Towards explainable automated machine learning

Julia Moosbauer

2023

Towards explainable automated machine learning

Julia Moosbauer

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

vorgelegt von
Julia Moosbauer
aus Bogen

München, den 22.02.2023

Erstgutacher/in: Prof. Dr. Bernd Bischl
Zweitgutachter/in: Prof. Dr. Avishek Anand
Drittgutachter/in: Prof. Dr. Eyke Hüllermeier
Tag der mündlichen Prüfung: 26.05.2023

Acknowledgments

First and foremost, I sincerely thank my supervisor, Prof. Dr. Bernd Bischl, for all his trust, support, and insightful guidance throughout my academic journey so far. I am grateful for the opportunity to delve into a topic that I am passionate about in a supportive and open-minded environment.

I would also like to thank Prof. Dr. Avishek Anand and Prof. Dr. Eyke Hüllermeier for their role as second and third reviewers of this thesis.

Furthermore, I want to thank my colleagues for all their support and encouragement, invaluable feedback, and stimulating discussions. Their support and collaboration were especially important during these challenging times of the pandemic. In particular, I would like to express my appreciation to Martin Binder and Julia Herbinger for their exceptional teamwork, and to Janek Thomas and Giuseppe Casalicchio, the Postdocs of the group, for their guidance and valuable advice on various projects.

I am also grateful to all my co-authors for their collaborative mindset and fruitful collaborations. My special thanks go to Marius Lindauer, who provided advice and support during our joint research projects.

Lastly, I would like to thank my family and friends for their unconditional support and encouragement at all times.

Zusammenfassung

Automatisiertes maschinelles Lernen (AutoML) zielt darauf ab, Effizienz und Zugänglichkeit von Methoden des maschinellen Lernens (ML) zu erhöhen, indem komplexe Aufgaben wie die Modell- und Hyperparameterauswahl mithilfe von Optimierungsmethoden automatisiert werden. Während AutoML-Systeme, insbesondere die Hyperparameter-Optimierung (HPO), im Vergleich zu manuellen, expertengesteuerten Prozessen Effizienzgewinne demonstriert haben, wird die Forderung nach Erklärbarkeit derzeit kaum erfüllt.

Die Arbeit verbindet die beiden aufstrebenden Bereiche AutoML und Explainable AI auf systematische Weise und trägt mit verschiedenen methodischen Ansätzen zu einer verbesserten Erklärbarkeit im Kontext von AutoML bei. Hierbei werden drei Ebenen von Erklärbarkeitsanforderungen unterschieden: Erklärbarkeit des Modells (welches von einem AutoML-System zurückgegeben wird), Erklärbarkeit der induzierenden Mechanismen oder Lernalgorithmen, und Erklärbarkeit der Mechanismen und Komponenten von AutoML-Systemen selbst.

Der erste Teil dieser Arbeit befasst sich mit der Anforderung der Erklärbarkeit von Modellen, die von AutoML-Systemen zurückgegeben werden. Der erste Beitrag gibt einen Überblick über die multikriterielle Hyperparameter-Optimierung im Allgemeinen und motiviert Interpretierbarkeit und Modell Sparsity als weitere Ziele von HPO. Im zweiten Beitrag dieser Arbeit wurde ein effizientes multikriterielles HPO-Verfahren entwickelt, das sowohl die prädiktive Performanz als auch die Modell Sparsity als ein Kriterium der Modellerklärbarkeit optimiert.

Der zweite Teil der Arbeit befasst sich mit der Erklärbarkeit von Lernalgorithmen oder induzierenden Mechanismen, mit welchen während eines AutoML-Prozesses experimentiert wird. Der dritter Beitrag stellt eine neue Methode vor, die *Partial Dependence Plots* erweitert, um Effekte von Hyperparametern der Lernalgorithmen auf die prädiktive Performance darzustellen. Die Methode berücksichtigt post-hoc eine mögliche Stichprobenverzerrung, die typischerweise in den von AutoML-Systemen erzeugten experimentellen Daten vorhanden ist. Im vierten Artikel wird eine Methode vorgestellt, mit der das Problem der Stichprobenverzerrung ex-ante behoben werden kann. Hierbei wird die Suchstrategie eines Optimierers so angepasst, dass die Genauigkeit der Erklärungen in Bezug auf die geschätzte Hyperparameter-Effekte gezielt optimiert wird.

Der letzte Teil der Arbeit beschäftigt sich mit der Erklärbarkeit von Mechanismen und Komponenten von AutoML-Systemen selbst. Ziel ist es, zu erklären, warum bestimmte Optimierer und ihre Komponenten, die in AutoML-Systemen verwendet werden, besser funktionieren als andere. Im fünften Beitrag dieser Arbeit wird eine gründliche Analyse von Multifidelity-HPO-Algorithmen vorgestellt. Die Arbeit beinhaltet ein flexibles Software-Framework für Multifidelity-HPO-Algorithmen, über welches optimiert wird. Die Ergebnisse werden mithilfe einer Ablationsanalyse analysiert. Diese Arbeit wurde durch eine effiziente Multifidelity-Benchmarking-Suite (YAHPO gym) erleichtert, die den sechsten Beitrag dieser Arbeit darstellt.

Summary

Automated machine learning (AutoML) aims at increasing the efficiency and accessibility of machine learning (ML) techniques by automating complex tasks like model and hyperparameter selection through optimization. While AutoML systems and hyperparameter optimization (HPO) frameworks have demonstrated efficiency gains compared to manual expert-driven processes, the requirement for explainability is currently hardly satisfied.

The work connects the two emerging fields of AutoML and explainable AI in a systematic manner. The thesis includes diverse methodological approaches towards increased explainability in the context of AutoML. Throughout this thesis, three levels of explainability requirements are distinguished: (1) explainability of the final model, (2) explainability of the leraning algorithm, that found the model, and (3) explainability of the AutoML system, that configured the learning algorithm, that found the model.

The first part of this thesis addresses the requirement of explainability of models returned by AutoML systems. The first contributing article reviews multi-objective hyperparameter optimization in general and motivates interpretability and sparsity as further objectives of HPO. In the second contributing article, we develop an efficient multi-objective HPO method that optimizes for both predictive performance and model sparsity as one criterion of model explainability.

The second part of the thesis deals with the explainability of the learning algorithms or inducing mechanisms experimented with during an AutoML process. A third contributing article presents a novel method that extends partial dependence plots to generate insights on the effects of hyperparameters of learning algorithms on performance. The technique accounts in a post-hoc manner for sampling bias typically present in experimental data generated by AutoML systems. The fourth contributing article introduces a method to address the problem of a sampling bias ex-ante by adapting the search strategy of an optimizer to account for the reliability of explanations in terms of hyperparameter effects.

The final part of the thesis deals with the explainability of the mechanisms and components of AutoML systems. The aim is to explain why specific optimizers and their components used within AutoML systems work better than others. A fifth contributing article presents a thorough evaluation of multi-fidelity HPO algorithms. An expressive and flexible framework of multi-fidelity HPO algorithms is introduced and automatically configured through optimization, and the outcomes are analyzed through an ablation analysis. This work has been facilitated by an efficient multi-fidelity benchmarking suite (YAHPO gym), which is the sixth contribution of this thesis.

List of abbreviations

AC	algorithm configuration
AutoML	automated machine learning
AI	artificial intelligence
BO	Bayesian optimization
CV	k-fold-cross-validation
DC	algorithm design configuration
ERM	empirical risk minimization
ES	evolutionary strategies
EA	evolutionary algorithm
GS	grid search
HPC	hyperparameter configuration
HPO	hyperparameter optimization
i.i.d.	independent and identically distributed
IML	interpretable machine learning
ML	machine learning
NAS	neural architecture search
PDP	partial dependence plot
PFI	permutation feature importance
RS	random search

Contents

Part I.

Motivation and background

1. Motivation

The growing importance of automated machine learning

"Virtually all technologies introduced [...] do something *better*, *cheaper*, *sooner* or *more reliably*", as stated in an article published back in 1998 by the OECD (1998) discussing the opportunities and risks associated with new technologies of the upcoming twenty-five years. machine learning (ML) (ML) is among those technologies that have played an essential role in the past decade. ML relies on learning algorithms to automatically identify patterns and relationships in data without the need for explicit programming. According to Shalev-Shwartz and Ben-David (2014), the input to a learning algorithm is training data, representing experience. The output is some expertise, typically in the form of a model capable of making predictions or decisions about new data inputs. Through the widespread availability of data and computational resources, ML has become one of the fastest-growing fields in computer science, with applications in various industries and domains. The past few years have brought forth remarkable applications. It was demonstrated that *better* product recommendations (Portugal et al., 2018) can be given, manufacturing processes can be *cheaper* through predictive maintenance (Zonta et al., 2020), new drug compounds can be discovered *sooner* (Vamathevan et al., 2019), and anomalies in medical data can be detected *more reliably* (Park et al., 2019; Ardila et al., 2019; Baloglu et al., 2019) with the help of ML tools.

The ML lifecycle involves several stages, including (1) understanding and defining an underlying problem, (2) collecting, managing, and making data accessible, (3) developing a model, as well as (4) deploying and maintaining it in a production environment. Successful development and deployment of ML models have been a (mostly) manual process that requires, among others, a high level of domain expertise, statistical understanding and ML knowledge, and data and software engineering skills (Baskarada and Koronios, 2017). Organizations may lack the expertise or resources to successfully develop and deploy ML models, which hinders them from realizing the full potential of ML. At the core of the ML lifecycle is the development of the model itself, which typically comprises data pre-processing, feature engineering, model selection, and hyperparameter optimization. Any of these steps requires a multitude of decisions to be made, which ultimately add to a complex space where individual choices can significantly impact final model performance. Even experts have difficulty dealing with this complex decision space efficiently despite years of experience. Users with little or no ML expertise find it challenging to overcome this barrier (Hutter et al., 2019) at all.

Holding the promise of increasing efficiency of ML development and democratizing ML as technology, automated machine learning (AutoML) has attracted great interest in the ML community in recent years (Hutter et al., 2019). AutoML aims at streamlining the ML process by formulating relevant decisions to be made throughout the ML process as configurations of a pipeline and finding the best one via optimization. Applications of AutoML in a wide range of domains demonstrate that a systematic and efficient search through configurations of a well-designed pipeline can bring remarkable ML tools into practice. Successful applications can be found in the healthcare industry (Mustafa and Azghadi, 2021) with example uses in biomarker discovery (Karaglani et al., 2020), survival prediction for diseases like COVID-19 (Ikemura et al., 2021) or cardiovascular diseases (Alaa et al., 2019). Other fields of applications include manufacturing (Krauß et al., 2020), or finance (Yang et al., 2022). AutoML has the potential to give us *better* models, provide them *cheaper* and *sooner* by reducing the need for specialized knowledge, and deliver model quality *reliably* independent of human errors.

Risks associated with the use of AutoML

Undisputably, any new technology can transform the world for the better while simultaneously creating new dangers and risks (OECD, 1998). This is arguably true for AutoML.

One type of risk stems from the fact that it is often challenging to assess and avoid unintended behavior of ML models such as unethical or unfair decisions (e.g., a bias towards African-Americans in predicting the risk of a person to recommit another crime (Mehrabi et al., 2021)), vulnerability to attacks (e.g., adversarial attacks against neural networks in cyberspace (Akhtar and Mian, 2018)), inconsistent model behavior as a response to a domain shift (e.g., real-world digital histopathology differing from training data due to differences in the data acquisition pipeline (Stacke et al., 2021)), or otherwise unintended model behavior. The black-box nature of most modern ML models makes it challenging to understand, quantify and control potential risks. Even experts are rarely able to understand and explain model decisions. Just as AutoML could bring beneficial models into practice, AutoML could potentially also speed up the deployment of models unknowingly of highly unintended behavior and destructive influence on society.

As AutoML becomes more prevalent, risks may arise due to organizations becoming increasingly dependent on AutoML tools while losing control over and understanding of the modeling process itself. A well-performing AutoML system provided via a service interface, requiring, in the most extreme, no more than uploading data and pressing a button to develop an ML model, is likely to cause organizations to lose more and more understanding and knowledge of the learning process. Knowledge and intuition about which preprocessing methods, learning algorithms, and hyperparameters work best might not be built or maintained within organizations. It might be even more challenging to relate modeling decisions to model behavior and to understand and counter suboptimal or unexpected outcomes than today. In the face of changing business needs or technological developments, this could mean a loss of flexibility and adaptability for organizations. Furthermore, understanding how ML algorithms work is the key to scientific discoveries (Hutter et al., 2014) and the advancement of the state of the art in ML.

Another consideration concerns the risks associated with a lack of understanding why specific AutoML systems are superior to others. High-performing AutoML tools may be designed as overly complex and unsustainable frameworks because crucial design decisions can not be distinguished from ineffective ones. Researchers, who aim to develop more efficient or more sustainable AutoML tools, might err in the dark because there is a lack of understanding of the key performance drivers of AutoML tools. There is at least still the (usually effortful and non-trivial) possibility for open-source tools to deduce the importance of certain design decisions based on available code, publications, and well-designed benchmarking experiments. Unfortunately, many commercial AutoML tools are offered as service tools without publicly available code, and users interact with such tools only via interfaces. Focusing on the benefits of AutoML tools, companies may become dependent on service tools that they often can neither access as open-source code nor understand. There is a danger that a few prominent players will control the availability and accessibility of AutoML, and others can neither replicate nor control them properly.

Considering all the potential offered by AutoML, it is nevertheless essential to be aware of the technology's risks and limitations and establish appropriate safeguards.

Understandability is key to evaluate and control risks

How can we broadly deploy AutoML as a technology and deliver on all of its promises – democratizing ML as a technology, improving the efficiency of the ML lifecycle, and improving real-world outcomes through more accurate models – while hedging against the associated risks?

This work advocates that a reasonable level of control of AutoML systems requires a solid understanding of the complex underlying black-box mechanisms involved in AutoML in the first place. We believe that an increased understanding of AutoML systems is required at different levels: On the level of ML models, on the level of ML algorithms or inducing mechanisms, and on the level of AutoML systems themselves. We emphasize that the type and desired level of understanding depends on whom we regard as the user of the AutoML system. We broadly distinguish *ML practitioners*, *domain experts*, and *AutoML researchers* as three main user segments, which are further described in Table 1.

	Expertise			
User group	Domain	ML	AutoML	Motivation
Domain expert	high	low	low	Access ML as technology
ML practitioners	low	high	low	Increase efficiency and performance
AutoML researcher	low	high	high	Build better AutoML tools

Table 1.1.: A rough distinction into main user groups of AutoML, motivated by Zöller et al. (2022), as well as their motivation for using AutoML. AutoML may enable domain experts to use ML as technology at all and bring in their domain knowledge. ML practitioners hope to increase efficiency of model development processes. AutoML researchers study AutoML systems to build better and and more efficient tools.

Understanding ML models The main output of running an AutoML system on a real-world dataset is an ML model. The ultimate goal is to deploy such a model in a practical setting. Ideally, developing a model with AutoML only requires pushing button, waiting for the AutoML system to complete its execution and transferring the model into a production environment. How can organizations explain or justify the model decisions to affected individuals or tell them what to change to realize a different conclusion? How can we detect that a model discriminates against certain social groups or returns otherwise unfair or unethical decisions? How can such a model be audited against certain requirements such as fairness, privacy, reliability, or causality (Molnar, 2022; Doshi-Velez and Kim, 2017)? In certain cases, there is no need to understand why a specific prediction was made. Particularly if models are deployed in low-risk environments and if the underlying problem is well understood (Molnar, 2022), a carefully performed evaluation of model performance may be sufficient to gain enough trust that a model will behave reasonably in deployment. However, suppose models have a high impact, for example, in highly regulated domains like healthcare or finance. In that case, understandability should be an indispensable requirement to ensure flaws in the model can be assessed early on before causing broader harm to individuals or society. The opinion of the European Commission (Hamon et al., 2020) and the draft of an artificial intelligence (AI) regulatory framework proposal[1] support this view. Depending on the industry, the capability of explaining and understanding model behavior should be an essential requirement for AutoML systems. It is especially domain experts who need to justify using a model given a domain for which this kind of explainability is highly relevant.

Understanding ML algorithms Traditionally, manual experimentation with different preprocessing methods, feature engineering techniques, learning algorithms, and hyperparameters to come up with a well-performing ML pipeline given a problem had been the key task for ML practitioners. Despite demonstrated superiority of automated approaches (Bergstra et al., 2011; Hutter et al., 2019; Erickson et al., 2020), many ML practitioners still tend to use HPO methods with low sampling efficiency or manual tuning (Hasebrook et al., 2022). One reason they prefer manual tuning is that it helps understand how hyperparameters influence the respective ML models. Typical questions asked are which hyperparameters are crucial to high performance and what a hyperparameter's effect on performance looks like. A study conducted by Zöller et al. (2022) also gave additional evidence that such insights would increase trust that ML practitioners would place into an AutoML system. On the other hand, a better understanding of the inducing mechanism and the effects of hyperparameters may also be beneficial for AutoML researchers as it can help maintain flexibility and adaptability in times of changing business needs or technological developments. Let us consider a hypothetical scenario where relationships in future datasets were considerably more complex than the ones we encounter today. It is plausible that the existing hypothesis spaces, covered by conventional AutoML tools, may be inadequate in modeling the intricate relationships within these datasets. However, suppose we possessed a comprehensive knowledge of the hyperparameters that impact the capacity of a hypothesis space. In that case, we could strategically extend the respective hyperparameter domains, thereby overcoming the data challenges more efficiently. Furthermore, understanding which learning algorithms and hyperparameters work best in which types of application domains could help advance AutoML research and development faster. Generating insights about the functioning of ML algorithms and the influence of hyperparameters is key to advancing scientific discoveries (Hutter et al., 2014) and could help advance the state of the art. Already today, we do not know much about how certain architecture decisions and hyperparameters in the context of deep neural networks impact model performance, and research has started to explain specific mechanisms (Santurkar et al., 2018). We advocate designing AutoML systems that do not only return a recommendation in a *take-it-or-leave-it* manner (Zöller et al., 2022) but return in addition insights gained about the underlying ML algorithms. We believe that these types of insights are most relevant to ML practitioners and AutoML researchers rather than domain experts.

Understanding AutoML systems AutoML systems typically combine a rich pipeline covering various preprocessing operators, feature engineering variants, learning algorithms, and hyperpa-

[1]European Commission (2021). Proposal for a regulation of the European parliament and council laying down harmonized rules on artificial intelligence (artificial intelligence act) and amending certain union legislative acts. COM/2021/206 final. `https://artificialintelligenceact.eu/the-act/`.

rameter choices with a powerful optimizer that searches over the respective search space given a dataset. The potential of a robust, efficient, and safe implementation of an AutoML system is vast: Turning any task for which data is available into a prediction machine. There is commercial interest behind: big tech players like Google or Amazon put considerable resources into developing their AutoML systems. There are also increasingly many non-commercial open-source AutoML frameworks developed by the community (Zöller and Huber, 2021). Which of these AutoML systems work best? Why is one tool superior to the others? Do some AutoML systems only work well on certain tasks? Does an algorithmic difference impact performance, or is it related to a superior implementation? Are there ineffective components of AutoML implementations (e.g., too large search pipelines, optimizers with too much overhead) that do not impact performance, making it more complex than it has to be? It should be of interest to society and particularly the research community to maintain a solid understanding of AutoML as a technology, its drivers, and its limitations to avoid misuse of and maintain control over such a powerful technology. Moreover, understanding an AutoML system's workings will help us advance the state of the art and build more sustainable AutoML systems instead of sticking to unexplained design choices from prior work. Even though first broad initiatives to compare AutoML tools (Gijsbers et al., 2022) exist, we still need to generate a more profound knowledge about what AutoML systems work best and why and for which types of applications, keeping in mind that the *no-free-lunch theorem* (Wolpert and Macready, 1997) also applies to AutoML.

Explainability in AutoML

AutoML has made significant technical advances and performs solidly in many areas today (Zöller and Huber, 2021), but has yet to achieve widespread adoption, particularly in non-tech domains (Serband et al., 2020). We are in an area of tension between potential overregulation and underregulation of such automated systems, between the non-utilization of a promising technology and exposure to risks we are incapable of comprehending.

We believe that it is important to prioritize the design of AutoML systems, which give users a level of understanding to allow them to interact safely, robustly, and ethically with AutoML in the long-term and help us build trust into those systems and making informed decisions in the short term.

