann-La Roche AG, Switzerland

Abstract

Machine learning models are often required to generalize to new populations (domains) unseen during training, which may lead to model underperformance. So far, most research has focused on Domain Generalization methods for image classification tasks, which address the problem by learning domain invariant predictors. In this study, we assess the efficacy of domain generalization methods in survival analysis. The goal is to predict time-to-events such as death or disease progression based on baseline demographic and clinical variables of individuals exposed to medical treatment. We benchmark four domain generalization methods and several conventional/established methods on real world scenarios encountered in clinical practice. This includes tasks such as generalizing between randomized controlled trials to real world data, identification of prognostic models regardless of treatment or disease subtypes. We find that the generalization issue is often not as severe as reported in synthetic scenarios. Furthermore, our results corroborate previous findings that domain generalization often does not consistently outperform classical empirical risk minimization baselines also on low-dimensional data. Finally, to better understand when domain generalization methods can lead to performance gains and thus better outcomes for patients, we quantify the influence of different types of shifts occurring in the data.

Data and Code Availability Our study includes patient outcome and baseline characteristics data collected from several clinical studies on non-small-cell lung carcinoma (NSCLC) and diffuse large B-cell lymphoma (DLBCL). A short description of each dataset along with references is available in the Appendix. Details of the clinical trials can be found on https://clinicaltrials.gov/. Datasets are not publicly available at the time of writing, please contact the study team to obtain data access. Legal review for code sharing is in progress and the code cannot be shared by the time of manuscript submission. Please contact the authors regarding the access to the code.

1. Introduction

Machine learning is increasingly important in medical research. It can be used for a broad array of tasks ranging from improving the understanding of biological or chemical processes, automating and enhancing physician capabilities (e.g., by providing additional annotations or a second opinion in radiology) or providing additional diagnostic scores to predict patient survival. A central assumption for these tasks is that new data points stem from the same underlying distribution as that on which a machine learning model was trained (Widmer and Kubat, 1996; Quiñonero-Candela et al., 2009). This is a reasonable assumption when a large enough and representative data sample about the population we wish to predict for can be collected, or when variation among individuals is limited and central relationships between data and the corresponding property of interest are stable across sub-populations. Unfortunately, this is often not the case in clinical scenarios, e.g., when data is collected across different sub-populations with access to different hospitals, standards of care and disease heterogeneity (Challen et al., 2019). In such clinical generalization scenarios, clinical models often show lower predictive performance in medical imaging applications (Zhang et al., 2021) such as radiography (Zech et al., 2018; Pooch et al., 2019; Cohen et al., 2020) and MRI imaging (Mårtensson et al., 2020). A popular example of such a scenario is the CAMELEYON17 dataset (Bándi et al., 2019), part of the WILDS (Koh et al., 2021) domain generalization benchmark. Baseline models that attempt to generalize a histology image segmentation task across hospitals saw an average drop in accuracy from 93.2% on training domains to 70.3% on target domains (Koh et al., 2021) due to variations in slide staining and hospital populations.

A solution to this problem is the use of Domain Generalization (DG) methods (Pan and Yang, 2010; Zhou et al., 2021) which identify models that are robust to such shifts in domains by learning domain invariant representations or predictors. Although previous DG research mostly focused on (medical) image classification scenarios, many clinical applications rely on low dimensional tabular data to predict the expected time to a clinical event, using methods from survival analysis. In these scenarios only few highly relevant features are available, and there is considerable error even for a Bayes optimal predictor. This makes clinical time-to-event prediction distinct from the high dimensional scenarios in image classification. In this study we therefore benchmark four DG methods against several Empirical Risk Minimization (ERM) based baseline methods with respect to different types of distribution shifts.

To study the efficacy of domain generalization methods on clinical survival data we ask two questions: 1) how reliable are domain generalization methods in typical clinical survival prediction scenarios and 2) can we provide additional understanding by investigating types of shifts occurring in those scenarios. We use five tabular datasets obtained from randomized controlled trials and from electronic health record derived real-world data. Our main contributions are the following:

- We quantify different types of distribution shifts in the data to characterize scenarios where DG can be successfully applied.

- We find that domain shift and performance degradation of ERM models for survival analysis with tabular data in real-world clinical cases are often smaller than reported in imaging domains (Koh et al., 2021; Zhang et al., 2021).

- We corroborate findings in the imaging literature (Gulrajani and Lopez-Paz, 2020; Koh et al., 2021; Zhang et al., 2021) that DG methods offer an improvement over ERM only in the limited cases in real-world survival analysis scenarios and is correlated with degree of domain shift.

It is important to note that model and hyperparameter selection (Guyon et al., 2010) in DG scenarios is an active field of research and existing strategies often fail to provide a satisfying solution (Gulrajani and Lopez-Paz, 2020). This has drastic empirical consequences, since practitioners, lacking an estimate of a model's generalization error to the target domain, cannot determine whether ERM or DG methods should be preferred and failures w.r.t. generalization can only be detected at the time of prediction.

2. Related work

The goal of domain generalization is to estimate the functional relationship $f(x)$ between a data set X sampled from an input space X and a corresponding outcome of interest $y \in Y$. We further ask that this estimate $f(x)$ generalizes across changes in $P(X)$, $P(Y)$ and $P(Y|X)$ across a set of source domains used for training and a target domain we aim to conduct inference on. In the case of survival analysis, Y is often the cumulative survival distribution $S_t(x)$ for an observation $x \in X$ at time point t. Each data point is assigned a domain d_i. We will denote with $D_{src,i}$ the set of observations sampled from source domain i and with $D_{tgt,i}$ observations sampled from the target domain i ($i \in 1, ..., k$). Models fitted on source domains can now suffer from various distribution shifts that lead to worsened performance. An additional often encountered problem called single-source DG in (Zhou et al., 2021) is that datasets often lack labeled source domains, which either requires identifying domains (Creager et al., 2021) before applying domain generalization, or the use of methods that do not require information about domains (Wang et al., 2019). In contrast to models on high-dimensional data, models studied in our manuscript might suffer less severely from poor generalization as has been shown e.g. in (Simon-Gabriel et al., 2018) under small transformations (Azulay and Weiss, 2019) or adversarial examples (Szegedy et al., 2013).

2.1. Types of domain shifts

Distribution shifts may occur due to different reasons: Shifts in $P(X)$ might e.g. occur due to population differences between rural and urban hospitals, or shifts in $P(Y|X)$ occurring due to differences between clinical trials and treatment in the real-world. Typically, such shifts in distribution do not occur in isolation but domains exhibit several shifts of differing magnitudes. Depending on the perspective, several types of combined shifts can be identified (Zhang et al., 2015).

- Shift in $P(X)$ with constant $P(Y|X)$. This is often referred to as covariate shift in the literature (Zhang et al., 2015). In this case, model performance should theoretically not degrade, but in practice models might be oversimplified and under-fits the conditional models, which causes the predicted Y to depend on the input distribution $P(X)$.

- Shift in $P(Y|X)$. In this case, the optimal model takes into account variations in $P(Y|X)$ between source domains in order to predict the target domain.

- Shift in $P(Y)$: Since we model a functional relationship X→Y, a shift in Y can not occur in isolation (Zhang et al., 2015). define two types of shifts in Y for the reverse causal direction Y→X that result in subsequent changes of $P(X)$ or $P(Y|X)$. We assume effects only in the temporal direction $X \rightarrow Y$ in the remainder of this manuscript.

2.2. Domain generalization methods

Domain Generalization, in contrast to the related concepts of transfer learning and domain adaptation, does not assume access to or knowledge about statistics of the target domain (Pan and Yang, 2010). In recent years, a large variety of domain generalization methods have been proposed. Methods vary from kernel-based methods (Blanchard et al., 2011; Muandet et al., 2013) to approaches that incorporate causal frameworks such as Invariant Causal Predictions (ICP) (Peters et al., 2016; Rothenhäusler et al., 2021) as well as approaches that take a robustness perspective (Krueger et al., 2021; Sagawa et al., 2019). For brevity we only introduce methods relevant to our benchmark, a comprehensive overview over state-of-the-art DG methods is e.g. provided in (Zhou et al., 2021).

3. Method

3.1. ERM and domain generalization methods for survival analysis

3.1.1. Baselines

We investigate two methods based on empirical risk minimization (ERM) as baselines. We choose two widely used models, a Cox Proportional-Hazards model (coxph)(Cox, 1972) and a parametric model using a weibull distribution (weibull) (Kalbfleisch and Prentice, 2011). In coxph, the hazard function $h_i(t) = h_0(t)\, exp(\sum_{k=1}^{p} \theta_k x_{i,k})$, where each feature affects the hazard multiplicatively. In the Weibull model, baseline hazards are defined as $h_i(t) = \lambda \gamma t^{\gamma-1}$ with estimated shape γ and scale λ, as a linear combination of the features X.

3.1.2. Ensemble-based approaches

Ensembles of ML models can be used to obtain better generalizing estimators (Zhou et al., 2021). We investigate survival forests (Ishwaran et al., 2008) as well as more sophisticated survival quilts (Lee et al., 2019) as ensemble baselines for the survival context. Temporal quilting constructs ensembles of survival models assuring that the resulting model is a valid risk function. The core idea is to optimize weights $w_{j,t}$ for risk functions of individual ensemble members j and each time point t optimizing model calibration under a constraint for the predictive error using Bayesian Optimization.

3.1.3. Low-rank decomposition based approaches

Several low-rank decomposition based approaches have been proposed in literature (Khosla et al., 2018; Li et al., 2017; Piratla et al., 2020). We design a strategy heavily inspired by common-specific low-rank decomposition (LRD, (Piratla et al., 2020)). The core assumption is, that for each source domain k, the optimal model's parameters can be written as

$$\theta^{\star} = \theta_c + \gamma_k \theta_k$$

where θ_k is a domain specific effect and the goal therefore is to find model coefficients θ_c encompassing the signal that is common across all domains. We perform a low-rank decomposition on the model coefficients of a cox proportional hazards model fitted on each domain in order to find a set of coefficients θ_c containing the domain-independent signal which is used for subsequent prediction on the target domains.

3.1.4. Invariant Risk Minimization (IRM) & Environment Inference for Invariant Learning (EIIL)

Arjovsky et al. (2019) propose Invariant Risk Minimization (IRM), a novel risk minimization strategy with the goal to discover domain-invariant classifiers Φ by solving the following minimization problem:

$$\underset{\Phi}{min} \sum_i R^i(\Phi) + \lambda * \|\nabla_{w|w=1} R^i(w \cdot \Phi)\|^2$$

The resulting invariant predictor therefore is balanced by λ between predictive performance and a low gradient at w=1 as a measure of domain invariance (Arjovsky et al., 2019). If domain assignments d_i are latent, domains can be inferred as described in (Creager et al., 2021) (EIIL) by inferring domain assignments d_i, such that the domain invariance in the equation is maximized before training the model using IRM. This method will fit for the single-source DG use cases. We adapt IRM/EIIL a survival by optimizing for the negative log likelihood of the Cox PH risk as R^i (c.f. Kvamme and Borgan (2021)).

3.1.5. Continuously Index Domain Adaptation (CIDA)

Wang et al. (2020) propose an Encoder-Decoder based approach to obtain domain invariant representations E(x) for the scenarios where domain assignments i are continuous (Wang et al., 2020). The goal is to learn an encoder E that allows for training a predictor F on y which simultaneously does not permit predicting domain assignments i.

$$\underset{E,F}{min}\ \underset{D}{max}\ L_p(F(E(x,d_i)),y) - \lambda_d L_d(D(E(x,d_i)),d_i)$$

Models can be trained using either the $L_d = L_2$ (CIDA), i.e. the mean squared error loss or a probabilistic loss L_d modeling the mean and variance of a Gaussian distribution and optimizing for the negative log-likelihood (PCIDA). We configured the model both in a linear as well as a deep setting, implementation details can be found in the appendix. We adapt to a survival setting by employing the negative log likelihood of the Cox PH model as a loss for L_p (Kvamme and Borgan, 2021).

3.2. Quantifying shifts

We try to characterize types and magnitudes of shifts occurring in the data to better understand differences between the scenarios and investigate correlations between domain shifts and the improvement by DG methods. In particular, we measure the shift between the target domain and the pooled source domains. We propose 3 metrics allowing for measuring the different shifts, i.e., shifts in $P(X)$, $P(Y)$, and $P(Y|X)$ aforementioned. Other metrics for such shifts have been proposed in histopathology (Stacke et al., 2020), or structured biological data (Borgwardt et al., 2006).

3.2.1. Shift in $P(X)$

Shift in $P(X)$ are summarized using the Wasserstein distance (Dobrushin) between the distribution of the propensity score of data in the source and target domain. The propensity score of a sample in source or target domain was calculated using a logistic model with all features in X.

3.2.2. Shift in $P(Y)$

The distribution shift in $P(Y)$ were measured using the chi square statistics from the log-rank test between the outcomes in the source and target domain.

$$\tilde{\chi_i}^2 = \sum_{t=1}^{n} \frac{(O_{it} - E_{it})^2}{Var(E_{it})}$$

$$Var(E_{it}) = E_{ij}(\frac{N_t - O_t}{N_t})(\frac{N_t - N_{it}}{N_t - 1})$$

where O_{it} represents the observed number of events in the group i (target or source domain) over time, E_{it} represents the expected number of events in the group i over time, N_{it} represents the number of subject at time t in group i.

3.2.3. Shift in $P(Y|X)$

To measure the shift in $P(Y|X)$, we used the difference between a model trained on the source domain (Φ^{src}) and a model trained on the target domain (Φ^*, fitted with the training split of the target domain) in the Akaike information criterion (AIC) computed on the data from target domain for both models. Note that the Φ^* is fitted in the target domain, which usually have a much smaller sample size than the source domain in our experiments.

$$D_{x,y} = AIC(Y_{tgt}, \Phi^*(X_{tgt})) - AIC(Y_{tgt}, \Phi^{src}(X_{tgt}))$$

3.3. Model selection

Since DG assumes no access to target domain data, no reliable estimates for the generalization error GE_{tgt} are available for model selection or hyperparameter tuning (Sagawa et al., 2019; Gulrajani and Lopez-Paz, 2020). Since we are interested in the performance of models, we investigate models from a *post-hoc* perspective by reporting *best-in-class* performance on target domain data from each approach (Oracle, ERM, DG). Since this is not possible in practice, we additionally investigate a setting where a model is selected based on performance on a 30% validation sample collected from each source domain. Then, the best model is refitted on the full data to compute GE_{tgt} after model selection. We investigate differences to post-hoc selection in order to assess the effect of model selection.

3.4. Synthetic domain shift

Besides scenario A, all other scenarios has natural domains that can be identified. For scenario A, we create a label based on the quantiles (0-20%, 21-40%, 41-60%, 61-80%, 81-100%) of propensity score of one sample, which primarily categorize data based on the distribution of P(X). We use the propensity labels together with the original domain labels to rearrange the source and target domains to 10 different subdomains. By different combinations of these subdomains, we are able to create new source and target datasets with different degrees of distribution shifts and to test the performance of domain generalization under these scenarios (see Appendix for more details).

3.5. Data

In this study, we focus on patient-level tabular data from the oncology domain, in particular diffuse large B-cell lymphoma (DLBCL) as well as squamous and non-squamous small cell lung cancer (NSCLC). Datasets are obtained either from randomized controlled trials (RCTs) or electronic health record based real-world data (RWD) collection efforts (for more details see Appendix). Measured covariates contain demographic information such as sex and age as well as clinical variables, e.g., Eastern Cooperative Oncology Group (ECOG) score (Oken et al., 1982) and lab test results. For NSCLC datasets (dataset D,E), we select five covariates following Alexander et al. (2017), while for DLBCL (dataset A,C,E) we select five variables chosen for the International Prognostic Index (IPI) (International Non-Hodgkin's Lymphoma Prognostic Factors Project, 1993). For both diseases, patients' death of all causes is used as the target event for survival analysis. We further created synthetic datasets based on the DLBCL data in case A, using the method described above.

The five different domain shift cases from real clinical data are summarized in Table 1. In order to provide an overview, we indicate the presence of detected shifts. The datasets contain between 733 and 3218 samples distributed into 5-8 source domains while target domains are pooled into a single domain for evaluation. Target domain sizes vary from 107 to 733 (Table 1). Distributions across datasets reflect shifts that might typically be encountered in clinical practice: A) build a prognostic model for DLBCL on RCT data and apply the model in the real-world dataset; B) train a model on squamous NSCLC trials and apply it to a non-squamous NSCLC trial, which may reflect the case of generalization of models between disease subtypes; C) train a model on a set of randomly selected treatment groups in the NSCLC trials and apply to other treatment groups in NSCLC trials, which aims to investigate the potential influence by the change of care; D) train a model based on younger DLBCL patient groups (0-60 yrs) and apply the model to the older population (60+), which tests the generalizability between demographic subpopulations; E) train a model on low/intermediate risk populations and apply to the high risk population of DLBCL patients, which tests generalizability across risk groups (Wang et al., 2021).

4. Domain Generalization on clinical data

We design an experiment comparing a set of DG methods reflective of existing approaches to ERM baselines including cox proportional-hazards models (Cox, 1972) and weibull models (Kalbfleisch and Prentice, 2011). Together with ensemble models presented above, this allows us to go one step further beyond questions posed in previous publications (Gulrajani and Lopez-Paz, 2020; Zhang et al., 2021): analyz-

Table 1: **Overview of datasets and generalization cases.** Criteria for check marks: if the Wasserstein distance of features between the datasets is larger than 0.1, we consider it as a case with shift in P(X); if the differences on survival outcomes between the source and target domains is significant under a log-rank test ($p-value < 0.05$), we consider it as a case with shift in P(Y); if the difference on δAIC is larger than 6, we consider it as a case with shift in $P(Y|X)$.

ID	Generalization	$P(X)$	$P(Y)$	$P(Y\|X)$	Samples size (target domain)	Sample size (source domain)	Number of source domains
A	RCT to RWD	✓	✓	✓	733	1060	1
B	Cancer subtypes	-	✓	✓	107	3111	6
C	Treatment groups	-	✓	✓	254	2857	8
D	Age group	-	-	-	191	542	4
E	Disease risk	-	✓	✓	151	582	4

ing whether DG not only improves over simple baselines but instead also over more advanced methods that are readily available in practice. We adapt the following experimental protocol for all studied settings: After splitting data into fixed source and target domains, we impute missing values using their median/mode and subsequently fit the model. Each experiment is replicated 10 times on a randomly drawn 90% sample of the data to obtain an estimate of the variance of results.