This work contributes to more understandable AutoML at the levels introduced: models, learning algorithms, and AutoML systems. At all these three levels, we face a similar challenge: the object we wish to comprehend is a complex, often black-box object which is by nature difficult to understand for humans. The work connects the two emerging fields of AutoML and explainable AI. While the field of explainable AI has so far mainly focused on explaining model behavior, we expand respective concepts, discuss them in relation to AutoML and show methodological ways of building explainability into AutoML systems.

Section 2 will provide background on the topic of AutoML and explainability: First, a formal definition of ML, HPO, AutoML, and algorithm configuration (AC) is given in Section 2.1 and looked at from a multi-level inference perspective in Section 2.2. An overview of state of the art HPO algorithms and AutoML frameworks is given in Section 2.3, and the general field explainability as a subfield of ML is introduced in Section 2.4. Section 3 provides a systematic view on the field of explainable AutoML based on the different levels of inference (model, inducer, and

AutoML system level) and assesses the current state of the art in this field. Parts 4, 5, and 6 are the core of this thesis and present methodological advancements to the current state of the art. Finally, we discuss potential future works to advance the field further and ultimately create more explainable AutoML systems.

2. Background

2.1. Definitions

In this section, we introduce the concepts of supervised ML, HPO, AutoML, and AC, which represent elemental foundations of this work, in a formalized manner.

2.1.1. Supervised machine learning

The goal of supervised ML is to fit a model $f : \mathcal{X} \to \mathbb{R}^g, g \in \mathbb{N}$, based on a (training) dataset $\mathcal{D}$ with observations $\left(\mathbf{x}^{(i)}, y^{(i)}\right) \in \mathcal{X} \times \mathcal{Y}$, $i \in \{1, 2, ..., n\}$, so that it generalizes well on new observations $(\mathbf{x}, y)$ drawn independently from an unknown, underlying distribution $\mathbb{P}_{xy}$ (Bischl et al., 2021). The observations in $\mathcal{D}$ are assumed to be independent and identically distributed samples drawn $\mathbb{P}_{xy}$. The concept of the (true) *generalization error* captures a model's ability to make accurate predictions on unseen data

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

where the loss function $L : \mathcal{Y} \times \mathbb{R}^g \to \mathbb{R}_0^+$ is a measure of deviation between true label y and the prediction $f(\mathbf{x})$. Many different loss functions do exist; common examples for loss functions are L2 loss $L(y, f(\mathbf{x})) = (f(\mathbf{x}) - y)^2$ or L1 loss $L(y, f(\mathbf{x})) = |f(\mathbf{x}) - y|$ for regression tasks ($\mathcal{Y} \subseteq \mathbb{R}$), and 0-1-loss $L(y, f(\mathbf{x})) = \mathbf{1}_{[y \cdot f(\mathbf{x}) > 0]}$ or Bernoulli loss $L(y, f(\mathbf{x})) = -y \cdot f(\mathbf{x}) + \log(1 + \exp(f(\mathbf{x})))$ for binary classification tasks ($y \in \{0, 1\}$). There are also appropriately designed loss functions for other types of tasks, such as for multi-label classification (Díez et al., 2015) or survival prediction (Lee et al., 2022).

The process of fitting a model f from a dataset $\mathcal{D}$ is also referred to as *model training* or *model fitting* (Bischl et al., 2021). We define model training formally via an *inducer* function $\mathcal{I}$, also referred to as ML learner. Configured through a vector of hyperparameters $\boldsymbol{\lambda} \in \Lambda$, the inducer $\mathcal{I}$ maps a dataset $\mathcal{D}$ to a model $f : \mathcal{X} \to \mathbb{R}^g$ in the hypothesis space $\mathcal{H}$, which is the set of all functions f (hypotheses) that the dataset could hypothetically be mapped to by the inducer $\mathcal{I}$. Throughout this thesis, we assume that $\mathcal{H}$ is defined as a parameterized family of functions $\mathcal{H} := \{f : \mathcal{X} \to \mathcal{Y} \mid f(\mathbf{x}) = f_{\boldsymbol{\theta}}(\mathbf{x}) \text{ for } \boldsymbol{\theta} \in \Theta\}$ and consequently define

$$\begin{aligned} \mathcal{I} : \mathbb{D} \times \Lambda &\to \Theta \\ (\mathcal{D}, \boldsymbol{\lambda}) &\mapsto \boldsymbol{\theta}, \end{aligned} \tag{2.2}$$

where $\mathbb{D}$ is the set of all datasets an inducer can take as input.

Many modern ML algorithms rely on empirical risk minimization (ERM), i.e., the minimization of the empirical risk $\mathcal{R}_{\text{emp}}(\boldsymbol{\theta}) := \sum_{(\mathbf{x},y)\in\mathcal{D}} L\left(y, f_{\boldsymbol{\theta}}(\mathbf{x})\right)$ over Θ. It should be noted that $\mathcal{R}_{\text{emp}}$ is only a proxy of the true generalization error $\mathcal{R}$, which is the actual quantity of interest. In particular, efficient optimization over a hypothesis space that is too rich in relation to the complexity of a dataset can result in a model with a generalization error that is considerably higher than the empirical risk. This is a common phenomenon that is referred to as *overfitting* (Bischl et al., 2021).

The capability of $\mathcal{I}$ to return a well-generalizing model given a dataset of size n with hyperparameter configuration (HPC) $\boldsymbol{\lambda}$, measured via a observation-wise loss[1] function L, is defined as

$$\text{GE}(\mathcal{I}, \boldsymbol{\lambda}, n, L) := \mathbb{E}_{\mathcal{D}\sim\mathbb{P}_{xy}^n,(\mathbf{x},y)\sim\mathbb{P}_{xy}}\left[L(y, \mathcal{I}(\mathcal{D}, \boldsymbol{\lambda})(\mathbf{x}))\right], \tag{2.3}$$

where the expected value is taken over a training dataset $\mathcal{D}$ and test sample $(\mathbf{x}, y)$ sampled independent and identically distributed (i.i.d.) from $\mathbb{P}_{xy}$.

Usually, $\mathbb{P}_{xy}$ is unknown in practice and the generalization error GE must be estimated from a single, given dataset $\mathcal{D}$. Therefore, GE is estimated through holdout

$$\widehat{\text{GE}}\left(\mathcal{I}, \boldsymbol{\lambda}, \mathcal{D}, (J_{\text{train}}, J_{\text{test}}), L\right) = \frac{1}{n_{\text{test}}} \sum_{(\mathbf{x},y)\in\mathcal{D}_{J_{\text{test}}}} L\left(y, \mathcal{I}\left(\mathcal{D}_{J_{\text{train}}} \boldsymbol{\lambda}\right)(\mathbf{x})\right), \tag{2.4}$$

where $\mathcal{D}_{\text{train}} := \left\{\left(\mathbf{x}^{(i)}, y^{(i)}\right) \in \mathcal{D} \mid i \in J_{\text{train}}\right\}$ and $\mathcal{D}_{\text{test}} := \left\{\left(\mathbf{x}^{(i)}, y^{(i)}\right) \in \mathcal{D} \mid i \in J_{\text{test}}\right\}$ represent a partition of the dataset $\mathcal{D}$ of size n_{train} and n_{test} respectively, indexed through index vectors $J_{\text{train}} \in \{1, 2, ..., n\}^{n_{\text{train}}}$ and $J_{\text{test}} \in \{1, 2, ..., n\}^{n_{\text{test}}}$. To reduce the variance of the estimator, resampling strategies can be applied which essentially compute Estimate (2.4) repeatedly for different train-test-splits $\mathcal{J} = \left\{\left(J_{\text{train}}^{(b)}, J_{\text{test}}^{(b)}\right)\right\}_{b=1,...,B}$ and aggregate[2] those estimates

$$\widehat{\text{GE}}\left(\mathcal{I}, \boldsymbol{\lambda}, \mathcal{D}, \mathcal{J}, L\right) = \frac{1}{|\mathcal{J}|} \sum_{(J_{\text{train}}, J_{\text{test}})\in\mathcal{J}} \widehat{\text{GE}}\left(\mathcal{I}, \boldsymbol{\lambda}, \mathcal{D}, (J_{\text{train}}, J_{\text{test}}), L\right). \tag{2.5}$$

A popular variant is k-fold-cross-validation (CV), where the available data $\mathcal{D}$ is split into k partitions of same size where training sets are disjoint from each other.

Common learning algorithms are linear regression or regression trees for regression, and logistic regression or decision trees for classification tasks. Ensembling-based methods like bagging

[1] For the sake of simplicity, we use the above definition; if a non-pointwise performance measure ρ is used (such as the area-under-the-curve, AUC), the generalization error is defined via $\text{GE}(\mathcal{I}, \boldsymbol{\lambda}, n_{\text{train}}, \rho) := \mathbb{E}_{\mathcal{D},\mathcal{D}_{\text{test}}\sim\mathbb{P}_{xy}}\left[\rho\left(\boldsymbol{y}_{\text{test}}, \boldsymbol{F}_{\mathcal{D}_{\text{test}},\mathcal{I}(\mathcal{D},\boldsymbol{\lambda})}\right)\right]$, with $\boldsymbol{F}_{\mathcal{D}_{\text{test}},\mathcal{I}(\mathcal{D},\boldsymbol{\lambda})}$ the matrix of predictions of the model $\mathcal{I}(\mathcal{D}, \boldsymbol{\lambda})$ on $\mathcal{D}_{\text{test}}$ and $\boldsymbol{y}_{\text{test}}$ is the vector of labels of test dataset $\mathcal{D}_{\text{test}}$. For more information, see also Bischl et al. (2021).

[2] Here, we show the formula for aggregation via the mean function. However, also other aggregation functions could be used (Bischl et al., 2021).

(Breiman, 1996), e.g., random forests (Breiman, 2001), or boosting (Friedman, 2001), e.g., xgboost (Chen and Guestrin, 2016), have become popular in recent years. Neural networks have gained a lot of attention recently, particularly due to their ability to handle complex data types such as image and text data. Secondary properties beyond the capability to generalize – such as model complexity (Molnar et al., 2019), robustness (Scher and Trügler, 2022), and fairness (Corbett-Davies and Goel, 2018) – are often associated with the respective model class, even though it may be challenging to quantify these properties.

2.1.2. Hyperparameter optimization and automated machine learning

The capability of the trained model to generalize well on unseen data typically depends on the chosen HPC $\boldsymbol{\lambda}$ in a non-trivial and subtle way (Bischl et al., 2021). Identifying well-performing HPCs is a complex task requiring expert experience, unwritten rules of thumb, or sometimes brute-force search (Snoek et al., 2012). Therefore, one tries to solve the decision problem by formalizing it as an optimization problem, the HPO problem, and running an efficient optimizer over it to reduce human effort and improve generalization performance (Feurer and Hutter, 2019).

The goal of (automated) HPO is to identify a HPC $\boldsymbol{\lambda} \in \tilde{\Lambda}$ leading to an inducer $\mathcal{I}$ that generalizes well on a given dataset $\mathcal{D}$. We call $\tilde{\Lambda} =\subset \tilde{\Lambda}_1 \times \tilde{\Lambda}_2 \times \cdots \times \tilde{\Lambda}_l \subseteq \Lambda$ the *search space*, a typically box-constrained sub-space of Λ which contains all hyperparameters and respective ranges that are searched over during optimization. The general HPO problem is formally defined as

$$\boldsymbol{\lambda}^* \in \arg\min_{\boldsymbol{\lambda}\in\tilde{\Lambda}} c(\boldsymbol{\lambda}) \tag{2.6}$$

where $\boldsymbol{\lambda}^*$ denotes the theoretical optimum, and $c(\boldsymbol{\lambda})$ is a shorthand for the estimated generalization error $\widehat{\mathrm{GE}}\left(\mathcal{I}, \boldsymbol{\lambda}, \mathcal{D}, \mathcal{J}, L\right)$, taking $\mathcal{I}, \mathcal{D}, \mathcal{J}$ and L as fixed. Note that HPO problem can be extended by additional criteria, such as different metrics for the generalization error, or metrics indicative of fairness, robustness, sparsity, model complexity, or model interpretability. We then term the problem as *multi-objective* HPO problem. Additional criteria can be either introduced as constraints (*constrained HPO problem*) or as additional objectives $c : \Lambda \to \mathbb{R}^m, m \geq 1$, (*multi-objective HPO problem*).

An HPO algorithm $\mathcal{A}$, defined as

$$\begin{aligned} \mathcal{A} : \mathbb{D} \times \Gamma &\to \Lambda \\ (\mathcal{D}, \boldsymbol{\gamma}) &\mapsto \boldsymbol{\lambda}, \end{aligned} \tag{2.7}$$

takes a dataset $\mathcal{D}$ as input and returns a HPC which is then used to train a final model on the data $\mathcal{D}$. An HPO algorithm may be configured through a design configuration $\boldsymbol{\gamma} \in \Gamma$. We also regard the chosen inducer $\mathcal{I}$ and the specification of the search space $\tilde{\Lambda}$ as part of the input $\boldsymbol{\gamma}$ to the algorithm $\mathcal{A}$. In classical HPO, the choice of the inducer $\mathcal{I}$ and search space $\tilde{\Lambda}$ are typically left to the user of an HPO framework. In contrast, AutoML tools come with a definition of the inducer $\mathcal{I}$ (typically, a pipeline inducer as illustrated in Figure 2.1) as well as a defined search space $\tilde{\Lambda}$, taking those decisions from the user.

The HPO problem (2.6) is generally classified as an *expensive black-box* optimization problem (Feurer and Hutter, 2019), as typically no analytic information about $c(\boldsymbol{\lambda})$ is available, and an evaluation of $c(\boldsymbol{\lambda})$ is expensive as it involves training a model to estimate the generalization error of a model trained with HPC $\boldsymbol{\lambda}$ on the dataset $\mathcal{D}$. HPO algorithms are therefore typically required to be sample-efficient optimizers (i.e., requiring a low number of evaluations of c) that do not rely on analytical (e.g., gradient) information. They are also required to be able to operate efficiently on mixed[3] and possibly hierarchical[4] search spaces which are common to HPO problems.

Deriving a model from raw data typically requires a series of sequential decisions to be taken that may influence generalization performance. The generalization of the classical HPO problem to also cover model selection commonly referred to as the *Combined Algorithm Selection and Hyperparameter* (CASH) (Thornton et al., 2013) problem. We can extend our understanding of an inducer $\mathcal{I}$ from only performing model training to also performing model selection by introducing an additional parent-level hyperparameter that selects between different learning algorithms. Optimization of hyperparameters of the different learning algorithms is made conditional on the learning algorithm selected (Thornton et al., 2013). Beyond that, a whole pipeline of nodes, each of which may represent preprocessing operations, a learner, or postprocessing operations, can be considered an inducer $\mathcal{I}$ (Bischl et al., 2021). Similar to how model selection can be formalized by introducing a categorical parent-level hyperparameter which represents model choice, a branching between different preprocessing or postprocessing operators can be represented by an additional categorical hyperparameter. This way, our definition of the HPO problem (2.6) covers both HPO in the narrow sense (i.e., the optimization of hyperparameters of a learning algorithm) and the broader sense (i.e., the optimization of configuration of an. ML pipeline). Figure 2.1 shows a simple example of an ML pipeline along with the respective (hierarchical) search space spanned up by this pipeline (see Table 2.1).

Neural architecture search (NAS) is a subfield of AutoML which focuses on automating architecture engineering of deep neural networks (Elsken et al., 2019), intending to find the best possible architecture for a particular task. Through formulating a suitable search space over architectures, NAS can be solved through HPO. Another subfield of AutoML is *meta-learning* (Vanschoren, 2018), which refers to the science of systematically observing how different ML approaches perform on a wide range of learning tasks and exploiting this knowledge to learn new tasks faster than otherwise possible. This thesis focuses on HPO-based AutoML, and mostly on AutoML for tabular data tasks. We discuss the relevancy of this work for NAS and meta-learning, as well as potential future research directions in Part 7.

The key principle of AutoML is the definition of such a pipeline and running an efficient optimizer over the configurations of this pipeline (Bischl et al., 2021).

2.1.3. Algorithm configuration for AutoML systems

HPO aims to find a HPC $\boldsymbol{\lambda}$ of the inducer $\mathcal{I}$ which leads to a well-generalizing model on a *specific* dataset $\mathcal{D}$. For HPO and AutoML frameworks to be as useful as possible in practice, they should be designed to perform well on a broad range of different tasks (i.e., datasets from different data-generating processes). There is a multitude of design choices to be made by researchers or

[3]Containing continuous, discrete, and/or categorical HPs

[4]Containing conditionality, i.e., there is at least one HP λ_i is only active when another HP λ_j is an element of a given subset of Λ_j and inactive otherwise, i.e., not affecting the resulting learner (Thornton et al., 2013).

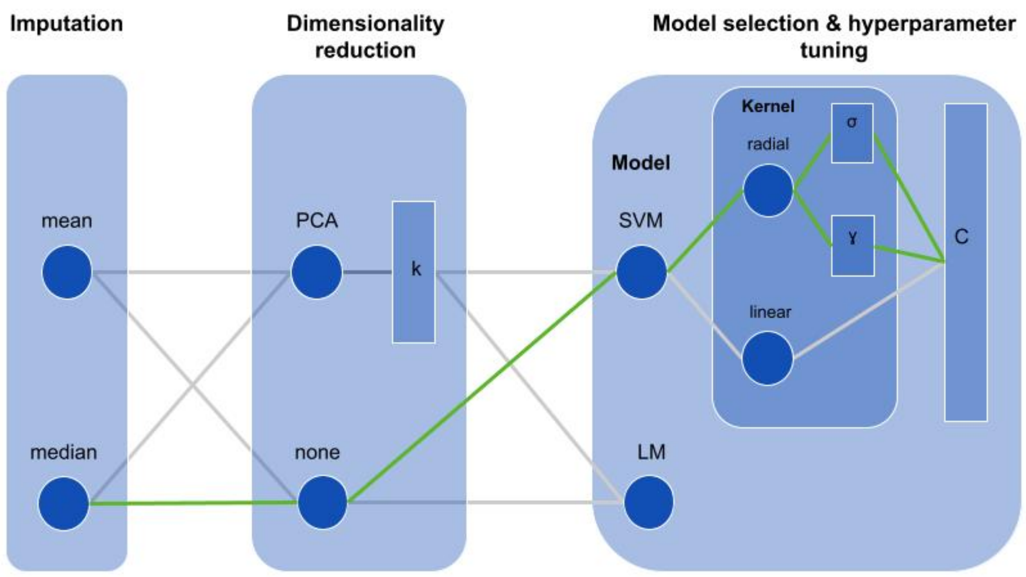


Figure 2.1.: Example illustration of a reasonably simple ML pipeline including imputation for missing values (mean, median), dimensionality reduction (PCA, no dimensionality reduction), two models (support vector machine and linear model), and hyperparameters of the SVM. For PCA, there is a hierarchical dependency on the integer parameter k (hidden dimension). The SVM model has a hierarchical dependence on the kernel type (radial vs. linear). The radial kernel has another hierarchical dependency on two numeric hyperparameters. The configuration space spanned up by this pipeline is mixed-hierarchical.

developers of AutoML systems: choosing the optimizer (e.g., Bayesian optimization (Snoek et al., 2012) or Hyperband (Li et al., 2017)), configuring the optimizer (e.g., the size and type of the initial design, the acquisition function, the surrogate model and further hyperparameters of the optimizer (Lindauer et al., 2019)), as well as the specific design of the pipeline to be tuned over. While the pipeline design ultimately influences the richness of the hypothesis space realizable within an AutoML system, the choice and design of the optimizer affect how efficiently the best configuration $\boldsymbol{\lambda}$ (and thus also the best model in the hypothesis space) can be found. All these design choices span a – typically mixed-hierarchical – space of design choices, leading to differences in performance of AutoML systems (Gijsbers et al., 2022).

Expressing all these decisions as design parameters $\boldsymbol{\gamma} \in \Gamma$, we formulate the design of an AutoML framework as an optimization problem. Let $\mathcal{A}$ be the HPO algorithm, configured via a design configuration parameter $\boldsymbol{\gamma}$, which takes a dataset $\mathcal{D}$ and returns a HPC $\boldsymbol{\lambda} \in \Lambda$. The AC problem for designing an HPO algorithm (or, more broadly, an AutoML system) that works well across a range of problem instances (i.e., different tasks) is defined as

$$\boldsymbol{\gamma}^* \in \arg\min_{\boldsymbol{\gamma} \in \Gamma} \mathbb{E}_{\omega \sim \mathbb{P}_\Omega} \left[\zeta(\mathcal{A}(\omega, \boldsymbol{\gamma})) \right], \tag{2.8}$$

Hyperparameter		Type	Range	Dependency
Imputation	λ_{imp}	categorical	{mean, median}	–
Dimensionality reduction	λ_{dim}	categorical	{PCA, none}	–
Hidden dimension	$\lambda_{\text{dim, k}}$	integer	$\{1, 2, ..., k_{\max}\}$	$\lambda_{\text{dim}} = \text{PCA}$
Model	λ_{model}	categorical	{SVM, LM}	–
Kernel type	λ_{kernel}	categorical	{radial, linear}	$\lambda_{\text{model}} = \text{SVM}$
Kernel lengthscale	λ_{γ}	numeric	$[\gamma_{\min}, \gamma_{\max}] \subseteq \mathbb{R}_0^+$	$\lambda_{\text{kernel}} = \text{radial}$
Regularization constant	λ_C	numeric	$[C_{\min}, C_{\max}] \subseteq \mathbb{R}_0^+$	$\lambda_{\text{model}} = \text{SVM}$

Table 2.1.: Search space spanned up by the hyperparameters of the example pipeline shown in Figure 2.1. The table shows the type of the hyperparameters (categorical, integer, and numeric) and shows respective ranges. The hyperparameter k, representing the hidden dimension, is only active the principal component analysis (PCA) is chosen as dimensionality reduction method. The SVM hyperparameters kernel type $\boldsymbol{\lambda}_{\text{kernel}}$ and regularization constant $\boldsymbol{\lambda}_C$ are only active if the SVM is chosen as model. If the radial basis function kernel $k(\mathbf{x}, \mathbf{x}') = \exp\left(-\gamma\|\mathbf{x} - \mathbf{x}'\|^2\right)$ is chosen, the hyperparameter γ becomes active.

where ζ is a cost metric (a metric for generalization performance and/or secondary metrics for fairness, robustness, complexity of a respective outcome) and Ω is the space of problem instances. Since the distribution over problem instances $\mathbb{P}_\Omega$ is usually unknown, we approximate the objective function through a mean estimate

$$\hat{\boldsymbol{\gamma}} \in \arg\min_{\gamma \in \Gamma} \kappa(\gamma), \tag{2.9}$$

with $\kappa(\gamma) := \frac{1}{K}\sum_{\omega \in \mathcal{W}} \zeta(\mathcal{A}(\omega, \boldsymbol{\gamma}))$ and $\mathcal{W} = \{\omega^{(1)}, \omega^{(2)}, ..., \omega^{(K)}\} \subseteq \Omega^K$ a set of $K \in \mathbb{N}$ problem instances, for example taken from benchmarking platforms like OpenML (Vanschoren et al., 2013).

We define a solver to the AC problem (2.9) as

$$\begin{aligned} \mathcal{C} : \bigcup_{k \in \mathbb{N}} \Omega^k &\to \Gamma \\ \mathcal{W} &\mapsto \boldsymbol{\gamma}, \end{aligned} \tag{2.10}$$

where Ω is the set of all problem instances.

The key difference between the AC problem and the HPO problem is that the HPO problem optimizes for a single problem instance, while the AC problem optimizes across multiple problem instances. An HPO algorithm should provide a configuration $\boldsymbol{\lambda}$ for which the inducer $\mathcal{I}$ returns a well-generalizing model for the given dataset $\mathcal{D}$. A solver to the AC problem for HPO should return a configuration γ, which results in an HPO algorithm that will perform well on problem instances sampled from $\mathbb{P}_\Omega$.

The AC problem in the context of HPO and AutoML also constitutes an expensive black-box problem: The analytical from of $\kappa(\gamma)$ is not known, and an evaluation of κ requires running $\mathcal{A}$ multiple times, which in turn requires evaluating c several times.

2.2. A multi-level inference perspective on AutoML

The term *inference* refers to the process of reasoning or drawing conclusions based on available evidence or information. In this section, we point out how the three concepts introduced in Section 2.1 can be seen as types of inference and how they are intertwined in terms of a multilevel inference perspective.

First-level inference The primary goal in ML is to identify a model f, as specified by its model parameters $\boldsymbol{\theta}$, from training data $\mathcal{D}$ so that it generalizes well on new data. In other words, ML aims to *infer* model parameters $\hat{\boldsymbol{\theta}}$ from $\mathcal{D}$

$$\hat{\boldsymbol{\theta}} \quad = \quad \mathcal{I}(\mathcal{D}, \boldsymbol{\lambda}). \tag{2.11}$$

Usually, the inducer $\mathcal{I}$, parameterized by $\boldsymbol{\lambda}$, internally solves an optimization problem

$$\hat{\boldsymbol{\theta}} \quad \in \quad \arg\min\nolimits_{\boldsymbol{\theta} \in \Theta} \mathcal{R}_{\text{emp}}(\boldsymbol{\theta}) \tag{2.12}$$

We refer to Equation (2.11) as *first-level inference*. The input (a dataset $\mathcal{D}$) is mapped to an output (model parameters $\hat{\boldsymbol{\theta}}$ specifying a model f) through an inducer function $\mathcal{I}$ that usually relies on minimizing the *first-level objective* (2.12). The inducer is commonly parameterized by hyperparameters $\boldsymbol{\lambda} \in \Lambda$, which may influence the effectiveness and efficiency of the inducer $\mathcal{I}$ to return a well-generalizing model f.

Second-level inference The goal of HPO is to find a configuration $\boldsymbol{\lambda}$ to set the stage for successful first-level inference. Hyperparameters $\boldsymbol{\lambda}$ usually must be set before first-level inference (2.11). The process of (automatically) finding a good HPC prior to model training can be understood as another type of inference: inferring a HPC $\hat{\boldsymbol{\lambda}}$ from data $\mathcal{D}$

$$\hat{\boldsymbol{\lambda}} = \mathcal{A}(\mathcal{D}, \boldsymbol{\gamma}). \tag{2.13}$$

The HPO algorithm $\mathcal{A}$, parameterized by $\boldsymbol{\gamma}$, internally solves an optimization problem

$$\hat{\boldsymbol{\lambda}} \quad \in \quad \arg\min\nolimits_{\boldsymbol{\lambda} \in \Lambda} c(\boldsymbol{\lambda}) \tag{2.14}$$

We refer to Equation (2.13) as *second-level inference*. The input (a dataset $\mathcal{D}$) is mapped to an output (hyperparameters $\boldsymbol{\lambda}$) through a function $\mathcal{A}$ that usually relies on minimizing the *second-level objective* (2.14). The HPO algorithm $\mathcal{A}$ is again parameterized by a (second-level) hyperparameter $\boldsymbol{\gamma} \in \Gamma$, which may influence the effectiveness and efficiency of the HPO algorithm $\mathcal{A}$ to return a well-generalizing inducer $\mathcal{I}$.

Note that we use the terms first- and second-level[5] inference in line with Guyon et al. (2010), who coined the term *multi-level inference* as a learning problem organized into a hierarchy of learning problems. The two levels show parallels: An inducing mechanism ($\mathcal{I}$ and $\mathcal{A}$, respectively), parameterized by a hyperparameter ($\boldsymbol{\lambda}$ and $\boldsymbol{\gamma}$, respectively), is used to infer a parameter ($\hat{\boldsymbol{\theta}}$ and $\hat{\boldsymbol{\lambda}}$, respectively) from data, by internally solving an optimization problem (objectives (2.12) and (2.14), respectively). We emphasize the nested character of the two optimization problems: A single evaluation of the second-level objective for a HPC $\boldsymbol{\lambda}$ requires performing first-level inference repeatedly. The outcome of the second-level inference, a HPC $\hat{\boldsymbol{\lambda}}$, sets the frame for where first-level optimization is taking place and is thus an often success-critical input to first-level inference.

Third-level inference The goal of AC for HPO is to find a configuration $\boldsymbol{\gamma}$ that will set the stage for successful second-level inference. The configuration of an HPO algorithm (or, more generally, an AutoML system) has to be set prior to the use of such an algorithm or framework in a way such that it ideally performs well on a broad set of problem instances. The process of (automatically) designing a well-configured HPO algorithm can be again understood as a type of inference: inferring a configuration $\hat{\boldsymbol{\gamma}}$ from data $\mathcal{W}$ (a set of different problem instances)

$$\hat{\boldsymbol{\gamma}} = \mathcal{C}\left(\mathcal{W}\right). \tag{2.15}$$

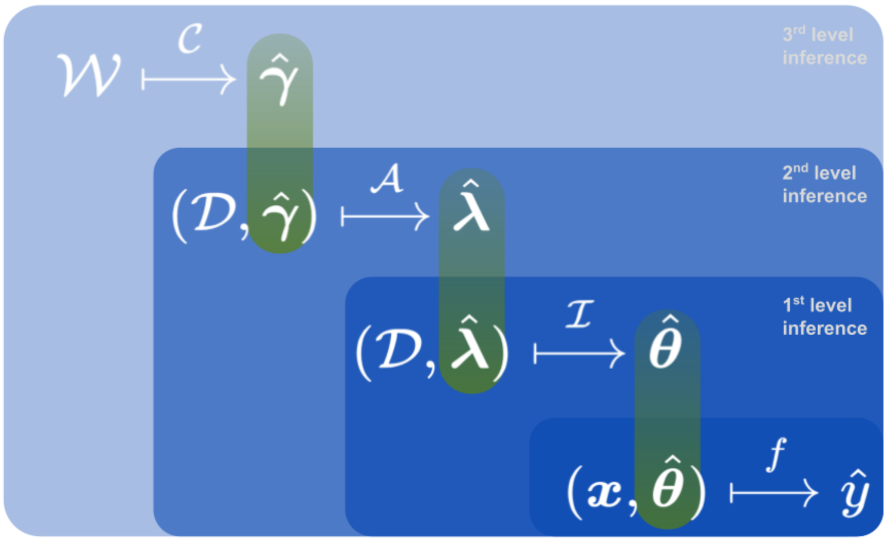
An (HPO) algorithm configurator $\mathcal{C}$ internally solves an optimization problem

$$\hat{\boldsymbol{\gamma}} \in \arg\min_{\boldsymbol{\gamma}\in\Gamma} \kappa(\boldsymbol{\gamma}) \tag{2.16}$$

We refer to Equation (2.15) as *third-level inference*. The input (a set of problem instances $\mathcal{W}$) is mapped to an output (second-level hyperparameters $\boldsymbol{\gamma}$) through a function $\mathcal{C}$ that relies on minimizing the *third-level objective* (2.14).

As suggested by Guyon et al. (2010), a multi-level perspective can introduce further inference levels. Here, we introduced a third level, setting the stage for second-level inference. We again emphasize that the different levels of inference are nested within each other: An evaluation of the third-level objective requires multiple evaluations of the second-level objective, which again requires multiple evaluations of the first-level objective. Figure 2.2 schematically visualizes how the different problems are intertwined.

[5]Note that (Franceschi et al., 2018) refer to the first and second level objective as inner and outer objective.



$\mathcal{W} \xrightarrow{\mathcal{C}} \hat{\gamma}$
3rd level inference
$(\mathcal{D}, \hat{\gamma}) \xrightarrow{\mathcal{A}} \hat{\boldsymbol{\lambda}}$
2nd level inference
$(\mathcal{D}, \hat{\boldsymbol{\lambda}}) \xrightarrow{\mathcal{I}} \hat{\boldsymbol{\theta}}$
1st level inference
$(\boldsymbol{x}, \hat{\boldsymbol{\theta}}) \xrightarrow{f} \hat{y}$

Figure 2.2.: This schematic representation illustrates the parallels and nested characters between the different levels of inference. For each level, there is an inducing mechanisms ($\mathcal{I}$, $\mathcal{A}$, and $\mathcal{C}$, respectively), configured by a hyperparameter ($\boldsymbol{\lambda}$, and $\boldsymbol{\gamma}$, respectively) which infers a lower-level parameter ($\hat{\boldsymbol{\theta}}, \hat{\boldsymbol{\lambda}}$ and $\hat{\boldsymbol{\gamma}}$, respectively). Note that we do not introduce a further hyperparameter to configure $\mathcal{C}$ even though also $\mathcal{C}$ can be considered configured by some higher-level hyperparameter.

2.3. Algorithms and frameworks

This section gives an overview of common HPO algorithms and AutoML frameworks. Note that this overview is not exhaustive and will not go into the details of individual algorithms and frameworks. In Chapter 3, we will provide an in-depth view of HPO and AutoML frameworks from an explainability perspective.

2.3.1. HPO algorithms

An HPO algorithm $\mathcal{A}$ is designed to solve the HPO problem (2.6). Bischl et al. (2021) characterize HPO algorithms by how the trade-offs exploration-vs-exploitation and the inference-vs-search trade-offs are handled; further relevant characteristics of HPO algorithms are parallelizability, noise handling, multi-fidelity, and how they can handle search space complexity.

Two of the simplest algorithms are grid search (GS) and random search (RS). GS evaluates every grid point of a discretized search space $\tilde{\Lambda} \subseteq \Lambda$ exhaustively. RS evaluates HPCs randomly drawn from an (e.g., uniform) distribution over $\tilde{\Lambda}$. Still today, RS is a ubiquitous algorithm because it is simple and works remarkably well in the context of HPO due to the often low effective dimensionality of the problems occurring there (Bergstra and Bengio, 2012).

Over the past several years, the use of Bayesian optimization (BO) (Mockus, 1974; Jones et al., 1998) to solve HPO problems (Snoek et al., 2012) has increased considerably because of its demonstrated superiority over RS in the context of HPO (Turner et al., 2021). BO is a black-box optimization algorithm which sequentially proposes configurations $\boldsymbol{\lambda}^{(1)}, ..., \boldsymbol{\lambda}^{(T)}$ on which the objective is evaluated $c_{\boldsymbol{\lambda}^{(1)}}, ..., c_{\boldsymbol{\lambda}^{(T)}}$. BO is based on the idea of learning from the collection of all evaluated points, also referred to as archive $A_T = \left\{ \left(\boldsymbol{\lambda}^{(i)}, c_{\boldsymbol{\lambda}^{(i)}} \right) \right\}_{i=1,...,T}$, to choose the next configuration $\boldsymbol{\lambda}^{(T+1)}$ as efficiently as possible. This is operationalized by training a surrogate $\hat{c}$ on the archive A_T and feeding it into an acquisition function $a : \Lambda \to \mathbb{R}$ to estimate the utility of evaluating an (unseen) configuration $\boldsymbol{\lambda}$. Typically, a probabilistic model is chosen as a surrogate model, such as a Gaussian process (Rasmussen, 2003). This allows not only considering the predicted performance of an HPC but also (posterior) uncertainty within an acquisition function. Based on predicted performance and (posterior) uncertainty, acquisition functions typically trade off exploration (i.e., sampling in regions with few data points and high posterior uncertainty) and exploitation (i.e., sampling in regions with low mean). Commonly used acquisition functions are the expected improvement (EI) (Jones et al., 1998), the lower confidence bound (LCB) (Jones, 2001; Srinivas et al., 2010), entropy search (Hennig and Schuler, 2012; Hernández-Lobato et al., 2014) and knowledge gradient (Wu et al., 2017). Song et al. (2022) demonstrate how many acquisition functions (e.g., EI) can be equivalently obtained by a likelihood-free formulation, thus extending BO to a broader class of models and acquisition functions. Extensions of BO to multi-objective problems do exist as well (Knowles, 2006; Ponweiser et al., 2008; Belakaria et al., 2019).

Another class of optimization algorithms applied to HPO problems are evolutionary strategies (ES), as very aptly summarized by Bischl et al. (2021). Examples are the (single-objective) CMAE-ES algorithm (Hansen and Auger, 2011), e.g., used in the context of HPO for deep neural networks (Loshchilov and Hutter, 2016). ES can be straightforwardly extended to also address multi-objective problems; a popular multi-objective algorithm is the NSGA-II (Deb et al., 2002), for example incorporated in the AutoML framework TPOT (Le et al., 2020). Figure 2.3 illustrates typical behavior of the GS, RS, BO, and ES.

In the context of increasing dataset sizes and increasingly complex models making performance evaluation more expensive, multi-fidelity algorithms have become a common technique to lower the resources for evaluations within HPO (Feurer and Hutter, 2019). Multifidelity algorithms are a class of optimization algorithms that leverage lower fidelity evaluations (e.g., training a neural network for fewer epochs) to optimize an objective function more efficiently. A popular example is the Hyperband (Li et al., 2017) algorithm, which is based on successive halving (Jamieson and Talwalkar, 2016), and extensions to it like BOHB (Falkner et al., 2018). Hyperband starts with a set of models trained at a low resource level (for example, the number of epochs a neural network is trained), typically resulting in a low-fidelity approximation of the objective value. The models with the best approximated function value are progressed into the next stage (e.g., half of the models), where they are retrained with increased resources (e.g., doubled resources). This process is repeated based on a fixed schedule, optimized for computational efficiency and fulfilling convergence properties.

Most established HPO libraries rely on variants of the algorithms introduced above. *SMAC* (Hutter et al., 2011), for example, implements an interleaving strategy, iterating between BO with a random forest surrogate model and randomly sampled configurations. The framework *optuna* (Akiba et al., 2019) allows the user to select between different samplers, including, but not limited to, GS, RS, BO with Tree-structured Parzen estimators (Bergstra et al., 2011), and CMA-ES, which can flexibly be combined with pruners (e.g., Hyperband) to foster efficiency. More exhaustive overviews over HPO frameworks are given by Bischl et al. (2021) and Zöller and Huber (2021).

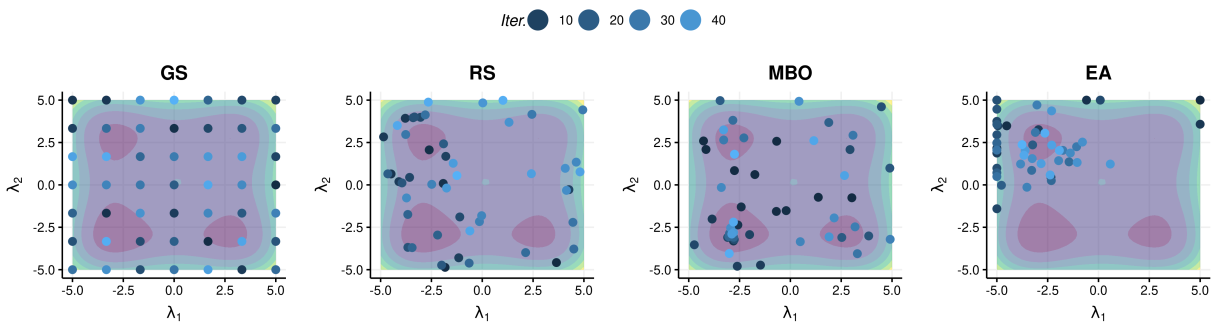


Figure 2.3.: Shown are example runs of four optimization algorithms – GS, RS, BO, and CMAE-ES – on the Styblinski-Tang function as objective. GS evaluates on a specified grid of configurations, potentially missing out good configurations in between grid points. RS evaluates points randomly chosen from a distribution (here: uniform). BO iterates between exploration and exploitation. ES, here, the CMAE-ES, often have a high risk of getting stuck in local minima.

2.3.2. AutoML frameworks

Typically, an AutoML framework consists of two main components: a pipeline definition and an optimizer. The optimizer generally is an HPO algorithm capable of optimizing over a search space defined based on the pipeline.