4.1. Performance metrics

Given that our outcome of interest is a continuous survival distribution, we measure the target domain generalization error (GE_{tgt}) in terms of the C-index (Harrell Jr et al., 1996) on data obtained from the target domain.

$$GE_{tgt} = C(Y_{tgt}, \Phi(X_{tgt}))$$

We further estimate the source domain generalization error GE_{src} using a random 70-30 split of source data from each source domain as training and validation data. The GE_{src} are used for the model selection process as the validation data strategy.

$$GE_{src} = C(Y^{'}_{src}, \Phi(X^{'}_{src}))$$

4.2. Oracle models

We further study an oracle condition to answer how much performance gain could be had, if access to the target domain data was available. Measured differences in C-index between source model and the oracle model stem from training on either a subset of the target domain (oracle model) or source domain data (source model). The resulting gap δ_{oracle} thus reflects differences in training data between source and target domain.

5. Results

5.1. Domain shifts and model performance degradation

In case A, we observe a significant difference in $P(X)$ (Wasserstein distance $W_p = 0.25$), $P(Y)$ ($\chi^2 = 23.49$, $p < 0.001$) and $P(Y|X)$ ($\Delta AIC = 12.62$) between the cohorts from RWD and the RCT. Case B and C both show distribution shifts in $P(Y)$ and $P(Y|X)$, but the metrics are larger in case B. In case D, we observe a moderate shift in $P(Y)$ (χ^2 = 5.55, p = 0.02), but not in $P(Y|X)$ or $P(X)$. In case E, we found a significant difference in P(Y) (χ^2 = 8.75, p = 0.003), and $P(Y|X)$ (ΔAIC = 16.33). Metrics of domain shifts are summarised in Table 2 (for additional details see Fig. 3 in the Appendix).

When measuring performance degradation by comparing GE_{src} and GE_{tgt}, differences range between 0.01 and 0.05 across all experiments (Fig. 4 in the Appendix). Interestingly, we find that performance of the ERM models on the source domain are worse than on the target domain in the conducted experiments. When using the oracle models as a comparator, oracle models are also not always superior to the ERM models. The performances of oracle models are worse than ERM models in the cases D by 0.03 and E by 0.01. Note, that model performances can be worse than random guessing if models over-fit on source domain data.

Table 2: **Summary of experimental results.** We compare the best performing baseline with the best performing DG method selected as best-of-class (DG) and selected according to the validation data strategy (DG'). We report the domain shift by each metric for each tested scenario. For the model performance, we report mean C-index along with standard deviations (in brackets) over 10 monte-carlo cross-validation iterations. The best class is denoted in bold.

	A	**B**	**C**	**D**	**E**
Domain shift metrics					
Shift in $P(X)$ (W_p)	0.25	0.02	0.01	0.01	0.03
Shift in $P(Y)$ (χ^2)	23.49	12.43	3.89	5.55	8.75
Shift in $P(Y\|X)$ (ΔAIC)	12.62	19.24	6.32	2.76	16.33
C-index by method class					
Oracle	0.70(0.02)	0.65(0.07)	0.63(0.04)	0.66(0.08)	0.67(0.07)
ERM	0.70(0.01)	0.59(0.01)	0.62(0.01)	**0.71(0.01)**	**0.69(0.01)**
Ensemble	0.65(0.01)	0.61(0.01)	0.640(0.00)	0.64(0.01)	0.66 (0.01)
DG	0.70(0.01)	**0.63(0.01)**	**0.66(0.01)**	0.70(0.01)	0.67(0.01)
DG'	0.65(0.01)	0.60(0.01)	0.66(0.01)	0.66(0.01)	0.66(0.01)

5.2. Performance of DG methods

We report best-of-class results across all investigated DG and ERM methods to provide an overview. We find DG only outperforms ERM in two out of five real-world cases, with improvement on C-index between 0.03 and 0.05. In case A, the difference between the best DG and ERM was almost negligible. In case D and E, ERM gives yields better C-indices than DG (Table 2). Among the selected DG methods, LRD performed competitively well in all cases. CIDALinear was the best performing method for case C, and outperformed ERMs in two out of five cases. Full results can be found in Table 3 in the Appendix.

When applying the *validation data strategy* for model selection, the selected methods (DG', Table 2) have a loss from 0.02 to 0.05 in C-index compared to the best-of-class results.

5.3. Correlation between domain shift and the improvement by DG methods

Since the introduced domain shift metrics such as chi-square statistics and ΔAIC are comparable with the same target domain, we derive an experiment allowing us to evaluate correlations with a fixed testing dataset while changing source domains to mimic scenarios with different degrees of shifts between source and target domain.

We first tested the correlation in case B. While keeping the same testing dataset, we remove clinical trials from the training datasets, two at each time, to create new training datasets. This generates shifts with ΔAIC ranging from 16.6 to 20.5. With increased domain shift, we observe a larger advantage in performance of DG over ERM based models (from 0.63 to 0.59 for DG, 0.63 to 0.56 for ERM, Figure 1).

Using synthetic domain labels, we applied the same approach and created four scenarios with increasing domain shift between the target and source domains based on case A (Fig. 5 in the Appendix). The ERM and DG models have similar performance in the initial scenario (median C-index of 0.70 in both cases with ΔAIC of 14). However, when the ΔAIC increases to 26 and 39, the performance was worse in ERM models (median C-index 0.68 and 0.66) compared to DG models (0.69 and 0.67, Fig. 2).

6. Discussion

6.1. Domain shift and generalizability issues in clinical data

A comparison between GE_{src} and GE_{tgt} is a common method to evaluate model performance under domain shift. In contrast to the literature on image data, where significant performance degradation has been reported in target domains (Arjovsky et al., 2019; Koh et al., 2021), we observed relatively small performance loss in most of the tested cases, which are low dimensional survival analysis settings. In our experiment we find domain shift are not reflected in differences between GE_{src} and GE_{tgt}, which is often

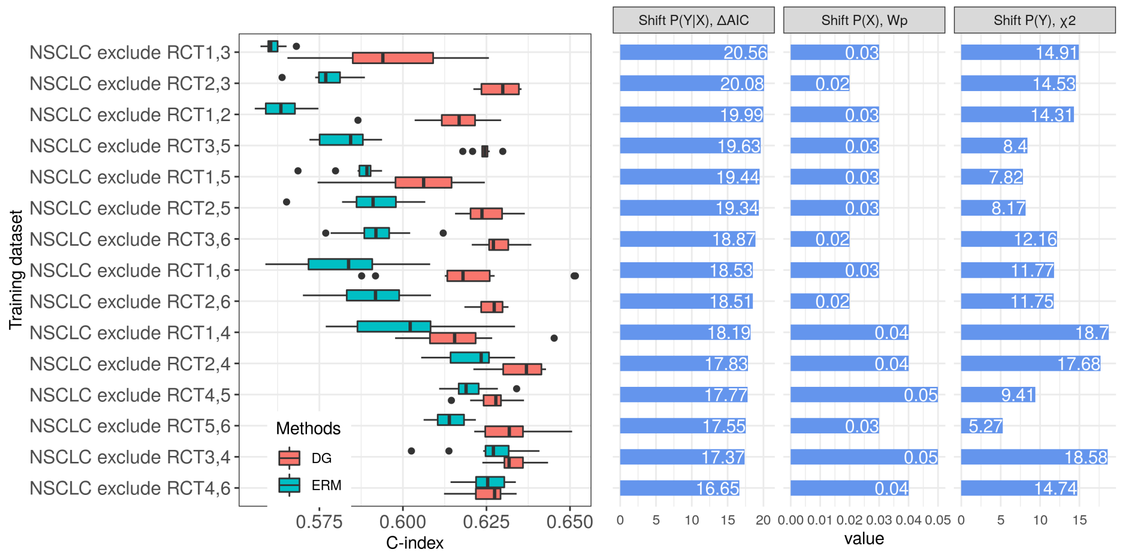


Figure 1: **Performance of ERM and DG models with increased domain shift in case B**. While keeping the same target domain, we create different training datasets by excluding trials from the source domain. See appendix for the details of the NSCLC RCTs.

used as an indicator for detecting potential generalization issues. We observe that models often exhibit $GE_{tgt} < GE_{src}$, which might stem from different Bayes error rates (Fukunaga, 1990) between the domains.

Oracle models are often used in previous studies to benchmark the performance of DG methods. However, oracle models may suffer from small sample sizes and large variations in the target domain. In the current study, oracle models also do not seem to be the upper limit of the performance as suggested in other literature (Zhang et al., 2021). In practice, comparing the ERM model with an oracle model trained on a small dataset from the target domain may not necessarily help to identify potential generalizability issues either. An additional open challenge is model selection from a set of candidate models, which can lead to severe performance degradation in comparison to the *post-hoc* best algorithm.

Because no single metric can directly measure the generalizability of a model, a set of carefully designed experiments are required to understand the underlying issues.

6.2. Synthetic data for DG assessment

Synthetic data allows to test DG methods in controlled experiments. However, previous publications have shown that DG methods only outperform ERMs in some special settings, particularly influenced by spurious correlations generated from the synthetic process. Synthetic DG scenarios such as coloured MNIST (Arjovsky et al., 2019) or artificial features often introduce spurious correlations in the source domains that are reversed on the target domain, which is perhaps rarely observed in real datasets (Zhang et al., 2021; Arjovsky et al., 2019).

Instead of creating synthetic datasets, we introduce a method to create synthetic domain labels based on propensity scores. One advantage of this approach is that the method only attempts to identify subclusters within the sample to be used as domains, and thus does not change the correlations between the features and the outcomes. This may mimic more realistic domain shift scenarios than directly modifying the distribution of the original data.

6.3. Factors influencing domain generalization methods

Previous studies reported that domain generalization provides no advantage in the case of more subtle data

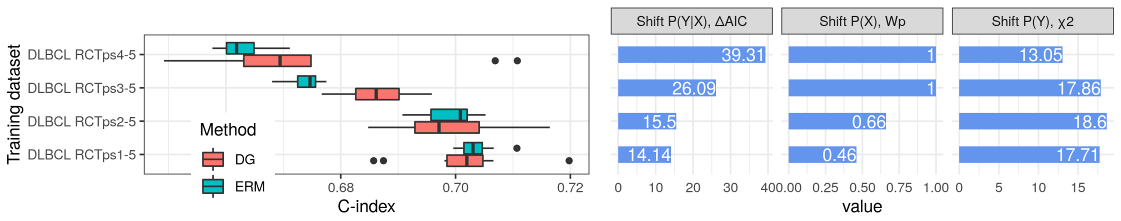


Figure 2: **Performance of ERM and DG models with increased domain shift in the synthetic data**. While keeping the same target domain as in case A, we create multiple training datasets by removing subgroups of patients from the original DLBCL trial data according to their propensity scores categories, which creates training datasets with different degrees of deviation from the target domain. As an example, RCTps2-5 includes patients in RCT with propensity score category 2-5. Models are tested on patients from RWD dataset with propensity score category 1 and 2.

shifts (Zhang et al., 2021; Gulrajani and Lopez-Paz, 2020). We observe that the DG methods show more benefits when the source and target domains have larger deviations. From a practical point of view, when the training datasets are expected to be close to the target domains, training with ERM methods might be sufficient; when the training data are more likely to be different from the target domain, e.g. from a different disease subtype (case B) or patients with different characteristics (the synthetic case), DG method could yield improvements.

It is worth noting that $\Delta AICs$ from different datasets are not directly comparable, comparisons are only meaningful when the models share the same test dataset. Although Case A and E have similar $\Delta AICs$ as in case B, it does not necessarily mean that they have the same degree of domain shift. On the other hand, there might be other factors influencing the efficacy of DG methods, such as the diversity of the source domains (Zhou et al., 2021) and sample size. In case A, since only one dataset is used in the training, the source domains are created by a random split of the dataset, which may result in very homogeneous source domains. Similarly in case E, although the target domain includes patients with higher risk, the source domains are similarly included patients with low/intermediate risk patients. Additionally, case E has a relatively small sample size in each source domain (100), which may influence the model fitting.

The process of model selection is another factor influencing the final performance of the DG methods (Gulrajani and Lopez-Paz, 2020). In our experiments, the selected models have a loss in c-index between 0.02 to 0.05 compared with post-hoc model selection. The model selection should not only include hyper-parameter tuning of the algorithms, but also feature selection. None of the original publications of the tested DG methods address the proper optimization under the DG scenario. As suggested previously (Zhang et al., 2021; Gulrajani and Lopez-Paz, 2020), the model selection strategy needs to be an integral part of a domain generalization method and its evaluation. Without it, the validity of the reported performance of these methods is limited.

6.4. Clinical applications and regulatory hurdles

It is encouraging for practitioners in the field that in most cases ERM methods are performing competitively. But we should also be aware that in many other scenarios DG methods do outperform ERM models. Both successful and failed attempts to use DG methods in different clinical applications abound in the literature (Lafarge et al.; Guo et al., 2021; Jin et al.). As observed here and elsewhere (Wang et al.), these conflicting observations may be explained by different degrees of domain shifts as well as the quantity and diversity of training data. The challenge thus lies in the correct choice of proper methods contingent on the recognition of the specific type of scenario. To aid this choice, we propose a set of metrics that can be used to qualify domain shifts, and to understand the diversity within the source domain. Additionally, estimating potential shifts in the target domain of the intended use cases will be a potential required step for identifying the proper use scenarios (Gossmann et al.).

For a clinical algorithm, the regulatory requirement plays a critical role. With the rising question on trustworthiness of the machine learning models, the request is not only on accuracy, but increasingly on transparency of the model training process, which includes a demonstration of model design tailored to the available data and intended use (FDA, 2021). These may imply a requirement of evidence to justify the use of selected methods, their applicable scenarios and potential risks, such as over-fitting, performance degradation, and security risks. In essence, this calls for comprehensive evaluation of methods before real-world application.

7. Conclusion

In this study, we evaluated four recently published domain generalization methods for their ability to generalize to an unseen data domain with real clinical data. Similar to previous findings with imaging data, these methods provided improvement over ERM for survival analysis in limited settings. However, our study is limited to comparatively low-dimensional settings (less than 10 features), which are often encountered in clinical practice. Richer settings employing a larger number of variables might result in different results due to possibly stronger over-fitting. We propose several metrics of domain shifts, and analyze the factors influencing the efficacy of the DG methods, which is a first step to find the right method that fits a particular domain shift scenario. Furthermore, data used throughout our study comes only from the US, a broader study across different populations could lead to interesting results.

Most of the current DG methods were developed for tasks outside low-dimensional clinical settings and may therefore not have been optimized for clinical use cases. We hope our work will encourage researchers in the field to further develop suitable DG methods for clinical research, as well as to develop more fitting evaluation frameworks and datasets to benchmark these methods.

Institutional Review Board (IRB)

The data used in this study were all published previously, the study did not require an IRB approval. For the original clinical studies, approval from the Independent Review Board (IRB)/Independent Ethics Committee (IEC) were obtained before the start of the studies, and all patients provided written informed consent.

Acknowledgments

The project is funded by F.Hoffmann-La Roche AG. The work was done while FP was interning at Fa. Hoffman-La Roche AG. We would like to thank Prof. Bernd Bischl for supporting FP taking the intern project and continuing with the write up after the internship.

References

Marliese Alexander, Rory Wolfe, David Ball, Matthew Conron, Robert G Stirling, Benjamin Solomon, Michael MacManus, Ann Officer, Sameer Karnam, Kate Burbury, et al. Lung cancer prognostic index: a risk score to predict overall survival after the diagnosis of non-small-cell lung cancer. *British journal of cancer*, 117(5):744–751, 2017.

Martin Arjovsky, Léon Bottou, Ishaan Gulrajani, and David Lopez-Paz. Invariant risk minimization. *arXiv preprint arXiv:1907.02893*, 2019.

Aharon Azulay and Yair Weiss. Why do deep convolutional networks generalize so poorly to small image transformations? *Journal of Machine Learning Research*, 20:1–25, 2019.

Gilles Blanchard, Gyemin Lee, and Clayton Scott. Generalizing from several related classification tasks to a new unlabeled sample. *Advances in neural information processing systems*, 24:2178–2186, 2011.

Karsten M Borgwardt, Arthur Gretton, Malte J Rasch, Hans-Peter Kriegel, Bernhard Schölkopf, and Alex J Smola. Integrating structured biological data by kernel maximum mean discrepancy. *Bioinformatics*, 22(14):e49–e57, 2006.

Péter Bándi, Oscar Geessink, Quirine Manson, Marcory Van Dijk, Maschenka Balkenhol, Meyke Hermsen, Babak Ehteshami Bejnordi, Byungjae Lee, Kyunghyun Paeng, Aoxiao Zhong, Quanzheng Li, Farhad Ghazvinian Zanjani, Svitlana Zinger, Keisuke Fukuta, Daisuke Komura, Vlado Ovtcharov, Shenghua Cheng, Shaoqun Zeng, Jeppe Thagaard, Anders B. Dahl, Huangjing Lin, Hao Chen, Ludwig Jacobsson, Martin Hedlund, Melih Çetin, Eren Halıcı, Hunter Jackson, Richard Chen, Fabian Both, Jörg Franke, Heidi Küsters-Vandevelde, Willem Vreuls, Peter Bult, Bram van Ginneken, Jeroen van der Laak, and Geert Litjens. From detection of individual metastases to classification of lymph node status at the patient level: The camelyon17 challenge. *IEEE Transactions on Medical Imaging*, 38(2):550–560, 2019. doi: 10.1109/TMI.2018.2867350.

Robert Challen, Joshua Denny, Martin Pitt, Luke Gompels, Tom Edwards, and Krasimira Tsaneva-Atanasova. Artificial intelligence, bias and clinical safety. *BMJ Quality & Safety*, 28(3):231–237, 2019.

Joseph Paul Cohen, Mohammad Hashir, Rupert Brooks, and Hadrien Bertrand. On the limits of cross-domain generalization in automated x-ray prediction. In *Medical Imaging with Deep Learning*, pages 136–155. PMLR, 2020.

David R Cox. Regression models and life-tables. *Journal of the Royal Statistical Society: Series B (Methodological)*, 34(2):187–202, 1972.

Elliot Creager, Joern-Henrik Jacobsen, and Richard Zemel. Environment inference for invariant learning. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 2189–2200. PMLR, 18–24 Jul 2021.

R. L. Dobrushin. Prescribing a system of random variables by conditional distributions. 15(3):458–486. ISSN 0040-585X. doi: 10.1137/1115049. Publisher: Society for Industrial and Applied Mathematics.

Health Center for Devices and Radiological FDA. Artificial Intelligence and Machine Learning in Software as a Medical Device. *FDA*, September 2021. Publisher: FDA.

Keinosuke Fukunaga. *Introduction to Statistical Pattern Recognition (Computer Science & Scientific Computing)*. Academic Press, hardcover edition, 10 1990. ISBN 978-0122698514.

Alexej Gossmann, Kenny H. Cha, and Xudong Sun. Performance deterioration of deep neural networks for lesion classification in mammography due to distribution shift: an analysis based on artificially created distribution shift. In *Medical Imaging 2020: Computer-Aided Diagnosis*, volume 11314, page 1131404. International Society for Optics and Photonics.

Ishaan Gulrajani and David Lopez-Paz. In search of lost domain generalization. *CoRR*, abs/2007.01434, 2020.

Lin Lawrence Guo, Stephen R Pfohl, Jason Fries, Alistair Johnson, Jose Posada, Catherine Aftandilian, Nigam Shah, and Lillian Sung. Evaluation of domain generalization and adaptation on improving model robustness to temporal dataset shift in clinical medicine. *medRxiv*, 2021. doi: 10.1101/2021.06.17.21259092.

Isabelle Guyon, Amir Saffari, Gideon Dror, and Gavin Cawley. Model selection: Beyond the bayesian/frequentist divide. *Journal of Machine Learning Research*, 11(3):61–87, 2010.

Frank E Harrell Jr, Kerry L Lee, and Daniel B Mark. Multivariable prognostic models: issues in developing models, evaluating assumptions and adequacy, and measuring and reducing errors. *Statistics in medicine*, 15(4):361–387, 1996.

International Non-Hodgkin's Lymphoma Prognostic Factors Project. A predictive model for aggressive non-hodgkin's lymphoma. *N Engl J Med*, 329(14): 987–994, 1993.

Hemant Ishwaran, Udaya B. Kogalur, Eugene H. Blackstone, and Michael S. Lauer. Random survival forests. *The Annals of Applied Statistics*, 2(3): 841–860, September 2008. ISSN 1932-6157, 1941-7330. doi: 10.1214/08-AOAS169. Publisher: Institute of Mathematical Statistics.

Xin Jin, Cuiling Lan, Wenjun Zeng, Zhibo Chen, and Li Zhang. Style normalization and restitution for generalizable person re-identification. URL http://arxiv.org/abs/2005.11037.

John D Kalbfleisch and Ross L Prentice. *The statistical analysis of failure time data*, volume 360. John Wiley & Sons, 2011.

Aditya Khosla, Tinghui Zhou, Tomasz Malisiewicz, Alexei Efros, and Antonio Torralba. Undoing the damage of dataset bias, Jun 2018.

Pang Wei Koh, Shiori Sagawa, Sang Michael Xie, Marvin Zhang, Akshay Balsubramani, Weihua Hu, Michihiro Yasunaga, Richard Lanas Phillips, Irena

Gao, Tony Lee, et al. Wilds: A benchmark of in-the-wild distribution shifts. In *International Conference on Machine Learning*, pages 5637–5664. PMLR, 2021.

David Krueger, Ethan Caballero, Joern-Henrik Jacobsen, Amy Zhang, Jonathan Binas, Dinghuai Zhang, Remi Le Priol, and Aaron Courville. Out-of-distribution generalization via risk extrapolation (rex). In *International Conference on Machine Learning*, pages 5815–5826. PMLR, 2021.

Håvard Kvamme and Ørnulf Borgan. Continuous and discrete-time survival prediction with neural networks. *Lifetime Data Analysis*, pages 1–27, 2021.

Maxime W. Lafarge, Josien P. W. Pluim, Koen A. J. Eppenhof, and Mitko Veta. Learning domain-invariant representations of histological images. 6: 162. ISSN 2296-858X. doi: 10.3389/fmed.2019.00162.

Changhee Lee, William Zame, Ahmed Alaa, and Mihaela Schaar. Temporal quilting for survival analysis. In *The 22nd international conference on artificial intelligence and statistics*, pages 596–605. PMLR, 2019.

Da Li, Yongxin Yang, Yi-Zhe Song, and Timothy M Hospedales. Deeper, broader and artier domain generalization. In *Proceedings of the IEEE international conference on computer vision*, pages 5542–5550, 2017.

Gustav Mårtensson, Daniel Ferreira, Tobias Granberg, Lena Cavallin, Ketil Oppedal, Alessandro Padovani, Irena Rektorova, Laura Bonanni, Matteo Pardini, Milica G Kramberger, et al. The reliability of a deep learning model in clinical out-of-distribution mri data: a multicohort study. *Medical Image Analysis*, 66:101714, 2020.

Krikamol Muandet, David Balduzzi, and Bernhard Schölkopf. Domain generalization via invariant feature representation. In *International Conference on Machine Learning*, pages 10–18. PMLR, 2013.

Martin M Oken, Richard H Creech, Douglass C Tormey, John Horton, Thomas E Davis, Eleanor T McFadden, and Paul P Carbone. Toxicity and response criteria of the eastern cooperative oncology group. *American journal of clinical oncology*, 5(6): 649–656, 1982.

Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010. doi: 10.1109/TKDE.2009.191.

Jonas Peters, Peter Bühlmann, and Nicolai Meinshausen. Causal inference by using invariant prediction: identification and confidence intervals. *Journal of the Royal Statistical Society. Series B (Statistical Methodology)*, pages 947–1012, 2016.

Vihari Piratla, Praneeth Netrapalli, and Sunita Sarawagi. Efficient domain generalization via common-specific low-rank decomposition. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 7728–7738. PMLR, 13–18 Jul 2020.

Eduardo HP Pooch, Pedro L Ballester, and Rodrigo C Barros. Can we trust deep learning models diagnosis? the impact of domain shift in chest radiograph classification. *arXiv preprint arXiv:1909.01940*, 2019.

Joaquin Quiñonero-Candela, Masashi Sugiyama, Neil D Lawrence, and Anton Schwaighofer. *Dataset shift in machine learning.* Mit Press, 2009.

Dominik Rothenhäusler, Nicolai Meinshausen, Peter Bühlmann, and Jonas Peters. Anchor regression: Heterogeneous data meet causality. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 83(2):215–246, 2021.

Shiori Sagawa, Pang Wei Koh, Tatsunori B Hashimoto, and Percy Liang. Distributionally robust neural networks for group shifts: On the importance of regularization for worst-case generalization. *arXiv preprint arXiv:1911.08731*, 2019.

Carl-Johann Simon-Gabriel, Yann Ollivier, Léon Bottou, Bernhard Schölkopf, and David Lopez-Paz. Adversarial vulnerability of neural networks increases with input dimension. 2018.

Raphael Sonabend, Franz J Király, Andreas Bender, Bernd Bischl, and Michel Lang. mlr3proba: An r package for machine learning in survival analysis. *Bioinformatics*, 02 2021. ISSN 1367-4803. doi: 10.1093/bioinformatics/btab039.

Karin Stacke, Gabriel Eilertsen, Jonas Unger, and Claes Lundström. Measuring domain shift for deep learning in histopathology. *IEEE journal of biomedical and health informatics*, 25(2):325–336, 2020.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

Hao Wang, Hao He, and Dina Katabi. Continuously indexed domain adaptation. *arXiv preprint arXiv:2007.01807*, 2020.

Haohan Wang, Zexue He, Zachary C Lipton, and Eric P Xing. Learning robust representations by projecting superficial statistics out. *arXiv preprint arXiv:1903.06256*, 2019.

Jindong Wang, Cuiling Lan, Chang Liu, Yidong Ouyang, and Tao Qin. Generalizing to unseen domains: A survey on domain generalization. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence*, pages 4627–4635. International Joint Conferences on Artificial Intelligence Organization. ISBN 978-0-9992411-9-6. doi: 10.24963/ijcai.2021/628.

Xuejian Wang, Wenbin Zhang, Aishwarya Jadhav, and Jeremy Weiss. Harmonic-mean cox models: A ruler for equal attention to risk. In Russell Greiner, Neeraj Kumar, Thomas Alexander Gerds, and Mihaela van der Schaar, editors, *Proceedings of AAAI Spring Symposium on Survival Prediction - Algorithms, Challenges, and Applications 2021*, volume 146 of *Proceedings of Machine Learning Research*, pages 171–183. PMLR, 22–24 Mar 2021. URL https://proceedings.mlr.press/v146/wang21a.html.

Gerhard Widmer and Miroslav Kubat. Learning in the presence of concept drift and hidden contexts. *Machine learning*, 23(1):69–101, 1996.

John R Zech, Marcus A Badgeley, Manway Liu, Anthony B Costa, Joseph J Titano, and Eric Karl Oermann. Variable generalization performance of a deep learning model to detect pneumonia in chest radiographs: a cross-sectional study. *PLoS medicine*, 15(11):e1002683, 2018.

Haoran Zhang, Natalie Dullerud, Laleh Seyyed-Kalantari, Quaid Morris, Shalmali Joshi, and Marzyeh Ghassemi. *An Empirical Framework for Domain Generalization in Clinical Settings*, page 279–290. Association for Computing Machinery, New York, NY, USA, 2021. ISBN 9781450383592.

Kun Zhang, Mingming Gong, and Bernhard Schölkopf. Multi-source domain adaptation: A causal view. In *Twenty-ninth AAAI conference on artificial intelligence*, 2015.

Kaiyang Zhou, Ziwei Liu, Yu Qiao, Tao Xiang, and Chen Change Loy. Domain generalization: A survey, 2021.

Appendix A. Data and Methods

A.1. Datasets

The data are collected from the following studies:
NSCLC:

- NCT01351415 (RCT1): A Study of Bevacizumab in Combination With Standard of Care Treatment in Participants With Advanced Non-squamous Non-small Cell Lung Cancer (NSCLC).
- NCT01496742 (RCT2): A Study of Onartuzumab (MetMAb) in Combination With Bevacizumab (Avastin) Plus Platinum And Paclitaxel or With Pemetrexed Plus Platinum in Patients With Non-Squamous Non-Small Cell Lung Cancer.
- NCT01903993 (RCT3): A Randomized Phase 2 Study of Atezolizumab (an Engineered Anti-PDL1 Antibody) Compared With Docetaxel in Participants With Locally Advanced or Metastatic Non-Small Cell Lung Cancer Who Have Failed Platinum Therapy - “POPLAR”.
- NCT02008227 (RCT4): A Study of Atezolizumab Compared With Docetaxel in Participants With Locally Advanced or Metastatic Non-Small Cell Lung Cancer Who Have Failed Platinum-Containing Therapy (OAK).
- NCT02366143 (RCT5): A Study of Atezolizumab in Combination With Carboplatin Plus (+) Paclitaxel With or Without Bevacizumab Compared With Carboplatin+Paclitaxel+Bevacizumab in Participants With Stage IV Non-Squamous Non-Small Cell Lung Cancer (NSCLC) (IMpower150).
- NCT02657434 (RCT6): A Study of Atezolizumab in Combination With Carboplatin or Cisplatin + Pemetrexed Compared With Carboplatin or Cisplatin + Pemetrexed in Participants Who Are Chemotherapy-Naive and Have Stage IV Non-Squamous Non-Small Cell Lung Cancer (NSCLC) (IMpower 132).
- NCT01519804 (target): A Study of Onartuzumab (MetMAb) Versus Placebo in Combination With Paclitaxel Plus Platinum in Patients With Squamous Non-Small Cell Lung Cancer.

DLBCL:

- NCT01287741 (RCT): A Study of Obinutuzumab in Combination With CHOP Chemotherapy Versus Rituximab With CHOP in Participants With CD20-Positive Diffuse Large B-Cell Lymphoma (GOYA)
- FlatironHealth (RWD): This study used the nationwide Flatiron Health electronic health record (EHR)-derived de-identified database. We selected a subset of patients from the DLBCL cohort for the analyses.

A.2. Model Selection

Throughout the manuscript, we report the *best-in-class* model. That is, for each group of models (ERM, DG, Oracle condition), we report the best model based on the average performance on the held-out 30% validation data from each split.
Since DG assumes no access to source domain data, no reliable estimates for the generalization error GE are available. Mainly three strategies have been proposed in literature (Sagawa et al., 2019; Gulrajani and Lopez-Paz, 2020):

- Validation Data Measure generalization error using average performance on a hold-out sample from each source domain.
- Worst-case analysis Measure generalization error as a method’s performance on the worst domain (Sagawa et al. 2019).
- Validation Domain Measure generalization error as a method’s performance on a (randomly) held out source domain.

While each method can help to obtain better estimates of eventual performance on a target domain, their efficacy heavily depends on the (dis-)similarity between source and target domains.

A.3. Implementation Details

IRM & EIL

Since we consider low-dimensional datasets with only few observations, we consider simplistic (linear) neural networks in our benchmarks. The regularization parameter λ for both is tuned on a grid of values: $, 1e-7, 1e-5, 1e-3, 1e-2, 1e-1, .5$ and trained using the Adam optimizer with a learning rate of 0.01.

CIDA

We consider versions of CIDA and PCIDA that include only a linear predictor (CIDALinear and PCIDALinear) as well as a deep version including 4 layers of widths $(8, 12, 12, 8)$ respectively (CIDA, PCIDA). Since we study settings with ~ 5 covariates, we consider a width of 12 to be appropriately big for our neural networks.

A.4. Building synthetic domain shift

We created the synthetic dataset based on case A to mimic scenario different degree of domain shift. The case A contains only one dataset in the source domain. For the main task, we simply separated the data according to the age groups of the population (0-50, 51-60, 61-70, 71-80, 80+), however, this is an over simplification of categorizing the heterogeneous subgroups within the population.

We applied the method described to create synthetic domain labels. Firstly, a logistic regression model was fitted on the combined source and target domain to calculate the propensity of each sample being in one of the domains. Based on the propensity scores, we stratified the whole population according to the quantiles of the propensity scores (0-20%, 21-40%, 41-60%, 61-80%, 81-100%), which are used as the propensity labels (ps_1-5, Fig. 5). We then combined the propensity labels and the original domain labels (RCT or RWD) to create a new domain label for each stratum in the population (e.g. RCT1, indicates the patient comes from the RCT data and belongs to the propensity category ps_1). Based on different combinations of stratum we are able to created scenarios with different degrees of domains shifts.

Appendix B. Figures and Tables

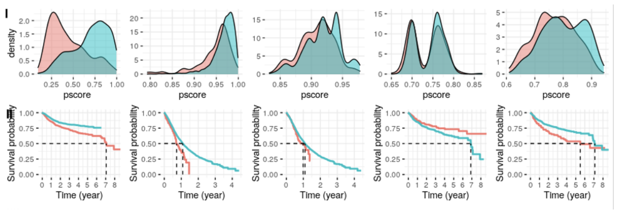


Figure 3: **Summary of the distribution of feature space X and outcome Y.** I) propensity score of data from the source and target domain; II) Kaplan-Meier curve of the source and target domain.

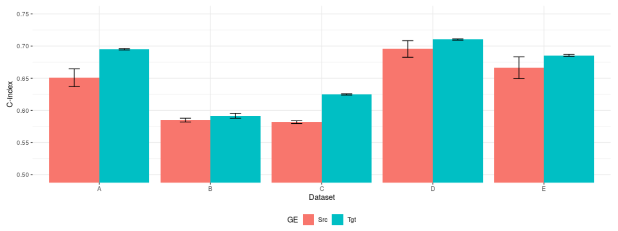


Figure 4: GE_{src} and GE_{tgt} of the ERM Model with error bars for the reported mean.

Table 3: **Full experimental results.** We report c-index across replications for all experiments by method and scenarios. Bold: best method according to GE_{tgt} except Oracle model, underlined: chosen via validation data strategy. We report averages along with standard deviations (in brackets).

Algorithm	**A**	**B**	**C**	**D**	**E**
Oracle	0.703 (0.023)	0.648 (0.069)	0.634 (0.036)	0.664 (0.078)	0.666 (0.069)
ERM					
coxph	**0.695 (0.003)**	0.592 (0.012)	0.625 (0.003)	**0.710 (0.003)**	0.686 (0.005)
weibull	0.694 (0.003)	0.588 (0.011)	0.624 (0.003)	0.710 (0.003)	0.687 (0.004)
LRD	0.683 (0.027)	**0.630 (0.002)**	0.629 (0.004)	0.698 (0.014)	0.654 (0.009)
CIDA					
CIDA	0.631 (0.062)	**0.588 (0.040)**	**0.601 (0.024)**	0.665 (0.044)	0.642 (0.043)
PCIDA	0.593 (0.067)	0.551 (0.054)	0.556 (0.051)	0.654 (0.036)	0.639 (0.049)
CIDALinear	0.648 (0.002)	0.601 (0.014)	**0.658 (0.014)**	0.659 (0.012)	0.659 (0.013)
PCIDALinear	0.646 (0.001)	0.595 (0.015)	0.607 (0.015)	0.683 (0.020)	0.674 (0.014)
IR					
IRM	0.345 (0.034)	0.465 (0.016)	0.478 (0.035)	0.605 (0.055)	0.518 (0.074)
EIIL	0.364 (0.019)	0.452 (0.002)	0.471 (0.002)	0.519 (0.110)	0.292 (0.007)
Ensemble					
surv.quilts	0.635 (0.049)	0.613 (0.006)	0.623 (0.002)	0.635 (0.007)	0.662 (0.009)
surv.forest	0.650 (0.004)	0.602 (0.008)	0.639 (0.003)	0.610 (0.004)	0.655 (0.008)

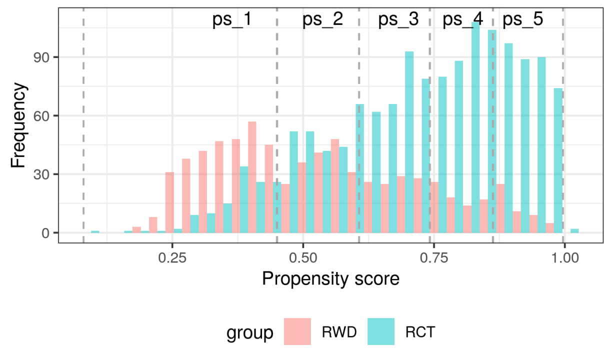


Figure 5: **Create synthetic domain labels based on propensity score in case A**. The RCT and RWD data were each categorized based on the quantiles of the propensity score (ps_1-5). Combining with the original domain labels, the target and source domains were divided into 10 different subdomains, labeled as RCT1-5 and RWD1-5.