There is a variety of open-source software packages for AutoML. Many of those use variants of BO as optimizers. One of the first AutoML frameworks was *Auto-WEKA* (Thornton et al., 2013), which employs *SMAC* (Hutter et al., 2011) to search over the space of configurations of the *WEKA* Java ML library. *auto-sklearn* (Feurer et al., 2015) uses *SMAC* to search over a pipeline defined via the *scikit-learn* ML library, in combination with meta-learning and ensembling. *H2O AutoML* (LeDell and Poirier, 2020) succeeds with simple, straightforward techniques like RS and stacking. Some systems focus on NAS as a subarea of AutoML, such as *Auto-keras* (Jin et al., 2019), which use BO with a special kernel and acquisition function optimizer to search over a space of network architectures. The tool *ATM* (Swearingen et al., 2017) uses bandit learning in combination with BO. *FLAML* (Wang et al., 2021) performs sampling based on the *estimated cost for improvement* (ECI) and randomized direct search. *MLJAR* (Płońska and Płoński, 2021) can be run in different modes, e.g., based on the HPO framework *optuna* or just ensembling and stacking. AutoML frameworks based on ES are less common but still worth mentioning. *TPOT* (Le et al., 2020) uses genetic algorithms to optimize ML pipelines. Also *GAMA* (Gijsbers and Vanschoren, 2019) is based on genetic programming. Mohr et al. (2018) propose to alternatively understand the problem AutoML tries to solve as hierarchical planning problems and have developed the AutoML system *MLPlan* based on hierarchical task networks. *AutoGluon* uses ensembling of models and stacking them in layers (Erickson et al., 2020). Coors et al. (2021) have developed an AutoML system that constructs an interpretable additive model.

There are other AutoML frameworks that focus on particular application areas: *AMLBID* (Garouani et al., 2021) for industrial applications, *LightAutoML* (Vakhrushev et al., 2021) for financial applications, and *AutoPrognosis* (Alaa and van der Schaar, 2018) for clinical prognostic modeling.

Beyond that, it should be noted that several commercial AutoML platforms are available on the market, including *DataRobot*, *H2O Driverless AI*, *Google Cloud AutoML*, *Microsoft Azure AutoML*, and *Rapidminer*.

2.4. Explainability in machine learning

In this section, we give background on explainability in general and point out how explainability relates to concepts such as interpretability, fairness, trust, and interactivity.

What is explainability? In linguistic usage, explanations provide answers to questions that present a certain level of intellectual difficulty (Bromberger, 1992). The term *explainability* expresses the capability of giving an answer to a specific question asked. In the field of AI and ML, explainability refers to the degree to which the decisions made by an ML model or system can be explained to and understood by humans. In this context, an *explanation* is often "the answer to a why-question" (Miller, 2019). Based on the exact sense of the word, the term *explainability* raises among philosophers the question of whether a fact can ever be shown to be ultimate and unexplainable (Bromberger, 1992). We use the term explainability – somewhat apart from the exact literal sense of the word – as the capability of finding *good enough* explanations to relevant and non-trivial questions about an ML model or system.

Note that the field of explainability in AI primarily focuses on the explainability of ML models. We will use the term explainability in the context of AutoML (see Chapter 3) more broadly to explain in human-understandable terms how *system* maps inputs to outputs.

What constitutes a good explanation? When it comes to what is a *good* explanation, we follow Gilpin et al. (2018) who argue for assessing the quality of explanations via their

- *interpretability*, i.e., the ability of describing the internals of a system in a way understandable to a human[6]
- *completeness*, i.e., the ability to describe the operation of a system in an accurate way.

Gilpin et al. (2018) note that these two desiderata of explanations are typically conflicting: the most accurate explanations are not easily interpretable by people, and conversely, the most interpretable descriptions often do not provide predictive power.

Many authors do not differentiate between the closely tied concepts of *explainability* and *interpretability* (Miller, 2019; Molnar, 2022), and the community is using these concepts interchangeably. Still today, the terms hold no agreed-upon meaning, and definitions lack mathematical rigor (Doshi-Velez and Kim, 2017). All definitions share that a human subject is in the center and that interpretability depends on the target explainee (Carvalho et al., 2019). We will further on always assume that *explainability* of a system is our target and will dub methods as *interpretability methods*, which shall be both human-understandable and complete to achieve explainability.

Just as there is no consensus on what interpretability is in the first place, there is no consensus on how explainability can and should be measured. Doshi-Velez and Kim (2017) propose different ways to evaluate interpretability: Application-level evaluation, referring to putting an explanation into the product and having it tested by the end user; human-level evaluation, which refers to a simplified setting in which typically laypersons evaluate explanations; and function-level

[6]Lipton (2018) define interpretability as "the ability to explain or to present in understandable terms to a human"; Doran et al. (2017) define interpretable systems as systems where users cannot only see but also study and understand how inputs are mathematically mapped to outputs.

evaluation, which does not require humans but instead tries to find a proxy measure that is related to increased explainability (e.g., model complexity).

Types of interpretability Molnar (2022) distinguishes interpretability methods via the following criteria[7]:

- *intrinsic* or *post-hoc*: Is interpretability achieved by restricting the complexity of a system (intrinsic) or by applying methods that analyze the system afterward (post-hoc)?

  An example of an intrinsically interpretable model is the linear model, for which humans understand model parameters in simple terms. For models that are not inherently interpretable, interpretability can be achieved by applying a method like a partial dependence plot (PDP) (Friedman, 2001) post-hoc to a model or system.

- *Result of interpretation method*: To which question does the interpretation method provide an answer?

  Examples are effect estimates (How does an input influence the output? Does it have a positive, negative, linear or nonlinear effect on the outcome?), importance (Which input feature influences the outcome most?), counterfactual explanations (What needs to change in the input such that the output of the system changes?)

- *Local or global*: Does the interpretation method explain an individual instance or the entire (system) behavior?

  Examples of a local method are counterfactual explanations since they relate to a specific instance, not the overall system. PDPs, for example, provide an interpretation of the whole system.

- *(Model-/system-)specific or agnostic:* Is the explanation method limited to a specific (model-/system-)class or is it agnostic?

  An example of an agnostic method is PDP or permutation feature importance (PFI) because they internally only require a model or system to return an output for a given input.

We refer to Molnar (2022) for a more exhaustive overview of different interpretability methods.

The importance of explainable systems Explainability of models and systems is important for several reasons:

- *Understanding*: Explainability allows users to understand how a model or system makes decisions. Understandability can make debugging, error analysis, and model selection easier.

- *Trust*: Trust is the confidence and reliance users have in a model or system. Explainability fostering understandability can be essential for building trust and gaining confidence in ML and automated systems.

[7]We mostly follow those criteria but extend this notion from *model interpretability* to *system interpretability* in general, so they also fit the concept of explainability within AutoML.

- *Fairness*: Fairness refers to the idea that a model or system should not make biased or discriminatory decisions. Explainability can be important for promoting fairness in ML, as it allows users to understand how a model or system makes decisions and helps identify potential biases or unfairnesses.
- *Interactivity*: Interactivity refers to the degree to which users can interact with and modify a model or system. Explainability can be important for promoting interactivity in ML because users need to understand the internals of a model or system to some degree to interact with the systems in a sensible manner.
- *Regulation*: In some contexts, laws or regulations may require interpretability of models or systems that significantly impact people's lives to reduce the risk of unintended consequences to society.

Overall, interpretability is closely related to several other essential concepts in ML, including fairness, trust, and interactivity.

Part II.

A systematic view on explainable automated machine learning

3. Levels of explainability in the context of AutoML

Most literature on explainability in the field of ML refers to model explainability. We argue that in the context of AutoML, explainability may not only play a role on the level of ML models, the outcome of first-level inference (2.11), but on all three levels of inference introduced in Section 2.2. We discuss the role of explainability in the AutoML systematically, and review and classify existing frameworks in Table 3.3.

3.1. Model explainability

Automated HPO aims to identify a HPC $\boldsymbol{\lambda}$ of an inducer $\mathcal{I}$ leading to a model f with a low generalization error. HPO algorithms focusing solely on generalization performance may be adequate for use cases (1) when no significant consequences for unacceptable results are to be expected or (2) if the problem is sufficiently well-studied and validated (Doshi-Velez and Kim, 2017). As pointed out in Section 2.4, there may be a need to explain the model, the outcome of an AutoML system. This need is particularly evident in areas where model decisions can profoundly impact human lives or where the system is considered high risk for other reasons.

The explainability of models can be a vital requirement for different stakeholders; stakeholder groups may contrast in the types of questions they ask about model behavior. Global interpretations, which describe the average behavior of an ML model, may be of interest to modelers (or executors of an AutoML system) who may want to understand the general mechanisms in the data or debug a model (Molnar, 2022). Local interpretations, which aim to explain individual predictions rather than global model behavior, are of particular interest to stakeholders who have to communicate and justify decisions based on a model (e.g., loan company officers being answerable for why the loan company rejected a loan). Policymakers, regulators, auditors, or other stakeholders with specific responsibilities may be interested in global and local interpretations to ensure no unintended social consequences.

Formally, the system we are trying to interpret is a (predictive) model

$$\begin{aligned} f_{\boldsymbol{\theta}} : \mathcal{X} &\to \mathbb{R}^g \\ \mathbf{x} &\mapsto \hat{y}, \end{aligned} \tag{3.1}$$

as the output of first-level inference (2.11) the context of AutoML systems. Primarily, stakeholders are interested in the relationship between input $\mathbf{x} \in \mathcal{X}$ to a predictive model and its output $\hat{y} = f_{\boldsymbol{\theta}}(\mathbf{x})$. Example questions that stakeholders seek explanations for include (Molnar, 2022):

- *Feature effects*: How does an input feature $\mathbf{x}_j$ impact the model output $\hat{y}$ on average?
- *Feature importance*: Which input feature j does influence the output $f(\mathbf{x})$ most on average?
- *Feature interaction*: To what degree to predictions result from joint effects $\mathbf{x}_j \cdot \mathbf{x}_k$ of the features?
- *Counterfactual explanations*: How does the input, $\mathbf{x}$, need to change, $\tilde{\mathbf{x}}$, to realize another model outcome?

AutoML systems are often solely designed to achieve high generalization performance (Hutter et al., 2011; Thornton et al., 2013; Feurer et al., 2015) and therefore cover a rich hypothesis space of complex, nonlinear black-box models. This may run the risk of choosing a complex black-box model, even though a more interpretable model exists with almost no loss in generalization performance. To assess the trade-off between predictive performance and interpretability, Freitas (2019) performed multiple runs of the AutoML framework *AutoWEKA* over the search space, which includes both inherently interpretable (white-box) as well as black-box models. They then compared the best white-box against the best black-box model. They find that often only a small loss in predictive accuracy (often, less than 1%) needs to be sacrificed for higher interpretability.

We see three ways of designing model explainability into AutoML systems, which can complement each other.

AutoML and usage of post-hoc IML methods (A) One approach to explainability of models returned by an AutoML system is to apply post-hoc methods to the returned model. Technically, any valid interpretability method such as permutation feature importance (Strobl et al., 2008), partial dependence plots (Friedman, 2001), or methods to obtain counterfactual explanations (Wachter et al., 2017; Dandl et al., 2020) can be used. Particularly model-agnostic methods qualify for being used for this purpose. Otherwise, an interpretation technique needs to be chosen conditional on the model class selected by the AutoML tool, which brings additional complexity. Appropriate interpretability methods should be implemented into AutoML systems for enhanced usability. The advantage of coupling AutoML with post-hoc, model-agnostic interpretability methods is that the search performed by the AutoML system is unaffected. We stress, however, that interpretability methods come with their pitfalls (Molnar et al., 2020). In particular, there is no one-fits-all interpretability method, and users of AutoML tools must carefully verify assumptions behind interpretability methods.

Examples of frameworks, including post-hoc interpretability methods, are *AMLBID* (Garouani et al., 2021) and *MLJar* (Płońska and Płoński, 2021), which have built-in methods like SHAP values and feature importance. Commercial platforms for AutoML also often offer post-hoc model explainability features. For example, *H20 Driverless AI* (LeDell and Poirier, 2020) provides an interpretability toolbox providing access to interpretability methods like SHAP (Lundberg and Lee, 2017), PDPs (Friedman, 2001) and individual conditional expectation (ICE) (Goldstein et al., 2015). Also, *DataRobot* offers a model-agnostic explainability framework for feature impact, feature effects, and prediction explanations. *Vertex AI*, an ML platform developed by Google, allows using AutoML in combination with an explanation component, which provides a user with example-based explanations (Cai et al., 2019). In addition, they give access to feature-based explanations for both tabular and image data based on Shapley values (Lundberg and Lee, 2017), integrated gradients (Sundararajan et al., 2017), and XRAI (Kapishnikov et al., 2019).

AutoML comprising inherently interpretable models (B) Another way to design explainability into AutoML systems is defining search spaces that only contain inherently interpretable models like the (normal) linear regression, logistic regression, or decision trees. Designing AutoML frameworks with only inherently interpretable models is aligned with the opinion of Rudin (2019), who strongly argues in favor of designing and using inherently interpretable models, stressing that post-hoc methods often provide explanations that are not faithful to what the original model computes or do not provide enough detail. The disadvantage of this variant is that the hypothesis space is limited, which can lead to models with less predictive power. We also emphasize that complex feature engineering operators may make the final prediction system less interpretable, so the pipeline should consider only implementing interpretable preprocessing operations.

An example of an inherently interpretable AutoML system is *autocompboost* (Coors et al., 2021), which is based on *component-wise boosting* (Schmid and Hothorn, 2008). Some AutoML systems can be run in an alternative mode where explainability is enforced through a search space restriction. Examples are *LightAutoML* (Vakhrushev et al., 2021), an AutoML framework designed for financial applications, or *AutoGluon*, which can be run on a search space restricted to interpretable tree-based models (Greedy CART decision tree, Hierarchical shrinkage tree, fast interpretable greedy-tree sum, RuleFit). *auto-sklearn* allows restricting the search space to an arbitrary subset of models and processing operators specified by the user. *MLJar* can be run in *Explain* mode, which combines a restricted search space and additional post-hoc interpretability methods.

Multi-objective AutoML (C) Predictive performance and model interpretability are often conflicting objectives. Different applications may require different trade-offs between the two objectives. As observed by Freitas (2019), it might be only a neglectable drop in predictive performance that must be sacrificed to replace a black-box model with an inherently interpretable one. The criterion of model interpretability can be formulated as constraint[1] to the HPO problem (2.6), or alternatively, be designed into the objective function through scalarization. However, both introducing a constraint and defining a scalarization can be challenging since it requires specifying the trade-off between predictive performance and interpretability a-priori. We suggest formulating the problem (2.6) as a multi-objective problem with predictive performance as primary and other measures related to model interpretability or model complexity as secondary objectives. Such a multi-objective AutoML system would be able to present the user with several solution candidates representing different trade-offs between the conflicting objectives. A multi-objective AutoML tool was developed by Pfisterer et al. (2019), allowing a user to optimize for multiple measures related to fairness or interpretability at the same time. One main challenge of such systems is finding a reasonable measure for model interpretability to be used as an objective. There are several attempts to quantify model interpretability, for example, by using proxies like model complexity Molnar et al. (2019). Ultimately, the difficulty of making interpretability measurable can be traced back to a lack of a mathematically sound definition of interpretability in the community.

Comparatively speaking, all approaches have their advantages and disadvantages. A clear disadvantage of AutoML in combination with post-hoc methods (A) is potentially unfaithful explanations due to using interpretability methods under wrong assumptions. The disadvantage

[1]Note that the restriction to the search space of only inherently interpretable models (B) corresponds to placing a constraint over the search space.

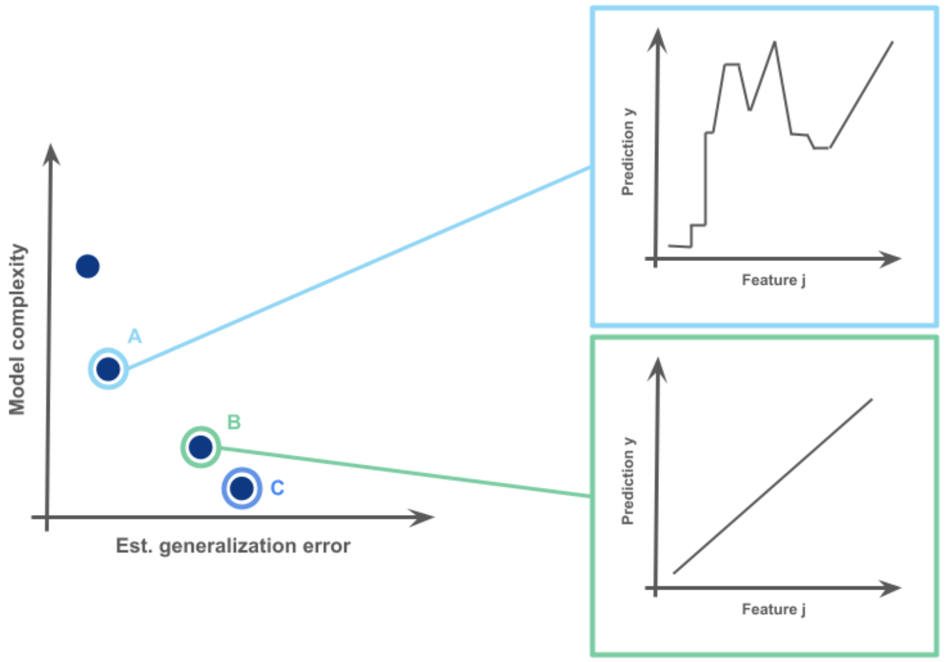


Figure 3.1.: Example of different solutions in configuration space Λ representing different trade-offs between the objectives predictive performance and model complexity (representing a proxy for model interpretability). Imagine that model A represents a random forest, model B represents a linear model with no features dropped, and model C represents a linear model with some features dropped. Suppose generalization performance is more important to a user. In that case, the user could choose de random forest and apply an IML method such as the PDP to analyze feature effects (which may be an unfaithful representation of the true effect because of interactions in the data, for example). Alternatively, if model interpretability is valued higher, a user could choose the linear model B, which is inherently interpretable. An additional reduction of model complexity through a drop in features, as in model C might not be worth the drop in performance.

of AutoML covering only inherently interpretable model classes (B) is that possibly better functioning, more complex models are not explored. We believe it should be left to the user, who is ideally also a domain expert, to decide how much loss in predictive performance can or should be taken in favor of higher interpretability, which can be facilitated through multi-objective AutoML systems (C). The most interpretable model can additionally be interpreted with post-hoc methods as needed.

3.2. Machine learning algorithm explainability

Beyond the explainability of the model as the final output of an AutoML system, the explainability of the learning algorithm $\mathcal{I}$ (or, more generally, of the inducing mechanism) is a requirement specific stakeholders may demand. Hasebrook et al. (2022); Drozdal et al. (2020); Garouani et al. (2021) assessed the lack of interpretability on this level as one of the reasons for the lack of adoption of AutoML systems and HPO. Hasebrook et al. (2022) tried to understand why practitioners still prefer manual tuning over automated approaches and identified the lack of comprehension about how certain hyperparameters influence ML models one of the reasons. Garouani et al. (2021) state that acceptance of, and trust in, an AutoML system highly depends on the transparency of its recommendations. Similarly, Drozdal et al. (2020) argue that a lack of confidence in AutoML tools slows or blocks broad adoption while warning that transparency without understandability will not lead to increased trust. Garouani et al. (2021) very aptly demand that AutoML designers and system builders present not only the final model results coming out of the system but also the pipeline steps and decisions made in each of those steps, along with the model generation process.

Explainability on this level will be mostly of relevance to ML practitioners and developers of AutoML systems who want to comprehend decisions of an AutoML system based on a general understanding of the relationship between inputs to modeling (the data $\mathcal{D}$ and hyperparameters $\boldsymbol{\lambda}$) and the output (the model and its properties). Beyond providing a general understanding, which is a benefit in itself, such insights can help answer other questions related to the correct operation or diagnostics of an AutoML system, such as: Did AutoML system run long enough or did it sufficiently explore the search space (Garouani et al., 2021)? Is the behavior and output of the AutoML system plausible? The explainability of the learning algorithm or inducer is also of general interest to the research communities. AutoML systems can be seen as huge experimentation systems experimenting with various modeling choices and assessing the respective output. Methods of interpretability developed to analyze this data could help generate new insights into how learning algorithms work. Similar to the above, decision-makers, regulators, auditors, or other stakeholders with special responsibilities may need to explain how a learning algorithm works before approving a model.