6.4 mlr3pipelines - Flexible Machine Learning Pipelines in R

Contributed Article:
M. Binder, F. Pfisterer, M. Lang, L. Schneider, L. Kotthoff, and B. Bischl. mlr3pipelines - flexible machine learning pipelines in R. *Journal of Machine Learning Research*, 22(184):1–7, 2021

Declaration of contributions The project was originally initiated by BB and FP based on previous work by MB in [24]. BB and FP developed initial code that was later revised and rewritten mostly by MB to arrive at the current software. MB also developed the core functionality of the package along with documentation. FP, LS and ML contributed several operators and extensions to the package and improved documentation as well as core functionality. ML, LK and BB advised throughout the process and improved the resulting manuscript.

Journal of Machine Learning Research 22 (2021) 1-7 Submitted 3/21; Revised 6/21; Published 6/21

mlr3pipelines – Flexible Machine Learning Pipelines in R

Martin Binder[1] MARTIN.BINDER@STAT.UNI-MUENCHEN.DE
Florian Pfisterer[1] FLORIAN.PFISTERER@STAT.UNI-MUENCHEN.DE
Michel Lang[1] MICHEL.LANG@STAT.UNI-MUENCHEN.DE
Lennart Schneider[1] LENNART.SCHNEIDER@STAT.UNI-MUENCHEN.DE
Lars Kotthoff[2] LARSKO@UWYO.EDU
Bernd Bischl[1] BERND.BISCHL@STAT.UNI-MUENCHEN.DE

[1] *Department of Statistics, LMU Munich, Germany*
[2] *Department of Computer Science, University of Wyoming, USA*

Editor: Alexandre Gramfort

Abstract

Recent years have seen a proliferation of ML frameworks. Such systems make ML accessible to non-experts, especially when combined with powerful parameter tuning and AutoML techniques. Modern, applied ML extends beyond direct learning on clean data, however, and needs an expressive language for the construction of complex ML workflows beyond simple pre- and post-processing. We present `mlr3pipelines`, an R framework which can be used to define linear and complex non-linear ML workflows as directed acyclic graphs. The framework is part of the `mlr3` ecosystem, leveraging convenient resampling, benchmarking, and tuning components.

Keywords: machine learning pipelines, preprocessing, automated machine learning

1. Introduction

As one of the most popular and widely-used software systems for statistics and ML, R (R Core Team, 2020) has several packages that provide a standardized interface for predictive modeling, such as `caret` (Kuhn, 2008), `tidymodels` (Kuhn and Wickham, 2020b), `mlr` (Bischl et al., 2016), and its successor `mlr3` (Lang et al., 2019). But real-world applications often require complex combinations of ML (pre-) processing steps, which can be expressed as a directed acyclic graph (DAG); we will call such graphs ML pipelines or ML workflows. Specifying such a workflow in an ML system without direct support requires error-prone glue code to combine the individual pieces. One particular difficulty is that (in ML) each pipeline operation is not a stateless function application, but consists of a train and predict stage, where the former not only transforms its inputs into an output, but also learns an internal parameter state, which the latter relies on.[1] `mlr3pipelines` provides a domain-specific language which allows building ML pipelines from individual processing operations (`PipeOp`s, also see Figure 1). It ships with a large collection of such operations and allows their custom extension through user-defined operations.

1. This implies that the pipeline idiom in `mlr3pipelines` is quite different compared to `magrittr`, `dplyr`, and `tidymodels`.

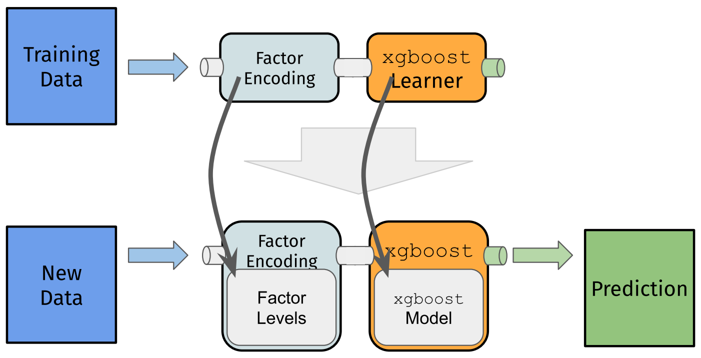


Figure 1: Train and predict steps for a short linear pipeline. Trained `PipeOp`s carry state / parameters (factors and the fitted model) that can be applied to test data.

2. Related Work

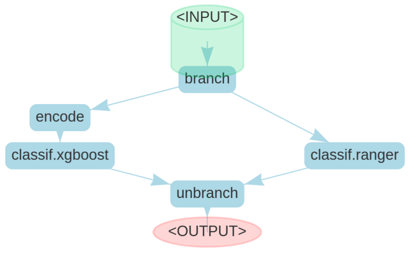
By far, the most widely-used implementation of ML pipelines is Python `scikit-learn`'s (Pedregosa et al., 2011) `pipeline` module (Buitinck et al., 2013). Unlike our software, `scikit-learn` supports only linear pipelines directly, although complex pipelines can be expressed through a wrapper mechanism. Several extensions such as `baikal` (Tineo, 2019) and `neuraxle` (Chevalier et al., 2019) extend scikit-learn's pipelining via a graph-based API similar to ours. `tidymodels` provides the `recipes` R package (Kuhn and Wickham, 2020a) for building linear preprocessing pipelines with limited flexibility and the `workflows` package (Vaughan, 2020) for combining these with models into pipelines. The `mlr` extension `mlrCPO` (Binder, 2021) also focuses on linear pipelines and has limited support for more complex structures. Industry is increasingly providing systems that support ML pipelines, e.g., Microsoft's `ml.net` (Ahmed et al., 2019) for C# and H2O (H2O.ai, 2021) with bindings for Python and R. The `d3m` software (Milutinovic et al., 2017) was developed as part of DARPA's Data Driven Discovery of Models program (Shen, 2018) and includes a pipeline system to combine ML primitives, again without the full flexibility of `mlr3pipelines`. The DAGs in `mlr3pipelines` go beyond simple combinations of preprocessing and ML models. They support ensemble models and conditional branching that can be represented explicitly as part of the graph structure. Existing pipelining frameworks are often limited to passing training or prediction data objects through the pipeline, while `mlr3pipelines` allows for passing arbitrary objects. Some operators, for example, pass on functions, which are then used to influence the behavior of operators later in the graph.

3. Design, Functionality, and Examples

`mlr3pipelines` represents ML workflows as `Graph` objects: DAGs, whose vertices are `PipeOp`s, which represent arbitrary ML processing operations. The pipeline can either be called to train or predict. Inputs and intermediate objects, most commonly data, move along the DAG's edges. When they pass through a vertex, they are processed by the cor-

```
factor_xgboost = po("encode") %>>%
  lrn("classif.xgboost")

pipe = ppl("branch", list(
  xgboost = factor_xgboost,
  ranger = lrn("classif.ranger")
))
```



Listing 1 (left) and Figure 2 (right): Example of a branching pipeline. LHS: The `%>>%` operator builds a linear partial graph. The "ppl branch" template constructs two alternative paths *xgboost* and *ranger*, where the latter does not require factor encoding. A new hyperparameter controls the path through which the data will flow. RHS: The pipeline can be plotted with `pipe$plot(html = TRUE)`.

responding `PipeOp`, and, depending on the call, are either transformed by its `train()` or `predict()` method, where the former also creates the operator's internal state.

This ensures that no information leakage from test data occurs, which is required for the evaluation of predictive systems (Bischl et al., 2012). `mlr3pipelines` and the `mlr3` ecosystem are integrated with each other, so that `mlr3`'s `Learners` can be used as `PipeOps` and `Graphs` adhere to the same interface as `mlr3` learners and can be, for example, resampled and tuned just like any other `Learner`. This also enables effortless parallelization of these operations for pipelines. `mlr3pipelines` provides the `%>>%` operator, which concatenates `Graphs` (or `PipeOps`) into larger `Graphs`. Templates for more complex but frequently used graph patterns are provided through the `ppl()` lookup function. Outputs from different nodes can be combined in non-trivial ways, for example, joining features created by different preprocessing steps, to create non-linear structures. Other examples include alternative path branching (one of several flows is executed, depending on a hyperparameter), ensembling (predictions from different `PipeOps` are averaged), and stacking (predictions from different `PipeOps` are combined in another `PipeOp`, usually a `Learner`, to produce a final prediction). Listing 1 shows an example of branching for model and preprocessing selection. Many more examples can be found at `https://mlr3gallery.mlr-org.com/#category:mlr3pipelines`.

Figure 3 shows examples of complex pipeline components. Some of these are already used in other packages in the `mlr3` ecosystem, e.g., `mlr3proba` (Sonabend et al., 2021) uses the pipeline in Figure 3(i). Users can easily implement their own `PipeOps` and define their exposed hyperparameters, by inheriting from the `PipeOp` class to, for example, implement custom feature extraction and processing.

4. Hyperparameter Tuning and AutoML

Each `Graph` exposes the hyperparameters of its constituent `PipeOps` for joint tuning via any of the automated tuning methods in `mlr3`. Simple tuners such as grid and random search, as well as advanced black-box optimizers like Bayesian Optimization (Snoek et al., 2012) and Hyperband (Li et al., 2018) are available through `mlr3tuning`. Building upon the branching principle of Listing 1, this allows to build entire AutoML systems by combining

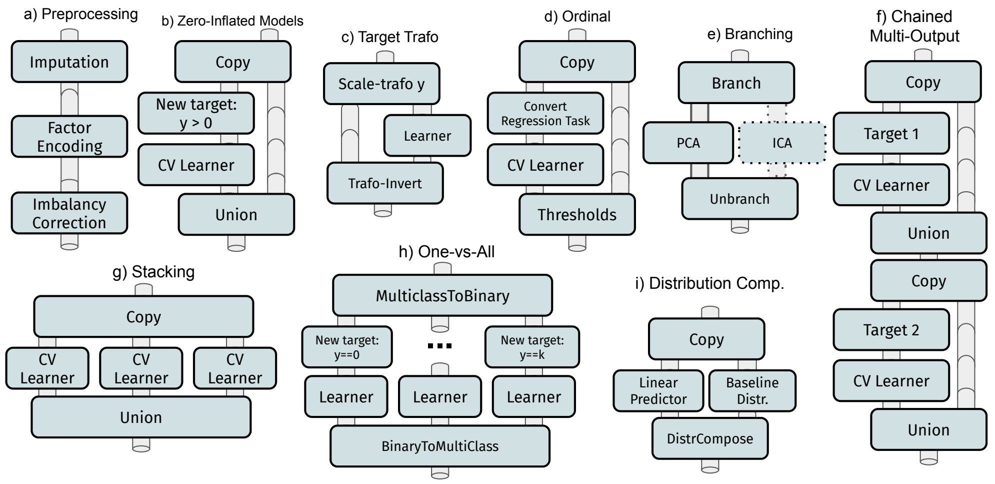


Figure 3: Example pipelines constructed from simple building blocks: (a) typical preprocessing pipeline, (b) zero-inflated data (Zuur et al., 2009), (c) target transformations (scaling to $[0, 1]$) before a model and back afterward, (d) ordinal regression through thresholding, (e) alternative path branching between different options, (f) chaining (Read et al., 2011), (g) stacking (Wolpert, 1992), (h) multiclass classification through ensembling of multiple class-vs-rest binary classifiers, (i) estimation of survival distributions and continuous risk rankings from linear predictors through composition (Sonabend et al., 2021).

multiple learners and preprocessing options. Jointly tuning the selection of these steps with their (subordinate) hyperparameters yields a single pipeline, tailored for a specific task.

5. Availability, Documentation, Code Quality Control

All packages of the `mlr3` ecosystem are released under LGPL-3 on GitHub (`https://github.com/mlr-org`) and on CRAN. Package documentation is available at `https://mlr3pipelines.mlr-org.com` and in the (work-in-progress) mlr3 book (`https://mlr3book.mlr-org.com`), with examples in the mlr3 gallery (`https://mlr3gallery.mlr-org.com`). An extensive suite of unit tests is run on each change via a continuous integration system.

6. Outlook

`mlr3pipelines` is complete and ready for production use. Our focus for future improvements is better integration of automated ML and deep learning (through `mlr3keras` and `mlr3torch`), and leveraging parallel processing specifically for pipelines.

Acknowledgments

This work has been funded by the German Federal Ministry of Education and Research (BMBF) under Grant No. 01IS18036A. LK is supported by NSF grant #1813537.

References

Zeeshan Ahmed, Saeed Amizadeh, Mikhail Bilenko, Rogan Carr, Wei-Sheng Chin, Yael Dekel, Xavier Dupre, Vadim Eksarevskiy, Senja Filipi, Tom Finley, Abhishek Goswami, Monte Hoover, Scott Inglis, Matteo Interlandi, Najeeb Kazmi, Gleb Krivosheev, Pete Luferenko, Ivan Matantsev, Sergiy Matusevych, Shahab Moradi, Gani Nazirov, Justin Ormont, Gal Oshri, Artidoro Pagnoni, Jignesh Parmar, Prabhat Roy, Mohammad Zeeshan Siddiqui, Markus Weimer, Shauheen Zahirazami, and Yiwen Zhu. Machine learning at Microsoft with ML.NET. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2448–2458, 2019.

Martin Binder. *mlrCPO: Composable Preprocessing Operators and Pipelines for Machine Learning*, 2021. URL `https://CRAN.R-project.org/package=mlrCPO`. R package version 0.3.7-2.

Bernd Bischl, Olaf Mersmann, Heike Trautmann, and Claus Weihs. Resampling methods for meta-model validation with recommendations for evolutionary computation. *Evolutionary Computation*, 20(2):249–275, 2012.

Bernd Bischl, Michel Lang, Lars Kotthoff, Julia Schiffner, Jakob Richter, Erich Studerus, Giuseppe Casalicchio, and Zachary M. Jones. mlr: Machine learning in R. *Journal of Machine Learning Research*, 17(170):1–5, 2016. URL `http://jmlr.org/papers/v17/15-066.html`.

Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: Experiences from the scikit-learn project. In *European Conference on Machine Learning and Principles and Practices of Knowledge Discovery in Databases*, 2013.

Guillaume Chevalier, Alexandre Brillant, and Éric Hamel. Neuraxle - a Python framework for neat machine learning pipelines, 09 2019. URL `https://github.com/Neuraxio/Neuraxle`.

H2O.ai. *h2o software*, 10 2021. URL `https://github.com/h2oai/h2o-3`. H2O version 3.32.1.3.

Max Kuhn. Building predictive models in R using the caret package. *Journal of Statistical Software*, 28(5):1–26, 2008.

Max Kuhn and Hadley Wickham. *recipes: Preprocessing tools to create design matrices*, 2020a. URL `https://CRAN.R-project.org/package=recipes`. R package version 0.1.16.

Max Kuhn and Hadley Wickham. *Tidymodels: A collection of packages for modeling and machine learning using tidyverse principles.*, 2020b. URL `https://www.tidymodels.org`.

Michel Lang, Martin Binder, Jakob Richter, Patrick Schratz, Florian Pfisterer, Stefan Coors, Quay Au, Giuseppe Casalicchio, Lars Kotthoff, and Bernd Bischl. mlr3: A modern object-oriented machine learning framework in R. *Journal of Open Source Software*, 4(44):1903, 2019.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18(185):1–52, 2018. URL `http://jmlr.org/papers/v18/16-558.html`.

Mitar Milutinovic, Atılım Güneş Baydin, Robert Zinkov, William Harvey, Dawn Song, Frank Wood, and Wade Shen. End-to-end training of differentiable pipelines across machine learning frameworks. In *31st Conference on Neural Information Processing Systems (NIPS 2017)*, 2017.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12(85):2825–2830, 2011. URL `http://jmlr.org/papers/v12/pedregosa11a.html`.

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2020. URL `https://www.R-project.org/`.

Jesse Read, Bernhard Pfahringer, Geoff Holmes, and Eibe Frank. Classifier chains for multi-label classification. *Machine Learning*, 85(3):333, 2011.

Wade Shen. Darpa's data driven discovery of models (D3M) and software defined hardware (SDH) programs. In D. Chen, H. Homayoun, and B. Taskin, editors, *Proceedings of the 2018 on Great Lakes Symposium on VLSI, GLSVLSI 2018, Chicago, IL, USA, May 23-25, 2018*, page 1. ACM, 2018.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical Bayesian optimization of machine learning algorithms. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 2951–2959. Curran Associates, Inc., 2012.

Raphael Sonabend, Franz J. Király, Andreas Bender, Bernd Bischl, and Michel Lang. mlr3proba: An R package for machine learning in survival analysis. *Bioinformatics*, 02 2021.

Alejandro Gonzalez Tineo. *baikal*, 2019. URL `https://github.com/alegonz/baikal`.

Davis Vaughan. *workflows: Modeling Workflows*, 2020. URL `https://CRAN.R-project.org/package=recipes`. R package version 0.2.1.

David H Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–259, 1992.

Alain F Zuur, Elena N Ieno, Neil J Walker, Anatoly A Saveliev, and Graham M Smith. Zero-truncated and zero-inflated models for count data. In *Mixed Effects Models and Extensions in Ecology with R*, pages 261–293. Springer, New York, NY, USA, 2009.

6.5 Fairness Audits And Bias Mitigation Using mlr3fairness

Contributed Article:
F. Pfisterer, S. Wei, S. Vollmer, M. Lang, and B. Bischl. *Fairness Audits And Debiasing Using mlr3fairness*, Manuscript submitted for publication

Declaration of contributions The project originated with an initial draft by SW, guided and subsequently improved FP and ML. FP then refactored the resulting software and extended it in several directions. FP also implemented additional measures, debiasing techniques, and fair learners along with documentation. The idea of including model and data reporting functionality originated with FP. BB and ML supervised FP and SW with respect to scope, API design, and implementation details. SV provided feedback and further improved the manuscript.

Fairness Audits And Debiasing Using mlr3fairness

by Florian Pfisterer, Siyi Wei, Sebastian Vollmer, Michel Lang, and Bernd Bischl

Abstract Given an increase in data-driven automated decision-making based on machine learning models, it is imperative that along with tools to develop and improve such models there are sufficient capabilities to analyze and assess models with respect to potential biases. We present the package **mlr3fairness**, a collection of metrics and methods that allow for the assessment of bias in machine learning models. Our package implements a variety of widely used fairness metrics that can be used to audit models for potential biases along with a set of visualizations that can help to provide additional insights into such biases. **mlr3fairness** furthermore integrates debiasing methods that can help allevaite biases in ML models through data preprocessing or post-processing of predictions. These allow practicioners to trade off performance and fairness metric that are appropriate for their use case.

Introduction

Humans are increasingly subject to data-driven automated decision-making. Those automated procedures such as credit risk assessments are often applied using predictive models (Galindo and Tamayo, 2000). It is imperative that along with tools to develop and improve such models, we also develop sufficient capabilities to analyze and assess models not only with respect to their robustness and predictive performance, but also address potential biases. This is highlighted by the GDPR requirement to process data fairly. Popular R (R Core Team, 2021) modeling frameworks such as **caret** (Kuhn, 2021), **tidymodels** (Kuhn and Wickham, 2020), **SuperLearner** (Polley et al., 2021), or **mlr** (Bischl et al., 2016) implement a plethora of metrics to measure performance, but fairness metrics are widely missing. This lack of availability can be detrimental to obtaining fair and unbiased models if the result is to forgo bias audits due to the considerable complexity of implementing such metrics. Consequently, there exists considerable necessity for R packages (a) implementing such metrics, and (b) connecting these metrics to existing ML frameworks. If biases are detected and need to be mitigated, we might furthermore want to employ debiasing techniques that tightly integrate with the fitting and evaluation of the resulting models in order to obtain trade-offs between a model's fairness and utility (e.g., predictive accuracy).

In this article, we present the **mlr3fairness** package which builds upon the ML framework **mlr3** (Lang et al., 2019). Our extension contains fairness metrics, fairness visualizations, and model-agnostic pre- and postprocessing operators that aim to reduce biases in ML models. Additionally, **mlr3fairness** comes with reporting functionality to allow for fairness audits and reducing disparities in ML models.

In the remainder of the article, we first provide an introduction to fairness in ML with the goal to raise awareness for biases that can arise due to the use of ML models. Next, we introduce the **mlr3fairness** package, followed by an extensive case study, showcasing the capabilities of **mlr3fairness**. We conclude with a short summary.

Fairness in Machine Learning

Studies have found that data-driven automated decision-making systems often improve over human expertise (c.f. Dawes et al. (1989)) and high stakes decisions can therefore be improved using data-driven systems. This often does not only improve predictions but can also make decisions more efficient through automation. Such systems, often without human oversight, are now ubiquitous in everyday life (O'neil (2016), Eubanks (2018), Noble (2018)). To provide further examples, ML-driven systems are used for highly influential decisions such as loan accommodations (Chen (2018), Turner and McBurnett (2019)), job applications (Schumann et al., 2020), healthcare (Topol, 2019), and criminal sentencing (Angwin et al. (2016), Corbett-Davies et al. (2017), Berk et al. (2018)). With this proliferation, such decisions have become subject to scrutiny as a result of prominent inadequacies or failures, for example in the case of the COMPAS recidivism prediction system (Angwin et al., 2016).

Without proper auditing, those models can unintentionally result in negative consequences for individuals, often from underprivileged groups (Barocas et al., 2019). Several sources of such biases are worth mentioning in this context: Data often contains **historical biases** such as gender or racial stereotypes, that – if picked up by the model – will be replicated into the future. Similarly, unprivileged populations are often not represented in data due to **sampling biases** leading to models that perform well in groups sufficiently represented in the data but worse on others (Buolamwini and Gebru, 2018) - this includes a higher rate of missing data. Other biases include biases in how *labels* and *data* are measured as well as **feedback** loops where repeated decisions affect the population subject to such decisions. For an in-depth discussion and further sources of biases, the interested reader is referred to (Mehrabi et al., 2021, Mitchell et al. (2021)).

Quantifying Fairness

We now turn to the question of how we can detect whether disparities exist in a model and if so, how they can be quantified. What constitutes a fair models depends on a society's ethical values and whether we take a normative position, resulting in different metrics that are applied to a problem at hand. In this article, we focus on a subgroup of these, so-called *statistical group fairness* metrics. First, the observations are grouped by a protected attribute A ($A = 0$ vs. $A = 1$) which, e.g., is an identifier for a person's race or a person's gender. For the sake of simplicity, we consider a *binary classification* scenario and a *binary protected attribute*. Each observation has an associated label $Y, Y \in \{0, 1\}$ we aim to predict, e.g. whether a defendant was caught re-offending. A system then makes a prediction $\hat{Y}, \hat{Y} \in \{0, 1\}$ with the goal to predict whether an individual might re-offend. We assume that $Y = 1$ is the favored outcome in the following exposition. However, the concepts discussed in the following often extend naturally to more complex scenarios including multi-class classification, regression or survival analysis and similarly to settings with multiple protected attributes. We now provide and discuss several metrics grouped into metrics that require *Separation* and *Independence* (Barocas et al., 2019) to provide further intuition regarding core concepts and possible applications.

Separation

One group of widely used fairness notions requires **Separation**: $\hat{Y} \perp A | Y$. This essentially requires, that some notion of model error, e.g. accuracy or false positive rate is equal across groups A. From this notion, we can derive several metrics that come with different implications. It is important to note, that those metrics can only meaningfully identify biases under the assumption that no disparities exist in the data, or that they are legally justified. If e.g. societal biases lead to disparate measurements of an observed quantity (e.g. SAT scores) for individuals with the same underlying ability, *separation* based metrics might not identify existing biases. For this reason, Wachter et al. (2020) refer to those metrics as *bias-preserving* metrics since underlying disparities are not addressed. We now provide and discuss several metrics to provide further intuition regarding core concepts and possible applications.

Equalized Odds A predictor $\hat{Y}$ satisfies *equalized odds* with respect to a protected attribute A and observed outcome Y, if $\hat{Y}$ and A are conditionally independent given Y:

$$\mathbb{P}\left(\hat{Y} = 1 \mid A = 0, Y = y\right) = \mathbb{P}\left(\hat{Y} = 1 \mid A = 1, Y = y\right), \quad y \in \{0, 1\}. \tag{1}$$

In short, we require the same true positive rates (TPR) and false positive rates (FPR) across both groups $A = 0$ and $A = 1$. This intuitively requires, e.g. in the case of university admission, that independent of the protected attribute, qualified individuals have the same chance to be accepted and unqualified individuals are rejected. Similar arguments have been made for equalized false positive rates (Chouldechova, 2017) and false omission rates (Berk et al., 2018) depending on the exact scenario.

Equality of Opportunity A predictor $\hat{Y}$ satisfies *equality of opportunity* with respect to a protected attribute A and observed outcome Y, if $\hat{Y}$ and A are conditionally independent given $Y = 1$. This is a relaxation of the aforementioned *equalized odds* essentially only requiring equal TPRs:

$$\mathbb{P}\left(\hat{Y} = 1 \mid A = 0, Y = 1\right) = \mathbb{P}\left(\hat{Y} = 1 \mid A = 1, Y = 1\right). \tag{2}$$

Intuitively, this only requires that independent of the protected attribute, qualified individuals have the same chance of being accepted.

Performance Parity A more general formulation can be applied when we require parity of some performance metric across groups. To provide an example, Buolamwini and Gebru (2018) compare accuracy across intersectional subgroups, essentially arguing that model performance should be equal across groups:

$$\mathbb{P}\left(\hat{Y} = Y \mid A = 0\right) = \mathbb{P}\left(\hat{Y} = Y \mid A = 1\right) \quad Y \in \{0, 1\}. \tag{3}$$

This intuitively requires, that the model should work equally well for all groups, i.e. individuals are correctly accepted or denied at the same rate, independent of the predicted attribute. This notion can be extended across supervised learning settings and performance metrics, leading to considerations of equal mean squared error, e.g. in a regression setting.

Independence (Demographic Parity)

A second group of fairness metrics is given by so-called *bias-transforming* metrics (Wachter et al., 2020). They require, that decision rates, such as the positive rate are equal across groups. This notion can identify biases e.g. arising from societal biases that manifest in different base rates across groups. At the same time, employing such notions poses a considerable risk, as blindly optimizing for demographic parity might result in predictors that e.g. jail innocent people from an advantaged group in order to achieve parity across both groups (Dwork et al., 2012, Berk et al. (2018)).

A predictor $\hat{Y}$ satisfies *demographic parity* [@{Calders2010] with respect to a protected attribute A and observed outcome Y, if $\hat{Y}$ and A are conditionally independent.

$$\mathbb{P}\left(\hat{Y}=1 \mid A=0\right)=\mathbb{P}\left(\hat{Y}=1 \mid A=1\right). \tag{4}$$

In contrast to the previous definitions, this only requires that the chance of being accepted is equal across groups.

Fairness metrics In order to encode the requirements in equations 1) - 4 into a fairness metric, we often encode differences between measured quantities in two groups. For a performance metric M, e.g. the true positive rate (TPR), we calculate the difference in the metric across the two groups:

$$\Delta_{\mathrm{M}} = \mathrm{M}_{A=0} - \mathrm{M}_{A=1}.$$

When Δ_{M} now significantly deviates from 0, this can be indicative of a fairness violation with respect to the fairness notion described via M. To provide an example, with $\mathbb{P}\left(\hat{Y}=1 \mid A=\star, Y=1\right)$ denoted with $\mathrm{TPR}_{A=\star}$, we calculate the difference in TPR between the two groups:

$$\Delta_{\mathrm{TPR}} = \mathrm{TPR}_{A=0} - \mathrm{TPR}_{A=1}.$$

When Δ_{FPR} now significantly deviates from 0, the prediction $\hat{Y}$ violates the requirement for *equality of opportunity* formulated above.

It is important to note, that in practice we might not be able to perfectly satisfy a given metric, e.g. due to stochasticity in data and labels. Instead, to provide a binary conclusion regarding fairness, a model could be considered fair, if $|\Delta_{\mathrm{TPR}}| < \epsilon$ for a given threshold $\epsilon > 0$, e.g., $\epsilon = 0.05$. This allows for small deviations from perfect fairness due to variance in the estimation of $\mathrm{TPR}_{A=\star}$ or additional sources of bias. It is important to note, that choosing apropriate thresholds is difficult and widely used values for ϵ such as 0.05 are arbitrary and do not translate to legal doctrines, such as e.g. disparate impact (Watkins et al., 2022). A more in-depth treatment of metrics along with additional fairness metrics are described in (Saleiro et al. (2018), Kim et al. (2020), Mehrabi et al. (2021) and Wachter et al. (2020)).

Selecting fairness metrics While the three metrics are conceptually similar, they encode a different belief of what constitutes *fair* in a given scenario. Wachter et al. (2020) differentiate between *bias-preserving* and *bias transforming* metrics: Bias-preserving metrics such as equalized odds and equality of opportunity require that errors made by a model are equal across groups. This can help to detect biases, stemming e.g. from data acquisition, but might be problematic in cases where e.g. labels are biased. To provide an example, police enforcement and subsequent arrests of violent re-offenders might be different across ZIP code areas, a proxy for race. This might lead to situations where labels Y suffer from differential measurement bias strongly correlated with race (Bao et al., 2021).

Bias-transforming methods, in contrast do not depend on the labels and might therefore not suffer from this problem. They can help detecting biases arising from different base-rates across populations, arising e.g. from aforementioned biases in the labeling or as a consequence of structural discrimination. Deciding which metrics to use constitutes a value judgement and requires careful assessment of the societal context a decision making system is deployed in. A discussion of different metrics and their applicability can be found in the Aequitas Fairness Toolkit (Saleiro et al., 2018) which also provides guidance towards selecting a metric via the Aequitas Fairness Tree. Wachter et al. (2020) recommend using *bias-transforming* metrics and provide a checklist that can guide the choice of fairness metric. Corbett-Davies and Goel (2018) on the other hand point out several limitations of available metrics and argue for grounding decisions in real world quantities in addition to abstract fairness metrics. Similarly, Friedler et al. (2016) emphasize the need to differentiate between constructs we aim to measure (e.g. job-related knowledge) and the observed quantity that can be measured in practice (e.g. years in a job) when trying to automate decision, since disparities in how constructs translate to observed quantities might suffer from bias. To provide an example, individuals with similar ability might exhibit different measured quantities (grades) due to structural bias, e.g. worse access to after-school tutoring programs.

The dangers of fairness metrics We want to stress, that overly trusting in metrics can be dangerous and that fairness metrics can and should not be used to *prove* or *guarantee* fairness. Whether a selected fairness notion (and a corresponding numerical value) is actually fair depends on the societal context in which a decision is made and which action should be derived from a given prediction. Therefore, selecting the correct fairness metric requires a thorough understanding of the societal context, decisions are made in, as well as possible implications of such decisions. To provide an example, in some cases discrepancies in positive predictions might be justified or even desired, as they, e.g. allow for a more nuanced, gender-specific diagnosis (**?**). Furthermore, fairness metrics might not detect biases in more fine-grained subgroups, e.g. at the intersection of multiple protected attributes. It is also important to note, that fairness metrics merely provide a reduction of the aforementioned fairness notions into mathematical objectives. As such, they require a variety of abstraction steps that might invalidate the metric (Watkins et al., 2022), as they e.g. require that the data is a large enough and representative sample of the entire population that we aim to investigate. Furthermore, practitioners need to look beyond the model, but also at the data used for training and the process of data and label acquisition. If the data e.g. exhibits disparate measurement errors in the features or labels, valid fairness assessments can become impossible. Similarly, feedback loops might arise from a prediction leading to changes in the data collected in the future. Even an *initially fair* model might then lead to adverse effects in the long term (Schwöbel and Remmers, 2022).

Note, that the fairness definitions presented above serve a dual purpose (Wachter et al., 2020): First, as a *diagnostic tool* with the goal to detect disparities. This e.g. allows assessing whether a model has inherited biases, e.g. from historical disparities reflected in the data. The second purpose is as a basis for *model selection* and making fair decisions in practice. In this setting, fairness notions are employed with the goal to audit ML models or to select which model should be used in practice. It is important to note, that fairness metrics should however not be used as the sole basis for making decisions about whether to employ a given ML model or to assess whether a given system is fair. We therefore explicitly encourage using the presented metrics in an explorative manner.

Other notions of fairness In addition to *statistical group fairness notions* introduced above, several additional fairness notions exist. The notion of *individual fairness* was proposed by (Dwork et al., 2012). Its core idea comes from the principle of *treating similar cases similarly and different cases differently*. In contrast to statistical group fairness notions, this notion allows assessing *fairness* at an individual level and would therefore allow determining, whether an individual is treated fairly. A more in-depth treatment of individual fairness notions is given by Binns (2020) and Heidari et al. (2019). Similarly, a variety of *causal* fairness notions exist (c.f. Kilbertus et al. (2017)). They argue, that assessing fairness requires incorporating causal relationships in the data.

Fairness Constraints

Statistical group fairness notions suffer from two further problems in practice: First, it might be hard to exactly satisfy the required fairness notions, e.g. due to limited amount of data available for evaluation. Secondly, only requiring fairness might lead to degenerate solutions (Corbett-Davies and Goel, 2018) or models that have low utility, e.g. in separating *good* and *bad* credit risk. One approach to take this into account is to employ models which maximize utility but satisfy some maximum constraint on potential unfairness. This can be achieved via constraints on the employed fairness measure, e.g. $|\Delta M| \leq \epsilon$ requiring that the absolute difference in a metric M between groups is smaller than a chosen value ϵ. In the following, we denote the fairness metric we want to minimize with ΔM and the performance metric with ρ (assuming the latter should be maximized).

$$\rho_{|\Delta|M \leq \epsilon} = \begin{cases} \rho & |\Delta M| \leq \epsilon \\ -|\Delta M| & \text{else} \end{cases}$$

This approach has e.g. been employed by (Perrone et al., 2021) as the objective of an AutoML system. It is not immediately clear, how the constraint ϵ should be chosen. An alternative therefore is to employ *multi-objective optimization* in order to investigate available trade-offs between performance and accuracy metrics. This can be done via **mlr3tuning** which contains functionality to tune models for multiple metrics, described in more detail in the mlr3book. The result of optimization is the *pareto-set*: A list of models which optimally trade off the specified objectives.

Debiasing Models

If biases are detected in a model, we might now be interested in improving models in order to potentially mitigate such biases. Bias in models might arise from a variety of sources, so a careful understanding of the data, data quality and distribution might lead to approaches that can help in decreasing biases, e.g. through the collection of better or additional data or a better balancing of protected groups. Similarly, biases might arise from the model, e.g. through under- or overfitting and more careful tuning of model hyperparameters might help with improving fairness. Especially in the case of *bias-transforming* metrics, a better solution might often be to address fairness problems in the real world instead of relying on algorithmic interventions to solve fairness not only momentarily. In addition, a variety of algorithmic **debiasing techniques**, that might help with obtaining fairer models have been proposed. Their goal is to reduce measured gaps in fairness, either via data pre-processing, employing models that incorporate fairness or by applying post-processing techniques on a model's predictions. Popular examples for such techniques include computing instance weights before training (Kamiran and Calders, 2012), where each observation is weighted proportional to the inverse frequency of it's label and protected attribute. Other methods work by directly learning fair models that incorporate fairness constraints into the fitting procedure (Zafar et al., 2017) or by adapting model predictions, e.g. (Hardt et al., 2016) propose to randomly flip a small fraction of predictions in each group given by $\hat{Y}$ and A, such that fairness metrics are satisfied in expectation. Since debiasing techniques are often tailored towards a particular fairness metric, the optimal choice of debiasing technique is often not trivial and a combination of algorithms and debiasing techniques, e.g. determined via tuning might result in an optimal model.