Here, our goal is to ensure explainability of the inducing mechanism[2]

$$\begin{aligned} \mathcal{I} : \mathbb{D} \times \Lambda &\to \Theta \\ (\mathcal{D}, \boldsymbol{\lambda}) &\mapsto \boldsymbol{\theta}, \end{aligned} \tag{3.2}$$

as the output of second-level inference in the context of AutoML systems. Typically, we are interested in the relationship between hyperparameters and properties of the model $f_{\boldsymbol{\theta}}$. Typical questions that stakeholders may seek explanations for include:

- *Hyperparameter effects*: How do specific hyperparameter $\boldsymbol{\lambda}_j$ (incl. selected model) influence model properties like generalization performance? (Hasebrook et al., 2022)
- *Hyperparameter importance*: Which hyperparameters j are important to realize certain model properties? (Hutter et al., 2014; Probst et al., 2019; Biedenkapp et al., 2017)
- *Hyperparameter interaction*: To what degree are model properties result from joint effects $\boldsymbol{\lambda}_j \cdot \boldsymbol{\lambda}_k$ of hyperparameters? (Hutter et al., 2014)

[2]Note again, that even decisions like preprocessing or model selection can be seen as part of an inducing mechanism $\mathcal{I}$ and encoded into hyperparameters $\boldsymbol{\lambda}$

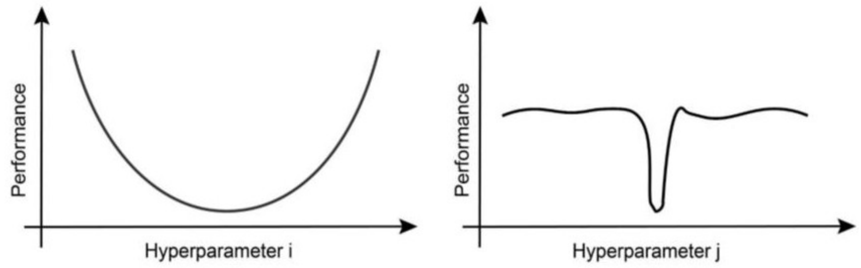
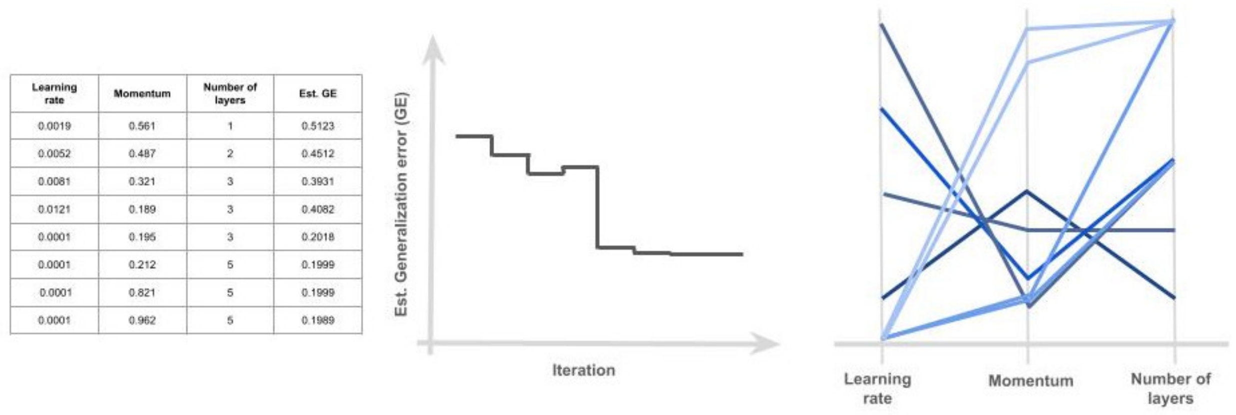


Figure 3.2.: Two hypothetical scenarios for the marginal effects of a hyperparameter λ_j. Hyperparameter importance assessed based on effects, e.g., as performed in (Hutter et al., 2014), might assess the hyperparameter in the left scenario as more important than in the right one because the hyperparameter is responsible for a bigger range of generalization performance. Assessing hyperparameter importance via ablation (Biedenkapp et al., 2017) might come to the contrary conclusion because only a slight change in the hyperparameter in the left scenario might cause a big change in performance. Another notion might be the importance of tuning a hyperparameter (Probst et al., 2019). In the left scenario, it might not be as important to tune the hyperparameter because we have a fairly good chance of achieving good performance by picking a moderate value for the hyperparameter as opposed to the right scenario.

Properties of interest are certainly generalization performance of the resulting model but may also include secondary properties like interpretability, fairness, and robustness. For example, it may be of interest to what degree a high generalization performance of neural networks results from the interaction between learning rate and momentum or to what degree the interpretability of neural networks depends on the interaction between the number of layers and the size of a layer. We also emphasize that the above list could be extended by other questions relevant to stakeholders that have been less discovered, for example, questions around counterfactuals (e.g., what is the minimal change in the modeling process to obtain a fair model?) or causal relationships. We stress that for many concepts investigated, there may be no clear, agreed-upon definition, for example, for the concept of hyperparameter importance as illustrated in Figure 3.2.

For interpretation, the archive (i.e., all configurations evaluated during an optimization run), as well as surrogate models (either already used during the optimization, if model-based optimization has taken place or post-hoc fitted on the archive), can be used. We observe two distinct variants used in practice:

Visualization tools (A) Some AutoML systems and HPO tools offer visual analytics tools that can be used to explore the artifacts from optimization runs. Commonly integrated visualizations are parallel coordinate plots (Weidele et al., 2020; Akiba et al., 2019), visualizing a performance metric against different hyperparameter configurations, or visualization of the pipeline structures explored by an AutoML system (Ono et al., 2021). Figure 3.3 illustrates common visualization tools and a hypothetical conclusion a user could draw from them. Appropriate use of visualizations (or combinations of different visualizations) of artifacts like the archive may allow for conclusions on concepts like hyperparameter effects and importance if interpreted correctly. The clear advantage is that there is no estimation with potential errors but only a simple visualization. On the other hand, it is left to the user to perform pattern recognition and draw conclusions on their



Learning rate	Momentum	Number of layers	Est. GE
0.0019	0.561	1	0.5123
0.0052	0.487	2	0.4512
0.0081	0.321	3	0.3931
0.0121	0.189	3	0.4082
0.0001	0.195	3	0.2018
0.0001	0.212	5	0.1999
0.0001	0.821	5	0.1999
0.0001	0.962	5	0.1989

Figure 3.3.: The figure shows different visualizations of the AutoML process, based on which a user of an AutoML system may conclude: the archive (left), a progress plot (middle), and a parallel coordinates plot (right).

questions, requiring expertise and time. In particular, the more complex and high-dimensional a search space is, the more difficult it is typically to conclude from simple visualizations.

Examples of frameworks that include visualization tools to explain the impact of hyperparameters are the HPO framework *optuna* (Akiba et al., 2019). *ATMSeer*, an extension to the AutoML framework *ATM*, does provide access to interactive tools to visualize performance against hyperparameters to assist users in refining the search space of AutoML and analyzing the results. Also, the AutoML tool *GAMA* (Gijsbers and Vanschoren, 2019) does provide an analysis functionality that has a set of visualization presets allowing to assess, for example, performance metrics against pipeline sizes.

There are also stand-alone visualization tools that can be used in combination with existing AutoML frameworks. One example is *Hypertendril* (Park et al., 2021), a visual analytics dashboard for HPO of deep neural networks to facilitate human interaction. Further examples are *AutoAIViz* (Weidele et al., 2020), a visualization toolbox based on conditional parallel coordinate plot, and *XAutoML* (Zöller et al., 2022), who propose a dashboard with interactive visualization tools and interpretability methods, and *PipelineProfiler* (Ono et al., 2021), an interactive visualization tool allowing to explore different pipeline configurations explored by an AutoML run. *CAVE* (Biedenkapp et al., 2018) and its successor *deepCAVE* (Sass et al., 2022) are interactive dashboards for analyzing AutoML runs with both simple visualization but also more advanced interpretability methods.

Interpretability methods (B) While visualization methods can provide insights, it is still up to the user to detect patterns and draw conclusions. Thus, visualization tools may not suffice to answer all relevant questions satisfactorily. Alternatively, a surrogate model fitted on the archive[3] can be used to derive explanations. If this surrogate model is not inherently explainable, post-hoc interpretability methods can be used. For example, hyperparameter importance can be estimated efficiently through a *functional ANOVA* (Hutter et al., 2014) or an ablation analysis (Biedenkapp et al., 2017), based on a surrogate model. In theory, the whole toolbox of model-agnostic post-hoc

[3]Note that if BO is used for HPO, the respective surrogate model can be interpreted directly.

interpretability methods could be applied to such a surrogate model to explain the relationship between hyperparameters of the inducer and model properties. However, assumptions behind interpretability methods must be carefully investigated. Many interpretability methods have requirements such as the absence of hyperparameter interactions which can rarely be assumed in the context of HPO.

Interpretability methods for accomplishing explainability at the inducer level are not yet broadly available in HPO frameworks and AutoML tools. One example is the AutoML framework *AML-BID* (Garouani et al., 2021), which provides access to hyperparameter importance in its explainer module. The HPO framework *optuna* presents the user with hyperparameter importance values (Akiba et al., 2019). Also, standalone visualization tools like the ones of Park et al. (2021), Zöller et al. (2022), Sass et al. (2022) give access to hyperparameter importance. Biedenkapp et al. (2017) and Sass et al. (2022) also offer a visualization of the configuration footprint, which is generated through a 2D projection of the configuration space through multi-dimensional scaling. One work that is very different from the previous ones, but all the more interesting and relevant, is the system of Real et al. (2020). Real et al. (2020) introduce a novel framework to automatically search for ML algorithms from scratch based on using simple mathematical operations as building blocks only. An analysis of which ML algorithm adaptions are found by the model in relation to the dataset inputted allows a deeper understanding of how AutoML systems work. Explainability features are, however, not implemented into their framework.

3.3. AutoML system explainability

Lastly, we discuss third-level explainability in the context of AutoML, by which we understand finding answers to questions of what AutoML frameworks work better than others and why.

Explainability at this level is especially important for AutoML researchers and the research community. Often, AutoML systems are proposed and published without rigorous proof of superior performance in comparison to other AutoML tools and without insights into what causes one AutoML system to perform better than another AutoML system. Research progress may be based on a kind of unsystematic, manual forward search, reusing design decisions from previously published work and extending them on a trial-and-error basis. As a result, we as a research community may be making slower progress than we possibly could because the effect of design options of AutoML systems is not adequately explored and understood. Furthermore, for safe usage of AutoML systems, it may be of importance to not only understand which AutoML systems work best on average but also understand which AutoML systems work best on which types of tasks. It should be in the highest interest of the research community to maintain a profound understanding of the workings of a technology such powerful as AutoML.

Formally, we are interested in understanding the relationship between configuration choices γ of AutoML systems and the output (and downstream properties) of an accordingly designed AutoML system

$$\begin{aligned} \mathcal{A} : \mathbb{D} \times \boldsymbol{\gamma} &\to \boldsymbol{\lambda} \\ (\mathcal{D}, \boldsymbol{\gamma}) &\mapsto \boldsymbol{\lambda}. \end{aligned} \tag{3.3}$$

Design choices γ may include the optimizer (e.g., RS, BO, Hyperband, or any combinations thereof) along with configurations of the optimizer itself, and the design of pipeline to be searched over. Example questions asked by the research community may include:

- Which AutoML system performs best on average (or by data domain, respectively)?
- What is the effect of a specific configuration parameter (e.g., a parameter controlling exploration vs. exploitation) on the efficiency of the AutoML system?
- What design choice is most important to the performance of an AutoML system?
- What design choices are irrelevant to achieve high performance of an AutoML system and can be dropped in favor of simplicity?

We distinguish two approaches, ideally to be used in combination with each other.

Benchmarking of AutoML tools (A) To answer questions around which AutoML tools work better than others, AutoML tools should be subject to a representative benchmark. Conducting high-quality benchmarks is expensive and prone to several types of errors (Gijsbers et al., 2022): Introducing a selection bias when picking benchmarking datasets, errors in installation and configuration of the respective frameworks, or choosing an appropriate set of metrics to compare results on. Most HPO and AutoML frameworks include benchmarks in their publications, but the set of benchmarking datasets does not follow a standardized setup (Jin et al., 2019; Zimmer et al., 2021; Vakhrushev et al., 2021; Wang et al., 2021; Lindauer et al., 2022).

This carries the risk of cherry-picking and bias in the reported results. To allow for a more standardized comparison, some authors mimic the setup used by prior work (Mohr et al., 2018). Most notably, Gijsbers et al. (2022) have developed a standardized benchmark of AutoML systems which allow for easy integration of AutoML frameworks and perform end-to-end evaluations on carefully curated sets of open data sets. Some authors (Coors et al., 2021; Erickson et al., 2020; Wang et al., 2021) already have included the results of this benchmark in their papers. A broad range of AutoML frameworks has been integrated into the benchmark by Gijsbers et al. (2022), including *autosklearn* (Doshi-Velez and Kim, 2020), *FLAML* (Wang et al., 2021), *GAMA* (Gijsbers and Vanschoren, 2019), *H2O AutoML* (LeDell and Poirier, 2020), *LightAutoML* (Vakhrushev et al., 2021), *MLJar* (Płońska and Płoński, 2021). Zimmer et al. (2021) have introduced a new, standardized benchmark called *LCBench* to benchmark multi-fidelity optimization algorithms. This benchmark is based on the AutoML benchmark of Gijsbers et al. (2022) but has been adapted to be more suitable for multi-fidelity optimization.

While the initiative of Gijsbers et al. (2022) is undoubted of great value, and we stress that such standardized benchmarks should be included in published work, standardized benchmarking suites are still to be improved by the community to support more types of problems (e.g., multi-objective problems) and to study secondary properties of AutoML frameworks (such as their ability to return fair models, model complexity or interpretability). We also emphasize the need for considering the performance of AutoML frameworks in specific domains of applications. For example, Conrad et al. (2022) have benchmarked AutoML frameworks for material design. Also, we observed a lack of standardized benchmarks for complex data types (images, text, speech).

Sensitivity analysis of AutoML tools (B) The field of sensitivity analysis (Saltelli et al., 2008; Iooss and Lemaître, 2015) studies how the output of a system is influenced by its inputs. In the context of this thesis, the system of interest is an AutoML system, the input are design configurations of the AutoML system, the output of the system is a proposed configuration which is eventually relates to a model and its properties. Arguably, the most obvious question is why one AutoML system works better than another one, and what is the design decision driving this. One key challenge is the expensiveness of evaluating the Objective (2.8) for different types of inputs, as it requires running an AutoML system on multiple tasks. In addition, the performance of an AutoML system can depend on many design factors, so that a high-dimensional input must be compared to multiple outputs, making it even more difficult to conduct a methodologically sound study in practice.

Ablation analysis is a commonly encountered method in the field of AutoML and ML in general to study the differences of a source configuration γ (typically, a default configuration) against a target configuration (typically, a newly proposed configuration) by modifying one factor at a time (Fawcett and Hoos, 2016). Many AutoML systems indeed include an ablation analysis in their respective publications. *AutoGluon* (Erickson et al., 2020), for example, conclude through an ablation analysis that iterated repetitions of bagging, multi-layer stacking, bagging, and neural networks as base models to multi-layer stack ensembling are crucial to the performance of their proposed system. The authors of *optuna* (Akiba et al., 2019) deduce that their pruning component plays a crucial role by studying the performance gained from their pruning procedure based on successive halving in combination with tree-parzen estimators and RS by comparing it against a run with the pruning component being disabled as well as to median pruning as implemented in *Vizier* (Golovin et al., 2017). The authors of the AutoML tools *Autopytorch* (Zimmer et al., 2021) perform an ablation analysis concluding that the performance of the system is driven by BO with hyperband (BOHB) (as compared to plain BO), warm-starting optimization on a greedily constructed portfolio of limited size (compared to a simple portfolio construction including all incumbent configurations found on meta-datasets), and ensembling over a diverse set of ensembles. Notably, the authors of *AutoKeras* (Jin et al., 2019) conducted a parameter sensitivity analysis investigating the influences of two most important hyperparameters two their method – a parameter handling exploration vs. exploitation and a kernel parameter balancing the distance of layers and skip connections. Similarly, other authors (Vakhrushev et al., 2021; Wang et al., 2021) also include ablation analyses into their work to justify that each of their proposed design choices is justified since a removal would decrease the performance of the system considerably.

Most authors choose one-factor-at-a-time (OFAT) analyses, where only one configuration is changed while all others are constant. While the advantage of this method is that it is simple to understand and, in combination with a restricted number of configurations investigated as inputs, comparatively cheap to conduct, it comes with clear disadvantages. Most and foremost, OFAT can miss important interactions between configurations. By only changing one variable at a time, OFAT analysis can miss important interactions that may significantly impact the outcome of the experiment (Czitrom, 1999). Furthermore, they are often criticized for leaving important areas of the input space unexplored (Saltelli et al., 2008).

Furthermore, it should be noted that most authors conduct ablation analyses between their system including a set of proposed designed choices (target) vs. their system with those designed choices turned off (source). More relevant to the community is, however, analyzing the difference between their system including a set of proposed design choices (target) vs. a state-of-the-art AutoML system (typically, not their own AutoML system) as a baseline, serving to answer the question of why their proposed AutoML system works better than another one. While most authors do not conduct such an analysis, Mohr et al. (2018) is a notable exception, who analyzes the effect of individual design choices of their AutoML system *MLPlan* and puts this into relation to design choices of *AutoWEKA* (Thornton et al., 2013). It should furthermore be noted that when comparing against other AutoML tools, the difference between algorithm and implementation may play a role, and that superiority of one AutoML tool may not only come from an algorithmic design choice but may go back to a specific implementation.

	Explainability on level of		
Framework	model (1)	inducer (2)	AutoML system (3)
AMLBID (Garouani et al., 2021)	(A)	(B)	✗
AutoxgboostMC (Pfisterer et al., 2019)	(C)	✗	✗
AutoCompBoost (Coors et al., 2021)	(B)	✗	(A*)
AutoGluon (Erickson et al., 2020)	(B)	✗	(A*,B)
Auto-keras (Jin et al., 2019)	✗	✗	(A,B)
Auto-pytorch (Jin et al., 2019)	✗	✗	(A,B)
Autosklearn (Doshi-Velez and Kim, 2020)	(B)	✗	✗
AutoWEKA (Thornton et al., 2013)	✗	✗	✗
ATM / ATMSeer (Swearingen et al., 2017)	✗	(A)	✗
DataRobot	(A)	✗	✗
FLAML (Wang et al., 2021)	✗	✗	(A*, B)
GAMA (Gijsbers and Vanschoren, 2019)	✗	(A)	(B)
Vertex AI	(A)	✗	✗
Hyperopt-sklearn (Komer et al., 2019)	✗	✗	✗
H20 AutoML (LeDell and Poirier, 2020)	(A)	✗	(A*)
LightAutoML (Vakhrushev et al., 2021) 2.0	(B)	✗	(A*,B)
MLJar (Płońska and Płoński, 2021)	(A,B)	✗	(A*)
MLPlan (Mohr et al., 2018)	✗	✗	(A,B)
Optuna (Akiba et al., 2019)	✗	(A,B)	
SMAC (Lindauer et al., 2022)	✗	✗	(A)
TPOT (Le et al., 2020)	✗	✗	(A)

Table 3.1.: This table summarizes how explainability is implemented within common AutoML frameworks. Explainability on the level of the model (1) means that an AutoML system supports features to enforce or support explainability of the model returned by an AutoML system, either through the implementation of post-hoc methods into the AutoML framework (A), through a search space restricted to explainable models only (B) or through multi-objective optimization with an additional objective related to explainability (C). Explainability of the inducer (2) means that the AutoML system supports functionality to generate knowledge about the working of the inducer, either by providing simple visualizations (A) or more advanced interpretability methods (B). Third, AutoML systems had been classified as to whether there is an openly accessible benchmark (A) and a sensitivity analysis on design choices of the AutoML system. For commercial tools, the information listed relies on openly accessible data (i.e., the website of the tool) and might therefore be incomplete.

Part III.