Bias mitigation techniques, as proposed above have the goal to mitigate fairness issues, as e.g. measured by fairness metrics. In practice, this usually comes with several drawbacks: First, bias mitigation strategies often lead to a decrease in a classifier's predictive performance (Corbett-Davies and Goel, 2018). In addition, processing schemes can worsen interpretability or introduce stochasticity during prediction (see e.g. (Hardt et al., 2016)). Furthermore, we want to caution against favoring bias mitigation techniques over policy interventions that tackle biases at their root cause. A different set of risk is posed by *fairwashing* (Aivodji et al., 2019), i.e. finding fair explanations or satisfying fairness metrics for otherwise unfair models. If biases are only addressed at a given moment and without regard for downstream effects, they might simultaneously lead to a decrease in predictive performance in the near term and to negative consequences for the protected group in the long term (Schwöbel and Remmers, 2022).

mlr3fairness

In this section, we first give an overview of related software. Next, we give a very briefly introduce to the **mlr3** ecosystem of packages. Finally, the implemented extensions for fairness are presented.

Related Software

Several R packages provide similar capabilities to our software, but mostly focus on fairness metrics and visualization. The **fairness** package (Kozodoi and V. Varga, 2021) allows for the calculation of a variety of fairness metrics, while **aif360** (Bellamy et al., 2018) wraps the Python **aif360** module allowing for the computation of fairness metrics and several debiasing techniques but has only limited interoperability with R objects such as `data.frames`. The **fairmodels**(Wiśniewski and Biecek, 2022) package again allows for the computation of fairness metrics for classification and regression settings as well as several debiasing techniques. It tightly integrates with **DALEX** (Biecek, 2018) to gain further insight using interpretability techniques.

Outside R, in Python, the **fairlearn** module (Bird et al., 2020) provides ample functionality to study a wide variety of metrics, debias with respect to a variety of pre-, in- and postprocessing methods as well as to visualize differences. It furthermore provides a *fairlearn dashboard* providing a comprehensive fairness report. The **aif360** (Bellamy et al., 2018) module similarly provides metrics as well as debiasing techniques while the **aequitas** fairness toolkit (Saleiro et al., 2018) provides similar capabilities. Interoperability with the **scikit-learn** (Pedregosa et al., 2011) ML framework allows for debiasing a wide variety of ML models in all aforementioned systems. Similar capabilities are also available in Julia's **Fairness.jl** (Agrawal et al., 2020a) library.

The mlr3 Ecosystem

mlr3fairness is tightly integrated into the ecosystem of packages around the ML framework **mlr3** (Lang et al., 2019). **mlr3** provides the infrastructure to fit, resample, and evaluate over 100 ML algorithms using a unified API. Multiple extension packages bring numerous additional advantages and extra functionality. In the context of fairness, the following extension packages deserve special mention:

- **mlr3pipelines** (Binder et al., 2021) for pre- and postprocessing via pipelining. This allows merging debiasing with arbitrary ML algorithms shipped with **mlr3** as well as comparison of different models through joint resampling and tuning. It furthermore integrates with **mcboost** (Pfisterer et al., 2021), which implements additional debiasing methods. We present an example in the supplementary material.
- **mlr3tuning** for its extensive tuning capabilities. Fusing debiasing techniques with ML algorithms as well as other often necessary preprocessing steps such as imputation of missing values or class balancing allows for joint tuning of hyperparameters with respect to arbitrary performance and fairness metrics.
- **mlr3proba** (Sonabend et al., 2021) for survival analysis.
- **mlr3benchmark** for post-hoc analysis of benchmarked approaches.
- **mlr3oml** as a connector to OpenML (Vanschoren et al., 2014), an online scientific platform for collaborative ML.

In order to provide the required understanding for **mlr3**, we briefly introduce some terminology and syntax. A full introduction can be found in the mlr3 book (Becker et al., 2022).

A `Task` in **mlr3** is a basic building block holding the data, storing covariates and the target variable along with some meta-information. The shorthand constructor function `tsk()` can be used to quickly access example tasks shipped with **mlr3** or **mlr3fairness**. In the following chunk, we retrieve the binary classification task with id `"compas_race_binary"` from the package. It contains a simplified version of the COMPAS data set (Angwin et al., 2016). The task is to predict whether a parolee will re-offend within a span of 2 years. The column `"race"` is set as a binary protected attribute with levels `"Caucasian"` and `"African American"`.

```
library("mlr3verse")
library("mlr3fairness")

# get a simplified compas example data set
task = tsk("compas_race_binary")
```

```
#> Warning in (function () : Using the COMPAS dataset for benchmarking is
#> generally discouraged in fairness literature. See `help('compas') for additional
#> information.`
```

```
print(task)
```

```
#> <TaskClassif:compas_race_binary> (5278 x 11)
#> * Target: two_year_recid
#> * Properties: twoclass
#> * Features (10):
#>   - int (5): age, days_b_screening_arrest, decile_score,
#>     length_of_stay, priors_count
#>   - fct (5): age_cat, c_charge_degree, race, score_text, sex
#> * 1: pta
```

The protected attribute(s) are identified by a `col_role` named `pta` and can be set accordingly, e.g. via `task$col_roles$pta = c("gender, "race")`. If more than one protected attribute is specified, metrics will be computed based on intersecting groups formed by the columns.

The second building block is the `Learner`. It is a wrapper around an ML algorithm, e.g., an implementation of logistic regression or a decision tree. It can be trained on a `Task` and used for obtaining a `Prediction` on an independent test set which can subsequently be scored using a `Measure` to get an estimate for the predictive performance on new data. The shorthand constructors `lrn()` and `msr()` allow for the instantiation of implemented `Learners` and `Measures`, respectively. In the following example, we will instantiate a learner, train it on the train set of the dataset and evaluate predictions on held-out test data. The train-test split in this case is given by row indices, here stored in the `idx` variable.

```
# initialize a classification tree from package rpart, predicting probabilities
learner = lrn("classif.rpart", predict_type = "prob")
```

```
# split into a list with train and test set
idx = partition(task)
# fit model on train set
learner$train(task, idx$train)
# predict on observations of test set
prediction = learner$predict(task, idx$test)
```

We then employ the `classif.acc` measure which measures the accuracy of a prediction compared to the true label:

```
measure = msr("classif.acc")
prediction$score(measure)
```

```
#> classif.acc
#>   0.6622631
```

In the example above, we obtain an accuracy score of 0.6623, meaning our ML model correctly classifies roughly 66 % of the samples in the test data. As the split into training set and test set is stochastic, the procedure should be repeated multiple times for smaller datasets (Bischl et al., 2012) and the resulting performance values should be aggregated. This process is called resampling, and can easily be performed with the `resample()` function, yielding a `ResampleResult` object. In the following, we employ 10-fold cross-validation as a resampling strategy.

```
resampling = rsmp("cv", folds = 10)
rr = resample(task, learner, resampling)
rr$aggregate(measure)
```

```
#> classif.acc
#>   0.6754507
```

We can call the `aggregate` method on the `ResampleResult` to obtain the accuracy aggregated across all 10 replications. Here, we again obtain an accuracy of 0.6755, so slightly higher than previous scores, due to using a larger fraction of the data. Furthermore, this estimate has a lower variance (as it is an aggregate) at the cost of additional computation time. To properly compare competing modeling approaches, candidates can be benchmarked against each other using the `benchmark()` function (yielding a `BenchmarkResult`). In the following, we compare the decision tree from above to a logistic regression model. To do this, we use the `benchmark_grid` function to compare the two `Learners` across the same `Task` and resampling procedure.

```
learner2 = lrn("classif.log_reg", predict_type = "prob")

# build an exhaustive grid design and run benchmark
grid = benchmark_grid(task, list(learner, learner2), resampling)
bmr = benchmark(grid)
bmr$aggregate(measure)[, .(learner_id, classif.acc)]
```

```
#>         learner_id classif.acc
#> 1:   classif.rpart   0.6716531
#> 2: classif.log_reg   0.6851047
```

After running the benchmark, we can again call `.$aggregate` to obtain aggregated scores. The **mlr3viz** package comes with several ready-made visualizations for objects from `mlr3` via the `autoplot` function. For a `BenchmarkResult`, the `autoplot` function provides a Box-plot comparison of performances across the cross-validation folds for each `Learner`. Figure **?**(fig:bmrbox) contains the box-plot comparison. We can see, that `log_reg` has a higher accuracy and lower inter-quartile range across the 10 folds and we might therefore want to prefer the `log_reg` model.

Selecting the protected attribute

For a given task, we can select one or multiple protected attributes. In **mlr3**, the protected attribute is identified by the column role `pta` and can be set as follows:

```
task$col_roles$pta = "race"
```

This information is then automatically passed on when the task is used, e.g. when computing fairness metrics.

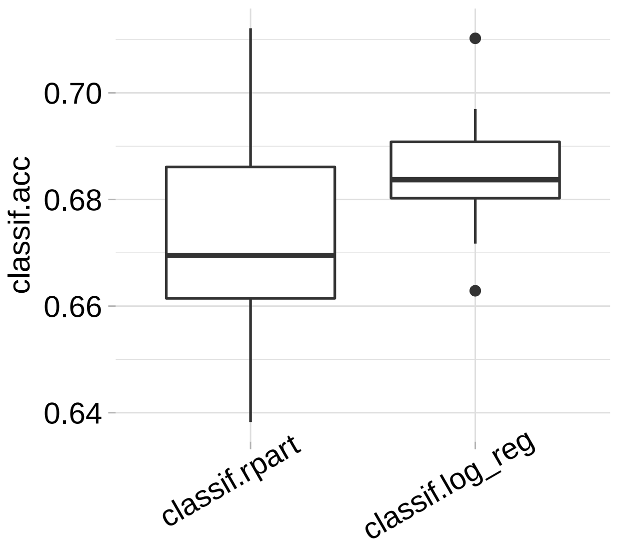


Figure 1: Model comparison for decision trees (rpart) and logistic regression (log_reg).

Quantifying Fairness

With the **mlr3fairness** package loaded, fairness measures can be constructed via `msr()` like any other measure in **mlr3**. They are listed with prefix *fairness*, and simply calling `msr()` without any arguments will return a list of all available measures. Table 1 provides a brief overview over some popular fairness measures which are readily available. The full list can be obtained from `mlr_measures_fairness`.

Table 1: Selection of implemented fairness metrics.

key	description
fairness.acc	Accuracy equality (Buolamwini and Gebru, 2018)
fairness.mse	Mean squared error equality (Regression)
fairness.eod	Equalized odds (Hardt et al., 2016)
fairness.tpr	True positive rate equality / Equality of opportunity (Hardt et al., 2016)
fairness.fpr	False positive rate equality / Predictive equality (Chouldechova, 2017)
fairness.tnr	True negative rate equality
fairness.fnr	False negative rate equality (Berk et al., 2018)
fairness.fomr	False omission rate equality (Berk et al., 2018)
fairness.tnr	Negative predictive value equality
fairness.tnr	Positive predictive value equality
fairness.cv	Demographic parity / Equalized positive rates (Calders and Verwer, 2010)
fairness.pp	Predictive parity / Equalized precision (Chouldechova, 2017)
fairness.{tp, fp, tn, fn}	Equal true positives, false positives, . . .
fairness.acc_eod=.05	Accuracy under equalized odds constraint (Perrone et al., 2021)
fairness.acc_ppv=.05	Accuracy under ppv constraint (Perrone et al., 2021)

Furthermore, new custom fairness measures can be easily implemented, either by implementing them directly or by composing them from existing metrics. This process is extensively documented in an accompanying vignette.

Here we choose the binary accuracy measure `"classif.acc"` and the equalized odds metric from above using `"fairness.eod"`: The constructed list of measures can then be used to score a `Prediction`, a `ResampleResult` or `BenchmarkResult`, e.g.

```
measures = list(msr("classif.acc"), msr("fairness.eod"))
rr$aggregate(measures)
```

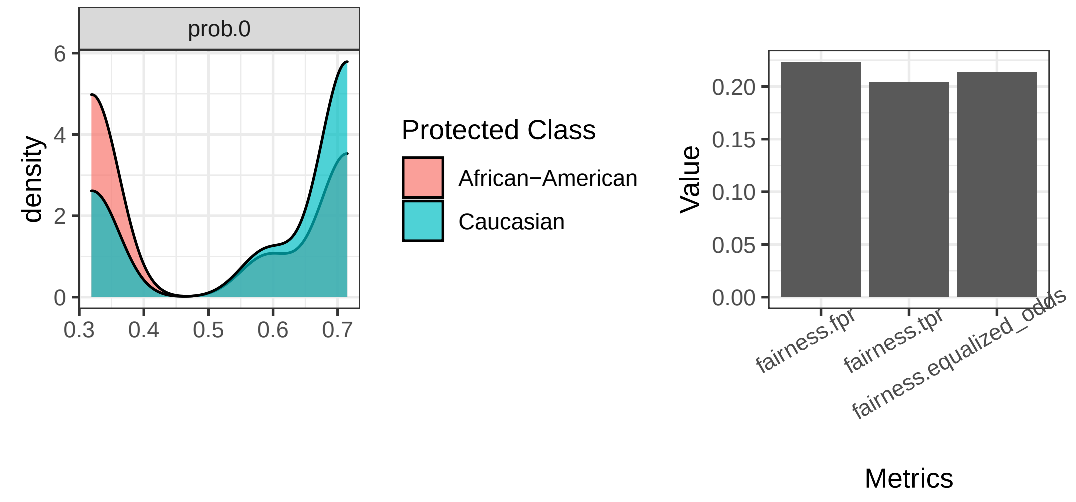


Figure 2: Left: Prediction densities for the negative class for races Caucasian and African-American. Right: Fairness metrics comparison for FPR, TPR, EOd fairness metrics.

```
#>             classif.acc fairness.equalized_odds
#>               0.6754507               0.2046731
```

We can clearly see a comparatively large difference in equalized odds at around 0.2. This means, that in total, the false positive rates (FPR) and true positive rates (TPR) on average differ by ~0.2, indicating that our model might exhibit a bias. Looking at the individual components, yields a clearer picture. Here, we are looking at the confusion matrices of the combined predictions of the 10 folds, grouped by protected attribute:

```
fairness_tensor(rr)

#> $`African-American`
#>         truth
#> response          0          1
#>        0 0.18112922 0.09321713
#>        1 0.10572186 0.22148541
#>
#> $Caucasian
#>         truth
#> response          0          1
#>        0 0.19780220 0.08071239
#>        1 0.04490337 0.07502842
```

Plotting the prediction density or comparing measures graphically often provides additional insights: We can e.g. see, that African-American defendants are more often assigned low probabilities of not re-offending (predicted class `0`). Similarly, we can see that both equality in FPR and TPR differ considerably.

```
fairness_prediction_density(prediction, task)
compare_metrics(prediction, msrs(c("fairness.fpr", "fairness.tpr", "fairness.eod")), task)
```

Debiasing

As mentioned above, several ways to improve a model's fairness exist. While non-technical interventions, such as e.g. collecting more data should be prefered, **mlr3fairnes** provides several debiasing techniques that can be used together with a `Learner` to obtain fairer models. Table 2 provides an overview over implemented debiasing techniques. They are implemented as `PipeOps` from the **mlr3pipelines** package and can be combined with arbitrary learners to build a pipeline. An introduction to **mlr3pipelines** is available in the corresponding mlr3book chapter.

```
# Automatically reweigh data before training a learner:
po("reweighing_wts") %>>% po("learner", lrn("classif.glmnet"))

# Post-process predictions for equalized odds.
po("learner_cv", lrn("classif.glmnet")) %>>% po("EOd")
```

Table 2: Overview over available debiasing techniques.

Key	Description	Type	Reference
EOd	Equalized-Odds Debiasing	Postprocessing	Hardt et al. (2016)
reweighing_os	Reweighing (Oversampling)	Preprocessing	Kamiran and Calders (2012)
reweighing_wts	Reweighing (Instance Weights)	Preprocessing	Kamiran and Calders (2012)

It is simple for users or package developers to extend **mlr3fairness** with additional debiasing methods – as an example, the **mcboost** package adds further postprocessing methods that can improve fairness. Along with pipeline operators, **mlr3fairness** contains several algorithms that can directly incorporate fairness constraints. They can similarly be constructed using `lrn()`.

Table 3: Overview over fair ML algorihtms.

key	package	reference
regr.fairfrrm	fairml	Scutari et al. (2021)
classif.fairfgrrm	fairml	Scutari et al. (2021)
regr.fairzlm	fairml	Zafar et al. (2017)
classif.fairzlrm	fairml	Zafar et al. (2017)
regr.fairnclm	fairml	Komiyama et al. (2018)

Reports

Because fairness aspects can not always be investigated based on the fairness definitions above (e.g., due to biased sampling or labelling procedures), it is important to document data collection and the resulting data as well as the models resulting from this data. Informing auditors about those aspects of a deployed model can lead to better assessments of a model's fairness. Questionnaires for ML models (Mitchell et al., 2019) and data sets (Gebru et al., 2021) have been proposed in literature. We further add an automated report template using R markdown (Xie et al., 2020) which includes many fairness metrics and visualizations to provide a good starting point in order to generate a full fairness report inspired by similar reports offered in the *Aequitas Toolkit* (Saleiro et al., 2018). A preview for the different reports can be obtained from the Reports vignette.

Table: Overview of reports generated by mlr3fairness. | Report | Description | Reference | |———————|—————————|——————————| | `report_modelcard` | Modelcard for ML models | Mitchell et al. (2019) | | `report_datasheet` | Datasheet for data sets | Gebru et al. (2021) | | `report_fairness` | Fairness Report | – |

Case Study

In order to demonstrate a full workflow, we conduct full bias assessment and debiasing on the popular adult data set (Dua and Graff, 2017). The goal is to predict whether an individual's income is larger than $50.000 with the protected attribute being *gender*. The data set ships with **mlr3fairness**, separated into a *train* and *test* task and can be instantiated using `tsk("adult_train")` and `tsk("adult_test")`, respectively. As a fairness metric, we consider *predictive parity* (Chouldechova, 2017) which calls for equality in true positive rates between groups. We furthermore are interested in the model's utility, here measured with its classification accuracy.

```
library("mlr3verse")
library("mlr3fairness")

task = tsk("adult_train")
print(task)
```

```
#> <TaskClassif:adult_train> (30718 x 13)
#> * Target: target
#> * Properties: twoclass
#> * Features (12):
#>   - fct (7): education, martial_status, occupation, race, relationship,
#>     sex, workclass
#>   - int (5): age, capital_gain, capital_loss, education_num,
#>     hours_per_week
#> * 1: pta
```

```
measures = msrs(c("fairness.tpr", "classif.acc"))
```

In order to get an initial perspective, we benchmark three models using 3-fold cross-validation each:

- a classification tree from the **rpart** package,
- a penalized logistic regression from the **glmnet** package and
- a penalized logistic regression from the **glmnet** package, but with reweighing preprocessing.

The logistic regression in the latter two approaches do not support operating on factor features natively, therefore we pre-process the data with a feature encoder from **mlr3pipelines**:

```
set.seed(4321)
learners = list(
    lrn("classif.rpart"),
    po("encode") %>>% lrn("classif.glmnet"),
    po("encode") %>>% po("reweighing_wts") %>>% po("learner", lrn("classif.glmnet"))
)
grid = benchmark_grid(
  tasks = tsks("adult_train"),
  learners = learners,
  resamplings = rsmp("cv", folds = 3)
)
bmr1 = benchmark(grid)
bmr1$aggregate(measures)[, c(4, 7, 8)]
```

```
#>                              learner_id fairness.tpr classif.acc
#> 1:                        classif.rpart  0.059767256   0.8407773
#> 2:               encode.classif.glmnet  0.070780981   0.8411354
#> 3: encode.reweighing_wts.classif.glmnet  0.004731584   0.8351453
```

The preprocessing step of reweighing already improved the fairness while sacrificing only a tiny bit of performance. To see if we can further improve, we use **mlr3tuning** to jointly tune all hyperparameters of the *glmnet* model as well as our reweighing hyperparameter. In order to do this, we use an `AutoTuner` from **mlr3tuning**; a model that tunes its own hyperparameters during training. The full code for setting up this model can be found in the appendix. An `AutoTuner` requires a specific metric to tune for. Here, we define a fairness-thresholded accuracy metric. We set $\epsilon = 0.01$ as a threshold.

$$if\ |\Delta_{EOd}| \leq \epsilon : accuracy\ \ else:\ -|\Delta_{EOd}|$$

```
metric = msr("fairness.constraint",
    performance_measure = msr("classif.acc"),
    fairness_measure = msr("fairness.eod"),
    epsilon = 0.01
)
```

We then design the pipeline and the hyperparameters we want to tune over.

In the following example, we choose `tuning_iters = 3L` and choose a small range for the hyperparameters in `vals` to shorten the run time of the tuning procedure. In real settings, this parameter would be set to a larger number, such as 100.

```
tuning_iters = 3L
at = AutoTuner$new(lrn, rsmp("holdout"),
    metric,
```

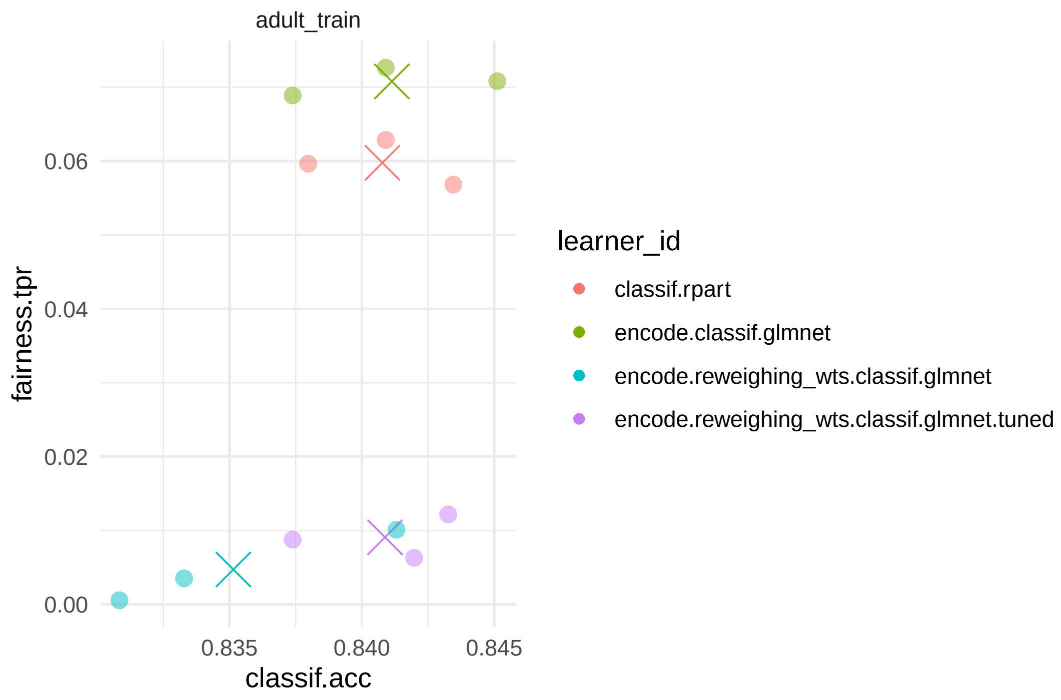


Figure 3: Fairness-Accuracy tradeoff for 3-fold CV on the adult train set.

```
    tuner = mlr3tuning::tnr("random_search"),
    terminator = trm("evals", n_evals = tuning_iters)
)

grd = benchmark_grid(
  tasks = tsks("adult_train"),
  learners = list(at),
  resamplings = rsmp("cv", folds = 3)
)

bmr2 = benchmark(grd, store_models = TRUE)
bmr2$aggregate(measures)[, c(4, 7, 8)]
```

```
#>                                       learner_id fairness.tpr classif.acc
#> 1: encode.reweighing_wts.classif.glmnet.tuned  0.009085494    0.840875
```

The result improves w.r.t. accuracy while only slightly decreasing the measured fairness. Note, that the generalization error is estimated using a holdout strategy during training and slight violations of the desired threshold ϵ can therefore happen. The results of both benchmark experiments can then be collected and jointly visualized in Figure **?**(fig:fat). In addition to aggregate scores (denoted by a cross) individual iterations of the 3 fold Cross-Validation (denoted by points) are shown to visualize variations in the individual results.

and print the aggregated scores:

```
bmr$aggregate(measures)[, c(4, 7, 8)]
```

```
#>                                       learner_id fairness.tpr classif.acc
#> 1:                                 classif.rpart  0.059767256   0.8407773
#> 2:                         encode.classif.glmnet  0.070780981   0.8411354
#> 3:          encode.reweighing_wts.classif.glmnet  0.004731584   0.8351453
#> 4: encode.reweighing_wts.classif.glmnet.tuned  0.009085494   0.8408750
```

Especially when considering optimizing accuracy while still retaining a fair model, tuning can be helpful and further improve upon available trade-offs. In this example, the `AutoTuner` improves w.r.t. the fairness metric while offering accuracy comparable with the simple `glmnet` model. Whether the achieved accuracy is sufficient, needs to be determined, e.g. from business context. For now, we assume that the model obtained from the AutoTuner is the model we might want to use going forward. Having decided for a final model, we can now train the final model

```
at_lrn = bmr$learners$learner[[4]]
at_lrn$train(tsk("adult_train"))
```

and predict on the held out test set available for the Adult dataset to obtain a final estimate. This is important since estimating fairness metrics often incurs significant variance (Agrawal et al., 2020b) and evaluation of the test-set provides us with an unbiased estimate of model performance after the previous model selection step.

```
test = tsk("adult_test")
at_lrn$predict(test)$score(measures, test)

#> fairness.tpr  classif.acc
#>   0.00734166   0.83421482
```

On the held-out test set, the fairness constraint is slightly violated which can happen due to the comparatively large variance in the estimation of fairness metrics.

Summary

The large-scale availability and use of automated decision making systems have resulted in growing concerns for a lack of fairness in the decisions made by such systems. As a result, fairness auditing methods that allow for investigating (un-)fairness in such systems are required. Implementations of such methods are still not widely available, especially considering the required interoperability with machine learning toolkits that allows for ease of use and integration into model evaluation and tuning. In future work we plan on implementing several tools that further support the user w.r.t. pinpointing potential fairness issues in the data, especially through the help of interpretability tools, such as the **iml** package. We furthermore aim to implement additional fairness metrics from the realm of 'individual fairness' (Dwork et al., 2012) and 'conditional demographic parity' [wachter-vlr2020].

Appendix

Tuning the ML Pipeline

We include the full code to construct the `AutoTuner` with additional details and comments below. We first load all required packages and use **mlr3**'s interaction with **future** to automatically distribute the tuning to all available cores in parallel by setting a `plan`. See the documentation of **future** for platform-specific hints regarding parallelization.

```
library(mlr3misc)
library(mlr3)
library(mlr3pipelines)
library(mlr3fairness)
library(mlr3tuning)

# Enable paralellization utilizing all cores
# future::plan("multicore")
```

We then instantiate an ML pipeline using **mlr3pipelines**. This connects several modeling steps, in our case **categorical encoding**, **reweighing** and a final **learner** using the `%>>%` (double caret) operator, ultimately forming a new learner. This learner can then subsequently be fit on a `Task`. We use the `po(<key>)` shorthand to construct a new pipeline operator from a dictionary of implemented operators. We conduct **categorical encoding** because **glmnet** can not naturally handle categorical variables and we therefore have to encode them (in our case using `one-hot` encoding).

```
# Define the learner pipeline.
lrn = as_learner(po("encode") %>>% po("reweighing_wts") %>>% po("learner", lrn("classif.glmnet")))
```

We furthermore have to specify the hyperparameter space our `Tuner` should tune over. We do this by defining a list of values with a `to_tune()` token specifying the range. Note, that hyperparameter names are prefixed with the respective operation's id.

```
# Define the parameter space to optimize over
vals = list(
  reweighing_wts.alpha = to_tune(0.75, 1),
  classif.glmnet.alpha = to_tune(0.5, 1),
  classif.glmnet.s = to_tune(1e-4, 1e-2, logscale = TRUE)
)
# Add search space to the learner
lrn$param_set$values = insert_named(lrn$param_set$values, vals)
```

Before we now train the model, we again specify a metric we aim to satisfy, here we would like the equalized odds difference to be smaller than 0.1. In this case, we set a constraint on the *equalized odds difference* comprised of the differences in true positive rate (TPR) and false positive rate (FPR):

$$\Delta_E Od = |TPR_{sex=M} - TPR_{sex=F}| + |FPR_{sex=M} - FPR_{sex=F}|$$

This can be done using the `fairness.constraint` measure.

```
metric = msr("fairness.constraint",
    performance_measure = msr("classif.acc"),
    fairness_measure = msr("fairness.eod"),
    epsilon = 0.1
)
```

We can now instantiate a new `AutoTuner` using `lrn` defined above by additionally providing arguments specifying the tuning strategy, in our case random search, the measure to optimize for as well as the number of tuning steps.

```
metric = msr("fairness.constraint",
    performance_measure = msr("classif.acc"),
    fairness_measure = msr("fairness.eod"),
    epsilon = 0.1
)
at = AutoTuner$new(
  learner = lrn, # The learner
  resampling = rsmp("holdout"), # inner resampling strategy
  measure = metric, # the metric to optimize for
  tuner = mlr3tuning::tnr("random_search"), # tuning strategy
  terminator = trm("evals", n_evals = 30)) # number of tuning steps
```

The so-constructed AutoTuner can now be used on any classification Task! Additional information regarding the AutoTuner is again available in the corresponding mlr3book chapter. In the following example, we will apply it to the Adult task and train our model. This will perform a tuning loop for the specified number of evaluations and automatically retrain the best found parameters on the full data.

```
at$train(tsk("adult_train"))
```

After training, we can look at the best models found, here ordered by our metric. Note, that our metric reports the negative constraint violation if the constraint is violated and the accuracy in case the constraint is satisfied.

```
head(at$archive$data[order(fairness.acc_equalized_odds_cstrt), 1:4])
```

We can then use the tuned model to assess our metric on the held out data:

```
prd = at$predict(tsk("adult_test"))
prd$score(c(metric, msr("classif.acc"), msr("fairness.eod")),  tsk("adult_test"))
```

So our tuned model manages to obtain an accuracy of `~0.84` while satisfying the specified constraint of $\Delta_{EOd} < 0.1$. So to summarize, we have tuned a model with the goal to optimize accuracy with respect to a constraint on a selected fairness metric using an AutoTuner.

Bibliography

A. Agrawal, J. Chen, S. Vollmer, and A. Blaom. Fairness.jl, 2020a. [p5]

A. Agrawal, F. Pfisterer, B. Bischl, J. Chen, S. Sood, S. Shah, F. Buet-Golfouse, B. A. Mateen, and S. Vollmer. Debiasing classifiers: is reality at variance with expectation?, 2020b. [p13]

U. Aivodji, H. Arai, O. Fortineau, S. Gambs, S. Hara, and A. Tapp. Fairwashing: the risk of rationalization. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 161–170. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/aivodji19a.html. [p5]

J. Angwin, J. Larson, S. Mattu, and L. Kichner. Machine bias, May 2016. URL https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing. [p1, 6]

M. Bao, A. Zhou, S. A. Zottola, B. Brubach, S. Desmarais, A. S. Horowitz, K. Lum, and S. Venkatasubramanian. It's compaslicated: The messy relationship between rai datasets and algorithmic fairness benchmarks. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021. [p3]

S. Barocas, M. Hardt, and A. Narayanan. *Fairness and Machine Learning*. fairmlbook.org, 2019. http://www.fairmlbook.org. [p1, 2]

M. Becker, M. Binder, N. Foss, L. Kotthoff, M. Lang, F. Pfisterer, N. G. Reich, J. Richter, P. Schratz, R. Sonabend, D. Pulatov, and B. Bischl. mlr3book, 09 2022. URL https://mlr3book.mlr-org.com. [p6]

R. K. E. Bellamy, K. Dey, M. Hind, S. C. Hoffman, S. Houde, K. Kannan, P. Lohia, J. Martino, S. Mehta, A. Mojsilovic, S. Nagar, K. N. Ramamurthy, J. Richards, D. Saha, P. Sattigeri, M. Singh, K. R. Varshney, and Y. Zhang. AI Fairness 360: An extensible toolkit for detecting, understanding, and mitigating unwanted algorithmic bias, Oct. 2018. URL https://arxiv.org/abs/1810.01943. [p5]

R. Berk, H. Heidari, S. Jabbari, M. Kearns, and A. Roth. Fairness in criminal justice risk assessments: The state of the art. *Sociological Methods & Research*, Aug. 2018. doi: 10.1177/0049124118782533. [p1, 2, 3, 8]

P. Biecek. Dalex: Explainers for complex predictive models in r. *Journal of Machine Learning Research*, 19 (84):1–5, 2018. URL https://jmlr.org/papers/v19/18-416.html. [p5]

M. Binder, F. Pfisterer, M. Lang, L. Schneider, L. Kotthoff, and B. Bischl. mlr3pipelines - Flexible Machine Learning Pipelines in R. *Journal of Machine Learning Research*, 22(184):1–7, 2021. URL `https://jmlr.org/papers/v22/21-0281.html`. [p6]

R. Binns. On the apparent conflict between individual and group fairness. In *Proceedings of the 2020 conference on fairness, accountability, and transparency*, pages 514–524, 2020. [p4]

S. Bird, M. Dudík, R. Edgar, B. Horn, R. Lutz, V. Milan, M. Sameki, H. Wallach, and K. Walker. Fairlearn: A toolkit for assessing and improving fairness in AI. Technical Report MSR-TR-2020-32, Microsoft, May 2020. URL `https://www.microsoft.com/en-us/research/publication/fairlearn-a-toolkit-for-assessing-and-improving-fairness-in-ai/`. [p5]

B. Bischl, O. Mersmann, H. Trautmann, and C. Weihs. Resampling methods for meta-model validation with recommendations for evolutionary computation. *Evolutionary computation*, 20(2):249–275, 2012. [p7]

B. Bischl, M. Lang, L. Kotthoff, J. Schiffner, J. Richter, E. Studerus, G. Casalicchio, and Z. M. Jones. mlr: Machine learning in r. *Journal of Machine Learning Research*, 17(170):1–5, 2016. URL `https://jmlr.org/papers/v17/15-066.html`. [p1]

J. Buolamwini and T. Gebru. Gender shades: Intersectional accuracy disparities in commercial gender classification. In *Conference on fairness, accountability and transparency*, pages 77–91. PMLR, 2018. [p2, 8]

T. Calders and S. Verwer. Three naive bayes approaches for discrimination-free classification. *Data Mining and Knowledge Discovery*, 21(2):277–292, 2010. doi: 10.1007/s10618-010-0190-x. [p8]

J. Chen. Fair lending needs explainable models for responsible recommendation. In *Proceedings of the 2nd FATREC Workshop on Responsible Recommendation*, Sept. 2018. [p1]

A. Chouldechova. Fair prediction with disparate impact: A study of bias in recidivism prediction instruments. *Big Data*, 5(2):153–163, June 2017. doi: 10.1089/big.2016.0047. [p2, 8, 10]

S. Corbett-Davies and S. Goel. The measure and mismeasure of fairness: A critical review of fair machine learning. *arXiv preprint arXiv:1808.00023*, 2018. [p3, 4, 5]

S. Corbett-Davies, E. Pierson, A. Feller, S. Goel, and A. Huq. Algorithmic decision making and the cost of fairness. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, page 797–806, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450348874. doi: 10.1145/3097983.3098095. URL `https://doi.org/10.1145/3097983.3098095`. [p1]

R. M. Dawes, D. Faust, and P. E. Meehl. Clinical versus actuarial judgment. *Science*, 243(4899): 1668–1674, 1989. [p1]

D. Dua and C. Graff. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`. [p10]

C. Dwork, M. Hardt, T. Pitassi, O. Reingold, and R. Zemel. Fairness through awareness. In *Proceedings of the 3rd innovations in theoretical computer science conference*, pages 214–226, 2012. [p3, 4, 13]