Contributions to explainable automated machine learning

4. Contributions to explainability of models returned by AutoML systems

4.1. Multi-Objective Hyperparameter Optimization – An Overview

This paper provides an overview of the field of multi-objective HPO for ML, presents applications and summarizes the state-of-the-art of different multi-objective HPO algorithms, including evolutionary algorithm (EA)s and BO. The paper discusses model interpretability as one of the additional objectives to be optimized. The paper contributes to provide a broad understanding of the field of multi-objective HPO in general, also covering multi-objective HPO and AutoML for model explainability, see Section 3.1 – *Multi-objective AutoML (C).*

Contributing article:

Karl*, F., Pielok*, T., Moosbauer, J., Pfisterer, F., Coors, S., Binder, M., Schneider, L., Thomas, J., Richter, J., Lang, M., Garrido-Merchán, E., Branke, J., Bischl, B. (2022). Multi-Objective Hyperparameter Optimization – An Overview. *arXiv preprint arXiv:2206.07438*. Under review (ACM Transactions on Evolutionary Learning and Optimization).

Author contributions:

Florian Karl and Tobias Pielok, as first authors, have taken the main responsibility for the content of the manuscript and its revisions. The other co-authors contributed to the manuscript by preparing first drafts for some of the sections and by their revisions of the paper. Julia Moosbauer has, together with Florian Karl, prepared a first draft of the section on *Evolutionary algorithms* (4.2). Furthermore, Julia Moosbauer has prepared a first draft of the section on *Sparseness via Feature Selection* (5.6) together with Martin Binder and Michel Lang, and has drafted and revised parts of the section on *Model-based optimization* (4.3). All authors contributed to revisions of the paper.

Supplementary material available at:

- Supplementary material: `https://arxiv.org/abs/2206.07438` (full version)

Multi-Objective Hyperparameter Optimization – An Overview

FLORIAN KARL*, Fraunhofer Institut für integrierte Schaltungen, Germany
TOBIAS PIELOK*, Ludwig-Maximilians-Universität München, Germany
JULIA MOOSBAUER, Ludwig-Maximilians-Universität München, Germany
FLORIAN PFISTERER, Ludwig-Maximilians-Universität München, Germany
STEFAN COORS, Ludwig-Maximilians-Universität München, Germany
MARTIN BINDER, Ludwig-Maximilians-Universität München, Germany
LENNART SCHNEIDER, Ludwig-Maximilians-Universität München, Germany
JANEK THOMAS, Ludwig-Maximilians-Universität München, Germany
JAKOB RICHTER, Ludwig-Maximilians-Universität München, Germany
MICHEL LANG, Technische Universität Dortmund, Germany
EDUARDO C. GARRIDO-MERCHÁN, Universidad Pontificia Comillas, Spain
JUERGEN BRANKE, Warwick Business School, Germany
BERND BISCHL, Ludwig-Maximilians-Universität München, Germany

Hyperparameter optimization constitutes a large part of typical modern machine learning workflows. This arises from the fact that machine learning methods and corresponding preprocessing steps often only yield optimal performance when hyperparameters are properly tuned. But in many applications, we are not only interested in optimizing ML pipelines solely for predictive accuracy; additional metrics or constraints must be considered when determining an optimal configuration, resulting in a multi-objective optimization problem. This is often neglected in practice, due to a lack of knowledge and readily available software implementations for multi-objective hyperparameter optimization. In this work, we introduce the reader to the basics of multi-objective hyperparameter optimization and motivate its usefulness in applied ML. Furthermore, we provide an extensive survey of existing optimization strategies, both from the domain of *evolutionary algorithms* and *Bayesian optimization*. We illustrate the utility of MOO in several specific ML applications, considering objectives such as operating conditions, prediction time, sparseness, fairness, interpretability and robustness.

CCS Concepts: • **Computing methodologies → Supervised learning**; • **Theory of computation → Evolutionary algorithms**; • **Applied computing → Multi-criterion optimization and decision-making**.

Additional Key Words and Phrases: Multi-Objective Hyperparameter Optimization, Neural Architecture Search, Bayesian Optimization

1 INTRODUCTION

With the immense popularity of machine learning (ML) and data-driven solutions for many domains [229], the demand for automating the creation of suitable ML pipelines has strongly increased [126]. Automated machine learning (AutoML) and automated hyperparameter optimization (HPO) promise to simplify the ML process by enabling less experienced practitioners to optimally configured ML models for a variety of tasks - reducing manual effort and improving performance at

*Both authors contributed equally to this research.

Authors' addresses: Florian Karl, florian.karl@iis.fraunhofer.de, Fraunhofer Institut für integrierte Schaltungen, Germany; Tobias Pielok, tobias.pielok@stat.uni-muenchen.de, Ludwig-Maximilians-Universität München, Germany; Julia Moosbauer, Ludwig-Maximilians-Universität München, Germany; Florian Pfisterer, Ludwig-Maximilians-Universität München, Germany; Stefan Coors, Ludwig-Maximilians-Universität München, Germany; Martin Binder, Ludwig-Maximilians-Universität München, Germany; Lennart Schneider, Ludwig-Maximilians-Universität München, Germany; Janek Thomas, Ludwig-Maximilians-Universität München, Germany; Jakob Richter, Ludwig-Maximilians-Universität München, Germany; Michel Lang, Technische Universität Dortmund, Germany; Eduardo C. Garrido-Merchán, Universidad Pontificia Comillas, Spain; Juergen Branke, Warwick Business School, Germany; Bernd Bischl, Ludwig-Maximilians-Universität München, Germany.

the same time [30, 109, 126, 184, 264]. HPO is often classified as a black-box optimization problem, a type of optimization problem where only the outputs of the function to be optimized can be observed and no analytic expression of the underlying function is known. Furthermore, these optimization problems are often noisy and expensive [12]. Another challenge in HPO is that there is rarely a clear-cut, obvious, single performance metric, which is in stark contrast to the setup often described in many algorithm-centric research papers and introductory books [109, 126, 181]. Nowadays, models and pipelines are held to a high standard, as the ML process (as currently realized in many businesses and institutions) comes with a number of different stakeholders. While predictive performance measures are still decisive in most cases, models must be reliable, robust, accountable for their decisions, efficient for seamless deployment, and so on. For example, in many internet-of-things applications, an ML model is deployed on edge devices like smartphones, watches, and embedded systems [117]. Power consumption, memory capacity, and latency can be limiting factors when deploying models in such settings, and obvious trade-offs between these factors and predictive performance exist. The current state-of-the-art models in computer vision and natural language processing with millions or billions of weights are not applicable in such settings [24]. Even putting aside secondary-order objectives like efficiency, interpretability etc., expressing just predictive performance as a single metric can frequently be challenging. The most well known example of this is arguably in medical applications: For a diagnostic test, solely looking at misclassification rates is ill-advised: Misclassifying a sick patient as healthy (false negative) has usually much more severe consequences than classifying a healthy person erroneously as sick (false positive), i.e., different *misclassification costs*, which are often unknown or hard to quantify, have to be considered [87]. Of course, multiple objectives can in principle be aggregated into a single metric, which converts a multi-objective optimization (MOO) problem to a single-objective one (SOO). However, it is often unclear how a trade-off between different objectives should be defined *a priori*, i.e., before possible alternative solutions are known [41]. In this paper, we argue that there is substantial merit in directly approaching a multi-objective HPO (MOHPO) problem as such and give a comprehensive review of methods, tools and applications; preliminary work [29, 116] supports our view. Furthermore, other prominent work in ML research - aside from HPO - has advocated for a multi-objective perspective [128, 135, 190]. For example, as argued in Mierswa [190], many ML and data mining applications inherently concern trade-offs and thus should be approached via an MOO formulation and MOO methods. And even if the main interest lies in a single objective it still might be advantageous to approach the problem via MOO methods since they have the potential to reduce local minimas in this case [151]. MOO algorithms seek to approximate the set of *efficient* or *Pareto-optimal* solutions. These solutions have different trade-offs, but it is not possible to improve any objective without degrading at least one other objective. This set of Pareto optimal solutions can then be analyzed by domain experts in a post-hoc manner, and an informed decision can be made as to which trade-off should be used in the application, without requiring the user to specify this a priory [135, 137].

This paper provides a comprehensive review on the topic of MOHPO, explains the most popular algorithms, discusses main challenges and opportunities, and surveys existing applications. We restrict the scope of this paper to the realm of supervised ML. Unsupervised ML, in contrast, entails a different set of metrics to the scenario studied in our manuscript and is largely governed by custom, usecase-specific measures [86, 203] and sometimes even visual inspection of results. The rest of the paper is structured as follows. First we define the MOHPO problem in Section 2. Section 3 presents the theoretical foundations of MOO, i.e., how to evaluate sets of candidate solutions. Then, we introduce several important MOHPO methods in Section 4, such as variants of Bayesian Optimization (BO) and also Hyperband. Finally, Section 5 introduces a number of applications for MOHPO and their associated objectives. We will categorize these applications through exploring three overarching perspectives on the ML process: (1) Performance metrics, (2) metrics that measure costs and restrictions at deployment like efficiency, and (3) metrics that enforce reliability and interpretability.

2 HYPERPARAMETER OPTIMIZATION

2.1 The machine learning problem

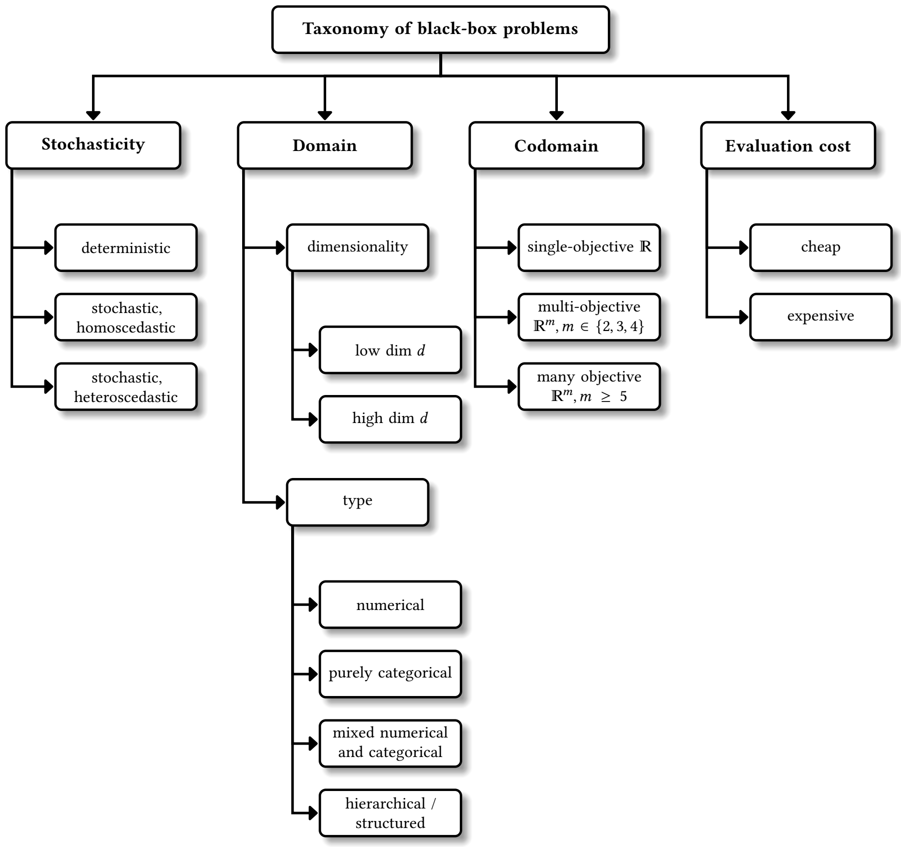

Taxonomy of black-box problems
Stochasticity
Domain
Codomain
Evaluation cost
deterministic
stochastic, homoscedastic
stochastic, heteroscedastic
dimensionality
low dim d
high dim d
type
numerical
purely categorical
mixed numerical and categorical
hierarchical / structured
single-objective $\mathbb{R}$
multi-objective $\mathbb{R}^m, m \in \{2,3,4\}$
many objective $\mathbb{R}^m, m \geq 5$
cheap
expensive

Fig. 1. Taxonomy of common black-box optimization problems. Attributes that are related to MOHPO - and therefore a substantial focus in this paper - are highlighted.

The fundamental ML problem can be defined as follows. Let $\mathcal{D}$ be a dataset with n input-output pairs $\left(\mathbf{x}^{(i)}, y^{(i)}\right) \in \mathcal{X} \times \mathcal{Y}$, which are independent and identically distributed (i.i.d.) from a data-generating distribution $\mathbb{P}_{xy}$. An ML algorithm $\mathcal{I}(\cdot, \boldsymbol{\lambda})$ configured by hyperparameters $\boldsymbol{\lambda} \in \Lambda$ maps a dataset $\mathcal{D}$ to a model $f : \mathcal{X} \to \mathbb{R}^g$ in the hypothesis space $\mathcal{H}$ via $\mathcal{I} : (\mathbb{D} \times \Lambda) \to \mathcal{H}$, where $\mathbb{D}$ is the set of all finite datasets. With a slight abuse of notation we will also write $\mathcal{I}_{\boldsymbol{\lambda}}$ if the hyperparameter $\boldsymbol{\lambda}$ is fixed, i.e., $\mathcal{I}_{\boldsymbol{\lambda}}(\mathcal{D}) = \mathcal{I}(\mathcal{D}, \boldsymbol{\lambda})$. What we would like to optimize is the expected generalization performance *GE* of our model, when trained on $\mathcal{D}$, on new unseen data $(\mathbf{x}, y) \notin \mathcal{D}$, i.e.,

$$\mathrm{GE}(\mathcal{I}, \boldsymbol{\lambda}, n, L) = \mathbb{E}\left[L(y, \mathcal{I}_{\boldsymbol{\lambda}}(\mathcal{D})(\mathbf{x}))\right] \tag{1}$$

with the expectation taken over the random data $\mathcal{D}$ of size n and a fresh test sample $(\mathbf{x}, y)$, both independently sampled from $\mathbb{P}_{xy}$. Since the data generating distribution $\mathbb{P}_{xy}$ is usually unknown, also GE has to be estimated. To do so, the data $\mathcal{D}$ is split into B training and test sets by a resampling method $\mathcal{D}^b_{\text{train}}$ and $\mathcal{D}^b_{\text{test}}$ such that $\mathcal{D} = \mathcal{D}^b_{\text{train}} \dot{\cup} \mathcal{D}^b_{\text{test}}, b = 1, \dots, B$. The expected generalization error can then be computed as

$$\widehat{\text{GE}}\left(\mathcal{I}, \mathcal{J}, L, \boldsymbol{\lambda}\right) := \frac{1}{B} \sum_{b=1}^{B} \frac{1}{|\mathcal{D}^b_{\text{test}}|} \sum_{(\mathbf{x}, y) \in \mathcal{D}^b_{\text{test}}} L\left(y, \mathcal{I}_{\boldsymbol{\lambda}}(\mathcal{D}^b_{\text{train}})(\mathbf{x})\right) \tag{2}$$

where $\mathcal{J}$ is the set of train-test-splits. Often, just a regularized error on the training data is minimized with a with point-wise loss function $L\left(y, f(\mathbf{x})\right) : \mathcal{Y} \times \mathbb{R}^g \to \mathbb{R}$.

$$\hat{f} = \arg\min_{f \in \mathcal{H}} \mathcal{R}_{\text{emp}}(f) + \eta J(f), \qquad \text{with} \qquad \mathcal{R}_{\text{emp}}(f) := \sum_{(\mathbf{x}, y) \in \mathcal{D}} L\left(y, f(\mathbf{x})\right), \tag{3}$$

The regularizer $J(f)$ expresses a preference (or prior in the Bayesian sense) for simpler models and is usually formulated as some kind of norm on the parameter vector, e.g., the L2 norm for a ridge penalty, and its strength is controlled by an HP η. [1] Most learning algorithms are configured by a possibly large number of hyperparameters $\boldsymbol{\lambda}$ that control the hypothesis space or the fitting procedure. Usually, the generalization error $\widehat{\text{GE}}\left(\mathcal{I}, \mathcal{J}, L, \boldsymbol{\lambda}\right)$ critically depends on the choice of $\boldsymbol{\lambda}$. In most cases, the analytical relationship of hyperparameters and generalization error is unknown, and even to experts it is not clear how to choose optimal hyperparameter values. Hyperparameter optimization aims to minimize the estimated generalization error $\widehat{\text{GE}}$ for a given dataset $\mathcal{D}$ using the hyperparameter configuration (HPC) $\boldsymbol{\lambda}$: $\min_{\boldsymbol{\lambda} \in \tilde{\Lambda}} \widehat{\text{GE}}\left(\mathcal{I}, \boldsymbol{\lambda}\right)$, where $\tilde{\Lambda}$, a bounded subspace of the hyperparameter space Λ, is the so-called *search-space* or *domain* in the context of the above optimization problem. Note that it is important to not use any test data during training or HPO as this could lead to an optimistic bias. Instead, nested resampling techniques should be applied [31]: The data is split into an optimization set $\mathcal{D}_{\text{optim}}$ and a test set $\mathcal{D}_{\text{test}}$. During the optimization process, the model performance should only be assessed by an (inner) resampling technique, e.g., cross-validation, on $\mathcal{D}_{\text{optim}}$, whereas $\mathcal{D}_{\text{test}}$ should only be used for the final assessment of the chosen model(s). A simple alternative is simply splitting $\mathcal{D}_{\text{optim}}$ into two datasets $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{val}}$, which leads to the widely known train/val/test-split. Afterwards, a resulting non-dominated solution set can be determined on $\mathcal{D}_{\text{test}}$ based on the solution candidates found on $\mathcal{D}_{\text{optim}}$. In practice, it is very important to have no hidden information leakage from test set to training, which may for example happen when preprocessing (such as missing value imputation or feature selection) is done on the combined set. The estimate based on a single train-test-split can have a high variance, for example because of a strong dependence on the data split. To counter this, an (outer) resampling strategy like cross-validation is usually used, and the average of the estimated generalizations of the quality indicator on $(\mathcal{D}^{(k)}_{\text{optim}}, \mathcal{D}^{(k)}_{\text{test}})$ with $k \in \mathbb{N}$ is reported.

2.2 Black box optimization

As there is generally no analytical expression of the general hyperparameter optimization problem, it forms a black-box function. Black-box functions can be characterized according to different attributes (see Figure 1), which influence the difficulty of the problem. For further fundamental details on SOO for HPO see [30]. In an ML scenario, we are often interested in the generalization error with respect to more than one loss as well as further relevant criteria like robustness, interpretability, sparseness, or efficiency of a resulting model [135]. Therefore, we introduce a generalized definition

[1]This can already be seen as a simple scalarization of a multi-objective problem, as we combine the different measures for empirical risk and regularization into a weighted sum. Mierswa [190] derives the two conflicting objectives from this scalarized version and presents an explicit multi-objective formulation of the optimization problem (minimizing the regularized empirical risk).

of the hyperparameter optimization problem that takes into account m criteria: Given a number of evaluation criteria $c_1 : \Lambda \to \mathbb{R}, \ldots, c_m : \Lambda \to \mathbb{R}$ with $m \in \mathbb{N}$, we define $c : \Lambda \to \mathbb{R}^m$ to assign an m-dimensional cost vector to a HPC $\boldsymbol{\lambda}$. The estimated generalization error $\widehat{\mathrm{GE}}$ is one evaluation criterion that is commonly used, but many further criteria will be discussed in this paper. The general multi-objective hyperparameter optimization problem can be defined as

$$\min_{\boldsymbol{\lambda} \in \Lambda} c(\boldsymbol{\lambda}) = \min_{\boldsymbol{\lambda} \in \Lambda} \left(c_1(\boldsymbol{\lambda}), c_2(\boldsymbol{\lambda}), \ldots, c_m(\boldsymbol{\lambda})\right) \ . \tag{4}$$

Without loss of generality, we assume that all criteria are minimized. The domain Λ of the problem is called numerical if only numeric HPs $\boldsymbol{\lambda}$ are optimized. By including additional discrete hyperparameters, like the type of kernel used in a support sector machine (SVM), the search space becomes *mixed* numerical and categorical. Mixed search spaces already require adaption of some optimization strategies, such as BO, which we will discuss in Section 4.3. It can also be necessary to introduce further conditional hierarchies between hyperparameters. For example, when optimizing over different kernel types of an SVM, the γ kernel hyperparameter is only valid if the kernel type is set to *Radial Basis Function* (RBF), while for a polynomial kernel, a hyperparameter for the polynomial degree must be specified. These conditional hierarchies can become highly complicated - especially when moving from pure HPO to optimizing over full ML pipelines, i.e., AutoML, or over neural network architectures, referred to as neural architecture search (NAS) [88, 200, 215]. The dimensionality of the search space $\dim(\tilde{\Lambda})$ also directly influences the difficulty of the problem: While it might be desirable to include as many hyperparameters as possible (since it is often unknown which hyperparameters are actually important), this increases the complexity of the optimization problem and requires an increasingly larger budget of (expensive) evaluations. The *co-domain*, also called objective space, i.e., $\mathbb{R}^m$, of the problem is characterized by the number of objectives m. Objective functions $c_i, i \in \{1, 2, ..., m\}$ can be characterized by their *evaluation cost* and stochasticity. Some evaluation criteria are deterministic, such as the required memory of a model on hard disk. Other criteria can only be measured with some additional noise, e.g., the generalization performance discussed above. We generally assume that all objective functions are *black-boxes*, i.e., analytical information about c_i is not available, even though there are exceptions to this (e.g., number of parameters in some models). To record the evaluated hyperparameter configurations and their respective scores, we introduce the so-called *archive* $\mathcal{A} = ((\boldsymbol{\lambda}^{(1)}, c(\boldsymbol{\lambda}^{(1)})), (\boldsymbol{\lambda}^{(2)}, c(\boldsymbol{\lambda}^{(2)})), \ldots)$, with $\mathcal{A}^{[t+1]} = \mathcal{A}^{[t]} \cup (\boldsymbol{\lambda}^+, c(\boldsymbol{\lambda}^+))$ if a single configuration is presented by an algorithm that iteratively proposes hyperparameter configurations.

Example 1. *Consider a standard scenario in which we would like to train a deep neural network for object detection on images. Assume that $m = 2$ criteria are relevant: While predictive performance is one criterion (c_1) to be optimized, we also optimize for memory consumption (c_2) of the network at prediction time, as the model is deployed on a mobile phone. Both criteria depend on the hyperparameters and architectural choices of the network $\Lambda = (\lambda_1, \lambda_2)$, and they are potentially conflicting: Memory-efficient models are usually less complex and may use special types of convolutions [117]. An insufficient capacity of the model may result in bad generalization. The trade-off between the two objectives is a priori unknown, because neither the range of generalization performance nor the user's hardware constraints are known (we explore scenarios like this one in Section 5.2). Ideally, we would like to have a set of solutions with different optimal trade-offs in the two objectives. We will formalize this notion in Section 3.2.*

2.3 Multi-objective machine learning

A concept closely related but different to MOHPO is multi-objective machine learning. To understand the difference, it is important to differentiate between first level *model parameters* (e.g., weights of a neural network or learned decision rules) and second order *hyperparameters* (HPs) (e.g., neural network architectures and optimizers) [30]. Model parameters are fixed by the ML algorithm at training time in accordance to one or multiple metrics, whereas HPs are chosen by the ML practitioner before training and influence the behavior of the learning algorithm and the structure of its associated hypothesis space. We define multi-objective ML methods as those that focus on learning first level parameters (sometimes together with second level hyperparameters). Our work, in contrast, concentrates on *hyperparameter optimization* for ML algorithms, i.e., *tuning* second level HPs. An example to further illustrate this distinction can be found in Suttorp and Igel [241]: The authors examine both tuning model parameters and hyperparameters in order to improve performance of a support vector machine (SVM). To provide a clearer picture and to enable a better distinction, we try to give an overview of existing approaches and inherent pros and cons in multi-objective ML before we continue. While this section focuses on summarizing key ideas, a more comprehensive summary and case study of such approaches was conducted in Jin and Sendhoff [137]. Broadly, multi-objective ML approaches try to internally balance for *improved generalization*, *multiple error metrics* or *improved interpretability* to obtain more diversity in *ensemble members* and many other criteria (c.f. Jin and Sendhoff [137] for a list of existing approaches). Another potential advantage of internally optimizing multiple objectives is that it may help the ML algorithm to escape local optima, thus improving the accuracy of the ML model. To provide an example, Iglesia et al. [127] use a variant of NSGA-II (c.f. Section 4.2.3) to learn association rules that are accurate, simple and diverse. However, trade-offs are not exposed as hyperparameters to be tuned, but instead the method yields a set of rules, which approximate the Pareto set. Feature selection is a topic that borders MOHPO and multi-objective ML and is often handled in a multi-objective manner [18, 29, 191]. We consider feature selection as closely related to HPO and will therefore dedicate large portions of Section 5.6 to this topic. However, it should be noted that feature selection is often grouped with multi-objective ML, as e.g., in Jin [135].

3 FOUNDATIONS OF MULTI-OBJECTIVE OPTIMIZATION

3.1 Objectives and constraints

In the context of this paper, objectives refer to the evaluation criteria of the ML model $c_1 : \tilde{\Lambda} \to \mathbb{R}, \dots, c_m : \tilde{\Lambda} \to \mathbb{R}$ with $m \in \mathbb{N}$. While oftentimes we simply aim to minimize these objectives, in real-world applications, we may well face a constrained HPO problem of the form:

$$\begin{aligned} &\underset{\boldsymbol{\lambda} \in \tilde{\Lambda}}{\text{minimize}} && c(\boldsymbol{\lambda}) && \\ &\text{subject to} && k_1(\boldsymbol{\lambda}) = 0, \dots, k_n(\boldsymbol{\lambda}) = 0 && \text{(equality constraints)}, \\ & && \hat{k}_1(\boldsymbol{\lambda}) \geq 0, \dots, \hat{k}_{\hat{n}}(\boldsymbol{\lambda}) \geq 0 && \text{(inequality constraints)}, \end{aligned}$$

where $c : \tilde{\Lambda} \to \mathbb{R}^m$ as before. It is the task of an ML practitioner to translate a real-world problem into an ML task - and therefore objectives and constraints - to measure the quality and feasibility of a given model. Depending on the use case it has to be carefully considered whether to frame a requirement to an ML model as an objective or a constraint. For example, is it important to have a model as memory-efficient as possible or is memory requirement limited by a hard constraint? Finding high performing and efficient deep learning architectures has emerged as a prominent task recently, coining the term hardware-aware NAS (HW-NAS) [24]. Successful approaches exist that frame the HW-NAS problem as a constrained optimization problem [44, 244] or a MOO problem [78, 182]. It should be noted that constrained optimization comes with its own set of challenges that are orthogonal to the multi-objective aspect we focus on in this paper. We therefore exclude constrained optimization from our work - only mentioning it in absolutely crucial parts and giving helpful references to the reader when appropriate.

3.2 Pareto optimality

A HPC $\boldsymbol{\lambda} \in \tilde{\Lambda}$ *(Pareto-)dominates* another configuration $\boldsymbol{\lambda}'$, written as[2] $\boldsymbol{\lambda} \prec \boldsymbol{\lambda}'$, if and only if

$$\begin{aligned} \forall i \in \{1, ..., m\} : c_i(\boldsymbol{\lambda}) \leq c_i(\boldsymbol{\lambda}') \wedge \\ \exists j \in \{1, ..., m\} : c_j(\boldsymbol{\lambda}) < c_j(\boldsymbol{\lambda}') . \end{aligned} \tag{5}$$

In other words: $\boldsymbol{\lambda}$ dominates $\boldsymbol{\lambda}'$, if and only if there is no criterion c_i in which $\boldsymbol{\lambda}'$ is superior to $\boldsymbol{\lambda}$, and at least one criterion c_j in which $\boldsymbol{\lambda}$ is strictly better. For example, if the number of false positive and number of false negative classifications are the two criteria of interest, a HPC configuration $\boldsymbol{\lambda}$ dominates $\boldsymbol{\lambda}'$ if the model trained by $\mathcal{I}(\cdot, \boldsymbol{\lambda})$ shows (at most) as many false positives as the model trained by $\mathcal{I}(\cdot, \boldsymbol{\lambda}')$, but produces less false negatives at prediction time.

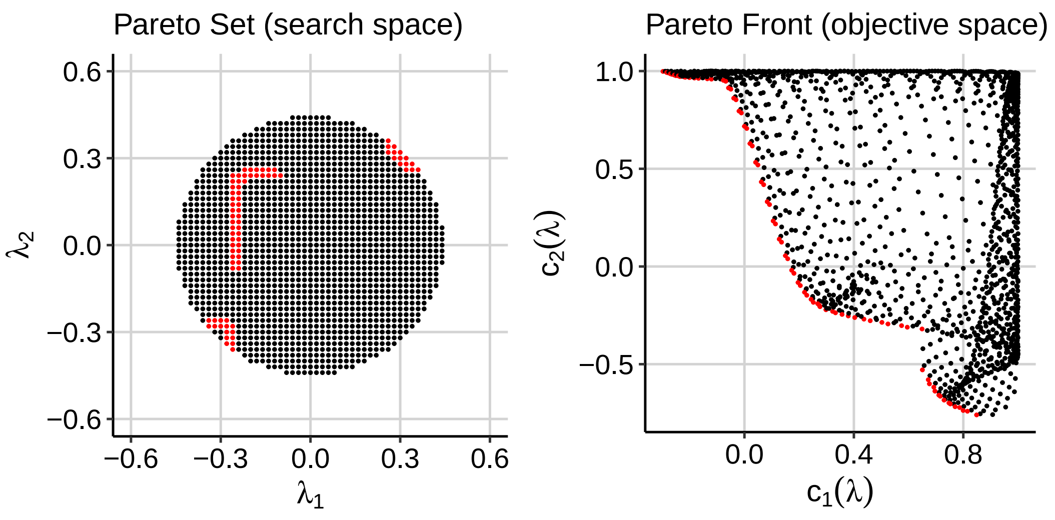


Fig. 2. Illustration for a two-dimensional MOO problem with two objectives c_1 and c_2. The left plot shows the search space $\tilde{\Lambda}$, and the right plot shows the objective space $\mathbb{R}^2$. Configurations for which no other configuration $\boldsymbol{\lambda}$ has lower objective values in both objectives - the estimated Pareto set (left) - and their mapping to the co-domain - the estimated Pareto front (right) - are highlighted.

We say $\boldsymbol{\lambda}$ *weakly dominates* $\boldsymbol{\lambda}'$, written as $\boldsymbol{\lambda} \preceq \boldsymbol{\lambda}'$, if and only if $\boldsymbol{\lambda} \prec \boldsymbol{\lambda}'$ or $\forall i \in \{1, \ldots, m\}\ c_i(\boldsymbol{\lambda}) = c_i(\boldsymbol{\lambda}')$. A configuration $\boldsymbol{\lambda}^*$ is called *non-dominated* or *(Pareto) optimal* if and only if there is no other $\boldsymbol{\lambda} \in \tilde{\Lambda}$ that dominates $\boldsymbol{\lambda}^*$. Pareto dominance defines only a partial order over $\tilde{\Lambda}$, i.e., two configurations $\boldsymbol{\lambda}$ and $\boldsymbol{\lambda}'$ can also be incomparable. This situation arises if there exist $i, j \in \{1, \ldots, m\}$ for which $c_i(\boldsymbol{\lambda}) < c_i(\boldsymbol{\lambda}')$ but also $c_j(\boldsymbol{\lambda}') < c_j(\boldsymbol{\lambda})$. Hence, in contrast to single-objective optimization, there is in general no unique single best solution $\boldsymbol{\lambda}^*$, but a set of Pareto optimal solutions that are pairwise incomparable with regard to $\prec$. This set of solutions is referred to as the *Pareto (optimal) set* or *efficient frontier* [10] and defined as

$$\mathcal{P} := \left\{\boldsymbol{\lambda} \in \tilde{\Lambda} \mid \nexists\, \boldsymbol{\lambda}' \in \tilde{\Lambda} \text{ s.t. } \boldsymbol{\lambda}' \prec \boldsymbol{\lambda}\right\} . \tag{6}$$

[2]In some literature, the direction of the domination relationship is reversed, i.e., they write $\boldsymbol{\lambda}' \prec \boldsymbol{\lambda}$ if $\boldsymbol{\lambda}$ dominates $\boldsymbol{\lambda}'$. We choose our notation because it naturally fits the minimization perspective taken in this paper.

The image of $\mathcal{P}$ under c, written as $c(\mathcal{P})$, is referred to as the Pareto front (see Figure 2). The goal of a multi-objective optimizer that solves (4) is not to find a single best configuration $\boldsymbol{\lambda}^*$, but rather a set of configurations $\hat{\mathcal{P}}$ that approximates the Pareto set $\mathcal{P}$ well.

EXAMPLE 2. *Continuing from Example 1, we compare two potential neural networks: network A and network B. Network A with configuration $\boldsymbol{\lambda}_A$ has an accuracy of $c_1(\boldsymbol{\lambda}_A) = 0.87$ and memory consumption of $c_2(\boldsymbol{\lambda}_A) = 127$, while network B with $\boldsymbol{\lambda}_B$ has the same accuracy $c_1(\boldsymbol{\lambda}_B) = 0.87$ but lower memory consumption $c_2(\boldsymbol{\lambda}_B) = 112$. Consequently, network A is dominated by network B as $\boldsymbol{\lambda}_B \prec \boldsymbol{\lambda}_A$. Considering a third network, network C, with $c_1(\boldsymbol{\lambda}_C) = 0.89$ and $c_2(\boldsymbol{\lambda}_C) = 180$, we can see that both $\boldsymbol{\lambda}_C$ and $\boldsymbol{\lambda}_B$ are non-dominated, and thus the current best approximation of the Pareto set is $\hat{\mathcal{P}} = \{\boldsymbol{\lambda}_B, \boldsymbol{\lambda}_C\}$.*

3.3 Evaluation

The result of a multi-objective algorithm is $\hat{\mathcal{P}}$, the set of points of the estimated Pareto front. In order to evaluate this set or compare it to other sets, one must define what it means for a Pareto front to be better than another. Usually, this comparison is quantitatively based on so-called quality indicators.

3.3.1 Comparing solution sets. Let $\hat{\Lambda}_{\mathcal{A}}$ and $\hat{\Lambda}_{\mathcal{B}}$ be two non-dominated solution sets - i.e., within each set, no configuration is dominated by another configuration. The associated approximated Pareto fronts are denoted by $\mathcal{A}$ and $\mathcal{B}$ respectively, i.e., $c(\hat{\Lambda}_{\mathcal{A}}) = \mathcal{A}$ and $c(\hat{\Lambda}_{\mathcal{B}}) = \mathcal{B}$. According to Zitzler et al. [271], $\mathcal{A}$ is said to *weakly dominate* $\mathcal{B}$, denoted as $\mathcal{A} \preceq \mathcal{B}$, if for every solution $\boldsymbol{\lambda}_b \in \hat{\Lambda}_{\mathcal{B}}$ there is at least one solution $\boldsymbol{\lambda}_a \in \hat{\Lambda}_{\mathcal{A}}$ which weakly dominates $\boldsymbol{\lambda}_b$. $\mathcal{A}$ is furthermore said to be *better* than $\mathcal{B}$, denoted as $\mathcal{A} \triangleright \mathcal{B}$, if $\mathcal{A} \preceq \mathcal{B}$, but not every solution of $\hat{\Lambda}_{\mathcal{A}}$ is weakly dominated by any solution in $\hat{\Lambda}_{\mathcal{B}}$, i.e., $\mathcal{A} \npreceq \mathcal{B}$. This represents the weakest form of superiority between two approximations of the Pareto front. Note that these order relationships defined for $\mathcal{A}$ and $\mathcal{B}$ can directly be transferred to the associated solution sets $\hat{\Lambda}_{\mathcal{A}}$ and $\hat{\Lambda}_{\mathcal{B}}$. How well a single solution set represents the Pareto front can be divided into four qualities [166]:

Convergence The proximity to the true Pareto front
Spread The coverage of the Pareto front
Uniformity The evenness of the distribution of the solutions
Cardinality The number of solutions

The combination of spread and uniformity is also referred to as *diversity*. One way of comparing these qualities is to visualize the solution sets. For bi-objective optimization problems, this can be straightforwardly done. However, for a higher number of objectives, the visualization and decision-making process based on this visualization can become substantially more challenging. Tušar and Filipič [251] offer a review of existing visualization methods.

3.3.2 Quality indicators. An objective measurement of the quantitative difference between solution sets is clearly desirable for comparing algorithms. Therefore, many quality indicators I - which map the approximation of the Pareto front to a real number representing the quality of a set of solutions - were proposed. An extensive overview of these indicators can be found in Li and Yao [166]. Quality indicators that focus on all four qualities listed above can be divided into *distance-based*, which require the knowledge of the true Pareto front or a suitable approximation of it, and *volume-based*, which measure the volume between the approximated Pareto Front and a method-specific point. Common distance-based quality indicators are the inverted generational distance [57], Dist2 [60] and the ϵ-indicator [271]. Common volume-based indicators are hypervolume indicator [270], the R class of indicators [106], and the integrated preference functional [47]. The most popular quality indicator is the *hypervolume indicator* [270], also called dominated hypervolume or $\mathcal{S}$-metric, since it does not require any prior knowledge of the Pareto front. The hypervolume, HV, of an approximation of the Pareto front $\mathcal{A}$ can be defined as the combined volume of the dominated hypercubes $\mathrm{domHC}_{\boldsymbol{r}}$ of all solution points $\boldsymbol{\lambda}_a \in \hat{\Lambda}_{\mathcal{A}}$ regarding a reference point $\boldsymbol{r}$, i.e.,

$$\mathrm{HV}_{\boldsymbol{r}}(\mathcal{A}) := \mu\left(\bigcup_{\boldsymbol{\lambda}_a \in \hat{\Lambda}_{\mathcal{A}}} \mathrm{domHC}_{\boldsymbol{r}}(\boldsymbol{\lambda}_a)\right),$$

where μ is the Lebesgue measure and the dominated hypercube

$$\mathrm{domHC}_{\boldsymbol{r}}(\boldsymbol{\lambda}_a) := \{\boldsymbol{u} \in \mathbb{R}^m \mid c_i(\boldsymbol{\lambda}_a) \leq \boldsymbol{u}_i \leq \boldsymbol{r}_i \ \forall i \in \{1, \dots, m\}\}.$$

HV is illustrated in Figure 3 (left). The hypervolume indicator is *strictly Pareto compliant* [269], i.e., for all solution sets $\hat{\Lambda}_{\mathcal{A}}$ and $\hat{\Lambda}_{\mathcal{B}}$, it holds that

$$\mathcal{B} \triangleright \mathcal{A} \Rightarrow \mathrm{HV}_{\boldsymbol{r}}(\mathcal{B}) < \mathrm{HV}_{\boldsymbol{r}}(\mathcal{A}).$$

Hence for the true Pareto front, the HV reaches its maximum. In practice, the *nadir point*, which is constructed from the worst objective values, is often used as the reference point. For a guideline on how to set a reference point, see [129]. Based on a quality indicator estimate, MOO strategies can

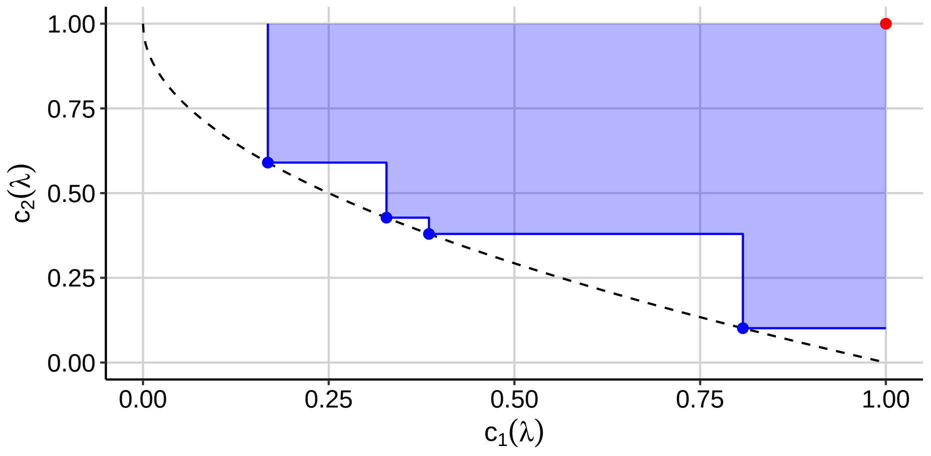


Fig. 3. The plot shows the hypervolume indicator (the area of the blue shaded region) regarding the reference point $(1, 1)$ (marked in red).

be evaluated over different benchmark datasets by using the *Nemenyi post hoc test* [71], where the strategies are compared pairwise to find the best-performing one under a significance level of α.