V. Eubanks. *Automating inequality: How high-tech tools profile, police, and punish the poor*. St. Martin's Press, 2018. [p1]

S. A. Friedler, C. Scheidegger, and S. Venkatasubramanian. On the (im)possibility of fairness, 2016. URL `https://arxiv.org/abs/1609.07236`. [p4]

J. Galindo and P. Tamayo. Credit risk assessment using statistical and machine learning: basic methodology and risk modeling applications. *Computational Economics*, 15(1):107–143, 2000. [p1]

T. Gebru, J. Morgenstern, B. Vecchione, J. W. Vaughan, H. Wallach, H. D. Iii, and K. Crawford. Datasheets for datasets. *Communications of the ACM*, 64(12):86–92, 2021. [p10]

M. Hardt, E. Price, and N. Srebro. Equality of opportunity in supervised learning. *Advances in neural information processing systems*, 29:3315–3323, 2016. [p5, 8, 10]

H. Heidari, M. Loi, K. P. Gummadi, and A. Krause. A moral framework for understanding fair ml through economic models of equality of opportunity. In *Proceedings of the Conference on Fairness, Accountability, and Transparency*, FAT* '19, page 181–190, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450361255. doi: 10.1145/3287560.3287584. URL `https://doi.org/10.1145/3287560.3287584`. [p4]

F. Kamiran and T. Calders. Data preprocessing techniques for classification without discrimination. *Knowledge and Information Systems*, 33(1):1–33, 2012. [p5, 10]

N. Kilbertus, M. Rojas Carulla, G. Parascandolo, M. Hardt, D. Janzing, and B. Schölkopf. Avoiding discrimination through causal reasoning. *Advances in neural information processing systems*, 30, 2017. [p4]

J. S. Kim, J. Chen, and A. Talwalkar. Fact: A diagnostic for group fairness trade-offs. In *International Conference on Machine Learning*, pages 5264–5274. PMLR, 2020. [p3]

J. Komiyama, A. Takeda, J. Honda, and H. Shimao. Nonconvex optimization for regression with fairness constraints. In *International conference on machine learning*, pages 2737–2746. PMLR, 2018. [p10]

N. Kozodoi and T. V. Varga. *fairness: Algorithmic Fairness Metrics*, 2021. URL `https://CRAN.R-project.org/package=fairness`. R package version 1.2.1. [p5]

M. Kuhn. *caret: Classification and Regression Training*, 2021. URL `https://CRAN.R-project.org/package=caret`. R package version 6.0-88. [p1]

M. Kuhn and H. Wickham. *Tidymodels: a collection of packages for modeling and machine learning using tidyverse principles.*, 2020. URL `https://www.tidymodels.org`. [p1]

M. Lang, M. Binder, J. Richter, P. Schratz, F. Pfisterer, S. Coors, Q. Au, G. Casalicchio, L. Kotthoff, and B. Bischl. mlr3: A modern object-oriented machine learning framework in R. *Journal of Open Source Software*, dec 2019. doi: 10.21105/joss.01903. URL `https://joss.theoj.org/papers/10.21105/joss.01903`. [p1, 6]

N. Mehrabi, F. Morstatter, N. Saxena, K. Lerman, and A. Galstyan. A survey on bias and fairness in machine learning. *ACM Computing Surveys (CSUR)*, 54(6):1–35, 2021. [p2, 3]

M. Mitchell, S. Wu, A. Zaldivar, P. Barnes, L. Vasserman, B. Hutchinson, E. Spitzer, I. D. Raji, and T. Gebru. Model cards for model reporting. In *Proceedings of the Conference on Fairness, Accountability, and Transparency*, FAT* '19, page 220–229, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450361255. doi: 10.1145/3287560.3287596. URL `https://doi.org/10.1145/3287560.3287596`. [p10]

S. Mitchell, E. Potash, S. Barocas, A. D'Amour, and K. Lum. Algorithmic fairness: Choices, assumptions, and definitions. *Annual Review of Statistics and Its Application*, 8:141–163, 2021. [p2]

S. U. Noble. *Algorithms of oppression*. New York University Press, 2018. [p1]

C. O'neil. *Weapons of math destruction: How big data increases inequality and threatens democracy*. Crown, 2016. [p1]

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of Machine Learning Research*, 12:2825–2830, 2011. [p5]

V. Perrone, M. Donini, M. B. Zafar, R. Schmucker, K. Kenthapadi, and C. Archambeau. Fair bayesian optimization. In *Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society*, pages 854–863, 2021. [p4, 8]

F. Pfisterer, C. Kern, S. Dandl, M. Sun, M. P. Kim, and B. Bischl. mcboost: Multi-calibration boosting for r. *Journal of Open Source Software*, 6(64):3453, 2021. doi: 10.21105/joss.03453. URL `https://doi.org/10.21105/joss.03453`. [p6]

E. Polley, E. LeDell, C. Kennedy, and M. van der Laan. *SuperLearner: Super Learner Prediction*, 2021. URL `https://CRAN.R-project.org/package=SuperLearner`. R package version 2.0-28. [p1]

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2021. URL `http://www.R-project.org/`. ISBN 3-900051-07-0. [p1]

P. Saleiro, B. Kuester, L. Hinkson, J. London, A. Stevens, A. Anisfeld, K. T. Rodolfa, and R. Ghani. Aequitas: A bias and fairness audit toolkit. *arXiv preprint arXiv:1811.05577*, 2018. [p3, 5, 10]

C. Schumann, J. S. Foster, N. Mattei, and J. P. Dickerson. We need fairness and explainability in algorithmic hiring. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '20, page 1716–1720, Richland, SC, 2020. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 9781450375184. [p1]

P. Schwöbel and P. Remmers. The long arc of fairness: Formalisations and ethical discourse. In *Proceedings of the Conference on Fairness, Accountability, and Transparency (FAccT'22)*, 2022. [p4, 5]

M. Scutari, F. Panero, and M. Proissl. Achieving fairness with a simple ridge penalty. *arXiv preprint arXiv:2105.13817*, 2021. [p10]

R. Sonabend, F. J. Király, A. Bender, B. Bischl, and M. Lang. mlr3proba: An R Package for Machine Learning in Survival Analysis. *Bioinformatics*, 02 2021. ISSN 1367-4803. doi: 10.1093/bioinformatics/btab039. [p6]

E. J. Topol. High-performance medicine: the convergence of human and artificial intelligence. *Nature Medicine*, 25(1):44–56, 2019. doi: 10.1038/s41591-018-0300-7. [p1]

M. Turner and M. McBurnett. Predictive models with explanatory concepts: a general framework for explaining machine learning credit risk models that simultaneously increases predictive power. In *Proceedings of the 15th Credit Scoring and Credit Control Conference*, 2019. URL https://crc.business-school.ed.ac.uk/wp-content/uploads/sites/55/2019/07/C12-Predictive-Models-with-Explanatory-Concepts-McBurnett.pdf. [p1]

J. Vanschoren, J. N. van Rijn, B. Bischl, and L. Torgo. OpenML. *ACM SIGKDD Explorations Newsletter*, 15(2):49–60, June 2014. doi: 10.1145/2641190.2641198. URL https://doi.org/10.1145/2641190.2641198. [p6]

S. Wachter, B. Mittelstadt, and C. Russell. Bias preservation in machine learning: the legality of fairness metrics under EU non-discrimination law. *West Virginia Law Review*, 123, 2020. [p2, 3, 4]

E. A. Watkins, M. McKenna, and J. Chen. The four-fifths rule is not disparate impact: a woeful tale of epistemic trespassing in algorithmic fairness, 2022. URL https://arxiv.org/abs/2202.09519. [p3, 4]

J. Wiśniewski and P. Biecek. fairmodels: a flexible tool for bias detection, visualization, and mitigation in binary classification models. *The R Journal*, 14:227–243, 2022. doi: 10.32614/RJ-2022-019. URL https://rj.urbanek.nz/articles/RJ-2022-019/. [p5]

Y. Xie, C. Dervieux, and E. Riederer. *R Markdown Cookbook*. Chapman and Hall/CRC, Boca Raton, Florida, 2020. URL https://bookdown.org/yihui/rmarkdown-cookbook. ISBN 9780367563837. [p10]

M. B. Zafar, I. Valera, M. Gomez Rodriguez, and K. P. Gummadi. Fairness beyond disparate treatment & disparate impact. In *Proceedings of the 26th International Conference on World Wide Web*, pages 1171–1180, Geneva, Switzerland, Apr. 2017. International World Wide Web Conferences Steering Committee. doi: 10.1145/3038912.3052660. [p5, 10]

Florian Pfisterer
LMU Munich

ORCiD: 0000-0001-8867-762X
florian.pfisterer@stat.uni-muenchen.de

Siyi Wei
University of Toronto

weisiyi2@gmail.com

Sebastian Vollmer
DFKI

University of Kaiserslautern

svollmer@stat.uni-muenchen.de

Michel Lang
LMU Munich

TU Dortmund University

ORCiD: 0000-0001-9754-0393
michel.lang@stat.uni-muenchen.de

Bernd Bischl
LMU Munich

ORCiD: 0000-0001-6002-6980
bernd.bischl@stat.uni-muenchen.de

6.6 Multi-Calibration Boosting for R

Contributed Article:
F. Pfisterer, C. Kern, S. Dandl, M. Sun, M. P. Kim, and B. Bischl. mcboost: Multi-calibration boosting for R. *Journal of Open Source Software*, 6(64):3453, 2021

Declaration of contributions FP implemented and extended the main part of the package, heavily influcenced by an unpublished python code-base written in large parts by MS. FP furthermore worked on the interaction with `mlr3` by integrating `mlr3` learners as Auditing Mechanism as well as exporting functionality to integrate `mcboost` as a 'PipeOp' into `mlr3pipelines`. CK prepared and contributed to the vignettes and co-authored the summary paper. CK contributed (very) moderately to the python code underlying this package and helped conceptionally in transitioning from the python code to the R implementation. SD reviewed the R package, provided advice on extensions and extended as well as improved vignettes, and worked towards thorough unit testing of the different methods. MS wrote the initial Python implementation of MCBoost, with feedback and oversight provided by MK. His version guided large parts of mcboost's current design and architecture. MK is a co-author of the research papers that introduced Multi-Calibration. MK oversaw the development of the initial Python implementation of MCBoost and provided additional advice and directions in the development of this R package. BB oversaw package development and provided feedback with respect to API design, implementation details, and methodology.

The Journal of Open Source Software

mcboost: Multi-Calibration Boosting for R

Florian Pfisterer[*1], **Christoph Kern**[2], **Susanne Dandl**[1], **Matthew Sun**[3], **Michael P. Kim**[4], **and Bernd Bischl**[1]

1 Ludwig Maximilian University of Munich **2** University of Mannheim **3** Princeton University **4** UC Berkeley

DOI: 10.21105/joss.03453

Software

- Review
- Repository
- Archive

Editor: Øystein Sørensen

Reviewers:

- @mwt
- @OwenWard

Submitted: 09 June 2021
Published: 06 August 2021

Summary

Given the increasing usage of automated prediction systems in the context of high-stakes decisions, a growing body of research focuses on methods for detecting and mitigating biases in algorithmic decision-making. One important framework to audit for and mitigate biases in predictions is that of Multi-Calibration, introduced by Hebert-Johnson et al. (2018). The underlying fairness notion, Multi-Calibration, promotes the idea of multi-group fairness and requires calibrated predictions not only for marginal populations, but also for subpopulations that may be defined by complex intersections of many attributes. A simpler variant of Multi-Calibration, referred to as Multi-Accuracy, requires unbiased predictions for large collections of subpopulations. Hebert-Johnson et al. (2018) proposed a boosting-style algorithm for learning multi-calibrated predictors. Kim et al. (2019) demonstrated how to turn this algorithm into a post-processing strategy to achieve multi-accuracy, demonstrating empirical effectiveness across various domains. This package provides a stable implementation of the multi-calibration algorithm, called MCBoost. In contrast to other Fair ML approaches, MCBoost does not harm the overall utility of a prediction model, but rather aims at improving calibration and accuracy for large sets of subpopulations post-training. MCBoost comes with strong theoretical guarantees, which have been explored formally in Hebert-Johnson et al. (2018), Kim et al. (2019), Dwork et al. (2019), Dwork et al. (2020) and Kim et al. (2021).

`mcboost` implements Multi-Calibration Boosting for R. `mcboost` is model agnostic and allows the user to post-process any supervised machine learning model. It accepts initial models that fit binary outcomes or continuous outcomes with predictions that are in (or scaled to) the range [0, 1]. For convenience and ease of use, `mcboost` tightly integrates with the **mlr3** (Lang et al., 2019) machine learning eco-system in R by allowing to calibrate regression or classification models fitted either within or outside of mlr3. Post-processing with `mcboost` starts with an initial prediction model that is passed on to an auditing algorithm that runs Multi-Calibration-Boosting on a labeled auditing dataset (Fig. 1). The resulting model can be used for obtaining multi-calibrated predictions. `mcboost` includes two pre-defined learners for auditing (ridge regression and decision trees), and allows to easily adjust the learner and its parameters for Multi-Calibration Boosting. Users may also specify a fixed set of subgroups, instead of a learner, on which predictions should be audited. Furthermore, `mcboost` includes utilities to guard against overfitting to the auditing dataset during post-processing.

[*] Corresponding author

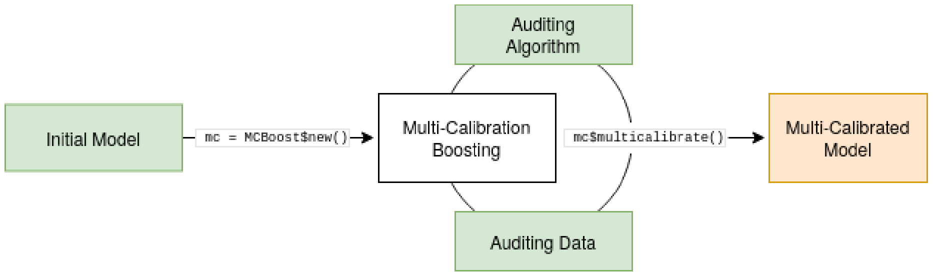


Figure 1: Fig 1. Conceptual illustration of Multi-Calibration Boosting with `mcboost`.

Statement of need

Given the ubiquitous use of machine learning models in crucial areas and growing concerns of biased predictions for minority subpopulations, Multi-Calibration Boosting should be widely accessible in the form of a free and open-source software package. Prior to the development of `mcboost`, Multi-Calibration Boosting has not been released as a software package for R.

The results in Kim et al. (2019) highlight that MCBoost can improve classification accuracy for subpopulations in various settings, including gender detection with image data, income classification with survey data and disease prediction using biomedical data. Barda, Yona, et al. (2020) show that post-processing for Multi-Calibration can greatly improve calibration metrics of two medical risk assessment models when evaluated in subpopulations defined by intersections of age, sex, ethnicity, socioeconomic status and immigration history. Barda, Riesel, et al. (2020) demonstrate that Multi-Calibration can also be used to adjust an initial classifier for a new task. They re-calibrate a baseline model for predicting the risk of severe respiratory infection with data on COVID-19 fatality rates in subpopulations, resulting in an accurate and calibrated COVID-19 mortality prediction model.

We hope that `mcboost` lets Multi-Calibration Boosting be utilized by a wide community of developers and data scientists to audit and post-process prediction models, and helps to promote fairness in machine learning and statistical estimation applications.

Acknowledgements

We thank Matthew Sun for developing an initial Python implementation of MCBoost. This work has been partially supported by the German Federal Ministry of Education and Research (BMBF) under Grant No. 01IS18036A. The authors of this work take full responsibilities for its content.

References

Barda, N., Riesel, D., Akriv, A., Levy, J., Finkel, U., Yona, G., Greenfeld, D., Sheiba, S., Somer, J., Bachmat, E., Rothblum, G., Shalit, U., Netzer, D., Balicer, R., & Dagan, N. (2020). Developing a COVID-19 mortality risk prediction model when individual-level data are not available. *Nature Communications*, *11*, 4439. https://doi.org/10.1038/s41467-020-18297-9

Barda, N., Yona, G., Rothblum, G. N., Greenland, P., Leibowitz, M., Balicer, R., Bachmat, E., & Dagan, N. (2020). Addressing bias in prediction models by improving subpopulation calibration. *Journal of the American Medical Informatics Association*, *28*(3), 549–558. https://doi.org/10.1093/jamia/ocaa283

Dwork, C., Kim, M. P., Reingold, O., Rothblum, G. N., & Yona, G. (2019). Learning from outcomes: Evidence-based rankings. *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, 106–125. https://doi.org/10.1109/FOCS.2019.00016

Dwork, C., Kim, M. P., Reingold, O., Rothblum, G. N., & Yona, G. (2020). *Outcome indistinguishability*. https://arxiv.org/abs/2011.13426

Hebert-Johnson, U., Kim, M., Reingold, O., & Rothblum, G. (2018). Multicalibration: Calibration for the (Computationally-identifiable) masses. In J. Dy & A. Krause (Eds.), *Proceedings of the 35th international conference on machine learning* (Vol. 80, pp. 1939–1948). PMLR.

Kim, M. P., Ghorbani, A., & Zou, J. (2019). Multiaccuracy: Black-box post-processing for fairness in classification. *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, 247–254. https://doi.org/10.1145/3306618.3314287

Kim, M. P., Kern, C., Goldwasser, S., Kreuter, F., & Reingold, O. (2021). *Universal generalization versus propensity scoring*. Manuscript submitted for publication.

Lang, M., Binder, M., Richter, J., Schratz, P., Pfisterer, F., Coors, S., Au, Q., Casalicchio, G., Kotthoff, L., & Bischl, B. (2019). mlr3: A modern object-oriented machine learning framework in R. *Journal of Open Source Software*. https://doi.org/10.21105/joss.01903

LIST OF FIGURES

LIST OF TABLES

EIDESSTATTLICHE VERSICHERUNG

(Siehe Promotionsordnung vom 12. Juli 2011, §8 Abs. 2 Pkt. 5)

Hiermit erkläre ich an Eides statt, dass die Dissertation von mir selbstständig, ohne unerlaubte Beihilfe angefertigt ist.

München, den 30.05.2022 Florian Pfisterer