4 MULTI-OBJECTIVE OPTIMIZATION METHODS

This work in general and the following section in particular, focus on *a posteriori* methods, i.e., those that return a set of configurations $\hat{\mathcal{P}}$ that tries to approximate the true Pareto set $\mathcal{P}$ as well as possible. We will also explore some *a priori* methods, i.e., those that will only return one configuration depending on preferences set before optimization. Finally, we will discuss integration of user preferences to customize certain methods in Section 4.5.1. While a multitude of very specific methods for MOO exist across various domains, we try to mainly focus on those that are actually applied to MOHPO.

4.1 Naive approaches

4.1.1 Scalarization. Scalarization transforms a multi-objective goal into a single-objective one, i.e., it is a function $s : \mathbb{R}^m \times \mathcal{T} \rightarrow \mathbb{R}$ that maps m criteria to a single criterion to be optimized, configured by scalarization hyperparameters $\tau \in \mathcal{T}$. Having only one objective often simplifies the optimization problem [192]. However, there are two main drawbacks to using scalarization for MOO [137]: The scalarization hyperparameters τ must be chosen sensibly, such that the single-objective represents the desired relationship between the multiple criteria – which is not trivial, especially without extensive prior knowledge of the optimization problem. We will outline three popular scalariztaion techniques:

Weighted sum approach. One of the most well-known scalarization techniques, where one looks for the optimal solution to:

$$\min_{\boldsymbol{\lambda} \in \tilde{\Lambda}} \sum_{i=1}^{m} \tau_i c_i(\boldsymbol{\lambda}). \tag{7}$$

It can be shown [77] that for a solution $\hat{\boldsymbol{\lambda}}$ of (7), it holds that if

$$\tau_i \geq 0, \quad i = 1, \ldots, m \Rightarrow \hat{\boldsymbol{\lambda}} \text{ is non-dominated in } \tilde{\Lambda}. \tag{8}$$

Additionally, it holds for convex $\tilde{\Lambda}$ and convex functions $c_i, i = 1, \ldots, m$ that for every non-dominated solution $\hat{\boldsymbol{\lambda}}$ there exist $\tau_i \geq 0, \ i = 1, \ldots, m$, such that $\hat{\boldsymbol{\lambda}}$ is the respective solution of (7); however the weighted sum might perform poorly for nonconvex problems [77].

Tchebycheff approach. The *weighted Tchebycheff problem* is formulated as:

$$\min_{\boldsymbol{\lambda} \in \tilde{\Lambda}} \max_{i=1,\ldots,m} [\tau_i |c_i(\boldsymbol{\lambda}) - z_i^*|], \tag{9}$$

where $z_i^* = min_{\boldsymbol{\lambda} \in \tilde{\Lambda}} c_i(\boldsymbol{\lambda}) \, for \, i = 1, \ldots, m$ defines the (ideal) reference point. For every optimal solution $\boldsymbol{\lambda}^*$ there exists one combination of weights $\hat{\tau}$, so that $\boldsymbol{\lambda}^*$ is the optimal solution to the optimization problem 9 defined through $\hat{\tau}$.

ϵ-constraint approach. As outlined in Section 3.1, in some instances it might be feasible to formulate one or more of the objectives as constraints. A MOO problem can also be reduced to a single scalar optimization problem by turning all but one objective constraints, as in the *ϵ-constraint method*: Given $m - 1$ constants $(\epsilon_2, \ldots, \epsilon_m) \in \mathbf{R}^{m-1}$,

$$\min_{\boldsymbol{\lambda} \in \tilde{\Lambda}} f_1(\boldsymbol{\lambda}), \text{ subject to } f_2(\boldsymbol{\lambda}) \leq \epsilon_1, \ldots, f_m(\boldsymbol{\lambda}) \leq \epsilon_m. \tag{10}$$

However, much like parameters in the weighted sum method, constraints must be sensibly chosen, which can prove to be challenging *a priori* and without sufficient domain knowledge.

These techniques (e.g., weighted sum method, Tchebycheff method) are the same ones used in the MOEA/D Evolutionary Algorithm as outlined in Section 4.2.3 or the BO technique ParEGO as outlined in Section 4.3.2, where several scalar optimization problems are created in place of the multi-objective one.

4.1.2 Random and grid search. Random and grid search are very basic and robust algorithms for single-objective HPO [30]. Random search is generally preferred, as it is an anytime algorithm and scales better in the case of low effective dimensionality of an HPO problem with low-impact parameters [26]; it often provides a surprisingly competitive baseline. Modifying random and grid search for MOO is trivial: as all points are independently spawned and evaluated, one simply returns all non-dominated solutions from the archive. Random and grid search can serve as reasonable

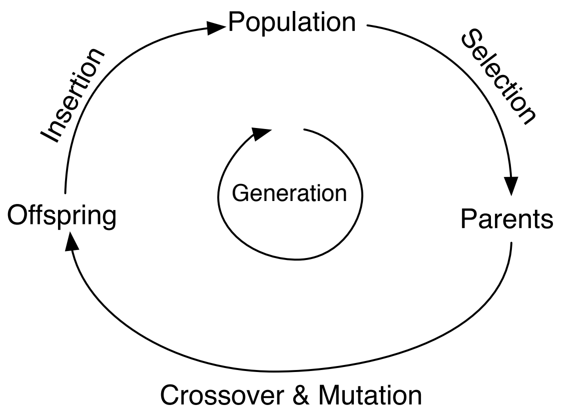


Fig. 4. Basic loop of an evolutionary algorithm.

baselines when introducing more sophisticated optimization methods - similar to the single-objective case. This procedure is adopted in a number of MOHPO works [128, 207, 230]

4.2 Evolutionary algorithms

Evolutionary algorithms (EAs) are general black-box optimization heuristics inspired by principles of natural evolution. This section provides a brief introduction, followed by some prominent examples of multi-objective EAs. EAs haven been used in HPO and AutoML applications, such as in the popular AutoML framework TPOT [201]. Swarm Intelligence methods are related to and share many of the advantages and disadvantages of EAs. They can be utilized for multi-objective optimization [89] and have been examined for MOHPO and feature selection in select works (e.g., Xue et al. [260], Bacanin et al. [13]).

4.2.1 Fundamentals of evolutionary algorithms. Evolutionary algorithms (EAs) are population-based, randomized meta-heuristics, inspired by principles of natural evolution. Historically, different variants have been independently developed such as genetic algorithms, evolution strategies and evolutionary programming. These are now generally subsumed under the term "evolutionary algorithm". EAs start by (often randomly) initializing a *population* of solutions and evaluating them. Then, in every iteration (often called *generation*), the better solutions are (often probabilistically) selected as *parents* and used to generate new solutions, so-called *offspring*. The two main operators to generate new solutions are *crossover*, which tries to recombine information from two parents into an offspring, and *mutation*, which randomly perturbs a solution. The resulting offspring solutions are then evaluated and inserted into the population. Since the population size is kept constant, some solutions have to be removed (*survival selection*), and once again usually the better solutions are chosen to survive with a higher probability. If the stopping criterion reached, the best encountered solution is returned, otherwise the next iteration starts. A schematic overview of a generation can be found in Figure 4. EAs are popular optimization methods for the following reasons: (i) no specific domain knowledge is necessary (at least for baseline results) [3], (ii) ease of implementation [3], (iii) low likelihood of becoming trapped in local minima [252], (iv) general robustness and flexibility [101], and (v) straightforward parallelization [3]. They are widely employed in practice and known for successfully dealing with complex problems and complex search spaces where other optimizers may fail [109]. Furthermore, because they are population-based, EAs are particularly well suited for multi-objective optimization, as they can simultaneously search for a set of solutions that approximate the Pareto front.

4.2.2 Multi-objective evolutionary algorithms (MOEAs). When applying EAs to a multi-objective problem, the only component that needs changing is the selection step (selecting parents and selecting survivors to the next generation). Whereas in single-objective optimization the objective function can be used to rank individuals and select the better ones, as discussed in Section 3.2, in multi-objective optimization there are many Pareto-optimal solutions with different trade-offs between the objectives, and we are interested in finding a good approximation of the Pareto front. Recent works categorize MOEAs into three classes [80]:

- Pareto dominance-based algorithms use two-levels for ranking: On the first level, Pareto dominance is used for a coarse ranking (usually non-dominated sorting, see below), while on the second level usually some sort of diversity measure is used to refine the ranking of the first level.
- Decomposition-based algorithms utilize scalarization (see Section 4.1.1) to decompose the original problem into a number of single-objective subproblems with different parametrizations, which are then solved simultaneously.
- Indicator-based algorithms use only a single metric, such as the hypervolume indicator, and selection is governed by a solution's marginal contribution to the indicator.

In the following, we will provide a prominent example for each category.

4.2.3 Prominent MOEAs.

NSGA-II (Pareto dominance-based). The *non-dominated sorting genetic algorithm* (NSGA-II) [68] is still one of the most popular MOEAs. In many benchmark studies in the field, it serves as a popular baseline [207, 252]. Being Pareto dominance based, it first uses *non-dominated sorting* to obtain an initial coarse ranking of the population. This iteratively determines the non-dominated solutions, assigns them to the next available class, and removes them from consideration. Among the solutions in each obtained class, the extreme solutions (best in each objective) are ranked highest, and the remaining solutions are ranked according to the *crowding distance*, the sum of differences between an individual's left and right neighbor, in each objective, where large crowding distances are preferred. While this works very well for problems with two objectives, it breaks down in case of a larger number of objectives, as the non-dominated sorting becomes less discriminative, and the left and right neighbor in each objective are often different solutions. NSGA-III [67, 131] has been developed as an alternative for problems with a higher number of objectives, but shares many similarities with decomposition-based algorithms.

MOEA/D (Decomposition-based). The *Multi-objective Evolutionary Algorithm based on Decomposition* (MOEA/D) decomposes the multi-objective problem into a finite number N of scalar optimization problems that are then optimized simultaneously [266]. Each single optimization problem usually uses a Tschebyscheff scalarization, see Section 4.1.1. In theory, each solution to such a scalar problem should be a point on the Pareto front of the original multi-objective problem. The distribution of solutions on the Pareto front is thus governed by the set of scalarizations chosen, and it is challenging to identify scalarizations without a good knowledge of the Pareto frontier. Rather than solving the different scalarized problems independently, the idea is to solve them simultaneously, and allow the different search processes to influence each other. In a nutshell, the population comprises of the best solution found so far for each of the sub-problems. In every generation, a new offspring is created for each sub-problem by randomly selecting two parents from the sub-problem's neighborhood, performing crossover and mutation, and re-inserting the individual into the population. The new individual replaces all individuals in the population for which it is better with respect to the corresponding sub-problem. In effect, this means mating is restricted to among individuals from the same region of the non-dominated frontier, and diversity in the population is maintained implicitly by the definition of the different sub-problems.

SMS-EMOA (Indicator-based). The *S metric selection evolutionary multi-objective optimization algorithm* (SMS-EMOA) [79] also uses the non-dominated sorting algorithm from NSGA-II for an initial coarse ranking, but then uses an individual's marginal hypervolume contribution for as a secondary criterion. The marginal hypervolume of an individual i is the difference in hypervolume between the population $\mathcal{R}$ including individual i, and excluding individual i:

$$\Delta_{HV}(i, \mathcal{R}) := HV(\mathcal{R}) - HV(\mathcal{R} \setminus i) \tag{11}$$

Different from the other two algorithms above, SMS-EMOA only produces one offspring per generation, adds it to the population and then discards the worst solution based on the ranking just described.

For more details on the above MOEAs, different MOEAs, and according applications, please refer to Coello et al. [56], Abraham and Jain [3], Deb [65], and Branke et al. [39]. The main disadvantage of many evolutionary algorithms is their relatively slow convergence (compared to other optimization methods) and the need for many evaluations; for more computationally expensive ML problems, a multi-objective HPO can become very costly when tackling it with these methods [65]. To alleviate this problems, MOEAs have been combined with surrogate-modelling techniques (e.g.,[181]) or gradient-based local search (e.g., [158]). EAs can further be found in several multi-objective AutoML solutions, such as TPOT [200] or FEDOT [216].

4.2.4 Relevant software and implementations. Two very established packages which offer EMOAs are PlatEMO[3] and pymoo[4]. They both offer a wide range of EMOAs and can be generally recommended. The mle-hyperopt[5] package offers directly MOHPO via NSGA-II using internally the nevergrad[6] package.

4.3 Model-based optimization

4.3.1 Bayesian Optimization. In the following, the basic concepts of Bayesian optimization (BO) are shown. For more detailed information, see [30]. BO has become increasingly popular as a global optimization technique for expensive black-box functions, and specifically for HPO [125, 139, 239]. BO is an iterative algorithm with the key strategy of modelling the mapping $\boldsymbol{\lambda} \mapsto c(\boldsymbol{\lambda})$ based on observed performance values found in the archive $\mathcal{A}$ via (non-linear) regression. This approximating model is called a *surrogate model.* Typical choices for the surrogate model are Gaussian processes (GPs) or random forests. BO starts on an archive $\mathcal{A}$ of evaluated configurations, typically sampled randomly, using e.g., Latin Hypercube Sampling or Sobol sampling [34]. BO then uses the archive to fit the surrogate model, which for each $\boldsymbol{\lambda}$ produces both an estimate of performance $\hat{c}(\boldsymbol{\lambda})$ as well as an estimate of prediction uncertainty $\hat{\sigma}(\boldsymbol{\lambda})$, which then gives rise to a predictive distribution for each possible HPC. Based on the predictive distribution, BO computes a cheap-to-evaluate acquisition function $u(\boldsymbol{\lambda})$ that encodes a trade-off between *exploitation* and *exploration* where exploitation favours solutions with high predicted performance, while exploration favours solutions with high uncertainty of the surrogate model because the surrounding area has not been explored sufficiently yet. Instead of directly looking for the optimum of the expensive objective, the acquisition function

[3]https://github.com/BIMK/PlatEMO
[4]https://github.com/anyoptimization/pymoo
[5]https://github.com/mle-infrastructure/mle-hyperopt
[6]https://github.com/facebookresearch/nevergrad

$u(\boldsymbol{\lambda})$ is optimized in order to identify a new candidate $\boldsymbol{\lambda}^+$ for evaluation. The true objective value $c(\boldsymbol{\lambda}^+)$ of the proposed HPC $\boldsymbol{\lambda}^+$ – generated by optimization of $u(\boldsymbol{\lambda})$ – is finally evaluated and added to the archive $\mathcal{A}$. The surrogate model is updated, and BO iterates until a predefined termination criterion is reached.

Simple acquisition functions. A very popular acquisition function is the *expected improvement* (EI) [139]). EI was introduced in connection with GPs that have a Bayesian interpretation, expressing the posterior distribution of the true performance value given already observed values as a Gaussian random variable $C(\boldsymbol{\lambda})$ with $C(\boldsymbol{\lambda}) \sim \mathcal{N}(\hat{c}(\boldsymbol{\lambda}), \hat{\sigma}(\boldsymbol{\lambda})^2)$. A further, very simple acquisition function is the *lower confidence bound* (LCB) [138]. The LCB treats local uncertainty as an additive bonus at each $\boldsymbol{\lambda}$ to enforce exploration, which can be controlled with a control parameter κ.

4.3.2 Multi-objective Bayesian Optimization. The previously presented BO framework can be extended to simultaneously optimize a set of possibly conflicting black-boxes. In particular, the multi-objective extensions to the BO framework (MO-BO) can be categorized into two categories as shown in Figure 5.

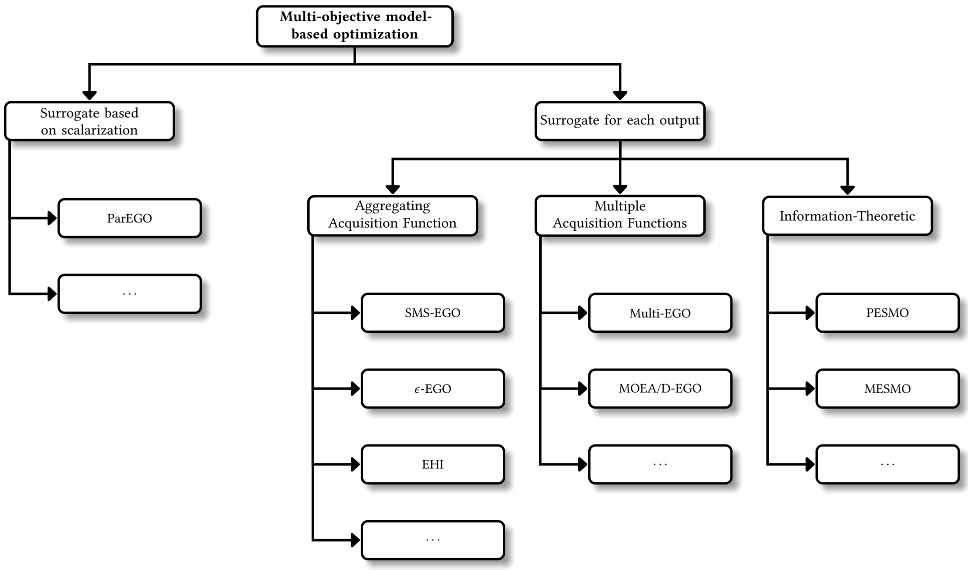

Multi-objective model-based optimization
Surrogate based on scalarization
Surrogate for each output
ParEGO
...
Aggregating Acquisition Function
Multiple Acquisition Functions
Information-Theoretic
SMS-EGO
ε-EGO
EHI
...
Multi-EGO
MOEA/D-EGO
...
PESMO
MESMO
...

Fig. 5. Different multi-objective model-based optimization approaches.

A simple approach is to create a new objective by scalarizing the multi-dimensional output of $c(\boldsymbol{\lambda})$ to a single value (see Section 4.1.1). Some MO-BO methods utilize one or more scalarizations of the MOO problem to fit surrogates in order to approximate the Pareto front. Algorithms that do not use scalarization of the outcomes instead train independent surrogates for each output dimension. By having a prediction for each output dimension, we can either obtain an acquisition function for each dimension and use a multi-objective optimizer to obtain a set of promising configurations, or we can build an acquisition function that aggregates the predictions for each dimension into a single-objective acquisition function. In the following, we discuss some of these approaches in more detail.

Scalarization and ParEGO . ParEGO is a scalarization-based extension of BO to MOO problems proposed by Knowles [149]. This method is built on the assumption that the Pareto front can be approximated by solving a number of single-objective problems that are scalarized variants of the m objective functions (see Section 4.1.1). In ParEGO, we proceed as follows: First, we must determine a set of scalarization weights that will be used throughout all BO-iterations. This set of weights should ensure that the Pareto front is explored evenly. Therefore, we create the set of all possible weight vectors using the following rule:

$$\left\{ \boldsymbol{\alpha} = (\alpha_1, \alpha_2, ..., \alpha_m) \,\middle|\, \sum_{j=1}^{m} \alpha_j = 1 \ \wedge \ \alpha_j = \frac{l}{s}, l \in \{0, 1, ..., s\} \right\}, \tag{12}$$

which generates $\binom{s+m-1}{m-1}$ different weight vectors. Second, the output space of each of our m objectives is normalized to $[0, 1]$. Finally, in each iteration of ParEGO, we create the scalarized objective using the so-called augmented Tchebycheff function [149]:

$$c_{\boldsymbol{\alpha}}(\boldsymbol{\lambda}) = \max_{j \in \{1,...,m\}} \left(\alpha_j c_j(\boldsymbol{\lambda})\right) + \rho[\boldsymbol{\alpha} \cdot c(\boldsymbol{\lambda})], \tag{13}$$

where ρ is a small positive constant and $\boldsymbol{\alpha}$ is a weight vector drawn uniformly from the set in (12). Instead of only using the linear second term, the first term with the Tschebycheff norm is added to ensure that we are able to find a solution in the non-convex parts of the Pareto front. In each iteration, a surrogate model is fitted on the design with scalarized outcomes $\left\{\left(\boldsymbol{\lambda}^{(i)}, c_{\boldsymbol{\alpha}}(\boldsymbol{\lambda}^{(i)})\right)\right\}_i$, and the EI is optimized on this model to propose a new HPC. ParEGO can easily be extended to parallel batch proposals. In order to propose a batch of size q, Horn et al. [116] suggest to sample q different weight vectors per iteration in a stratified manner. New design points are proposed by fitting q surrogates to the q differently scalarized outcomes in the design and optimizing the acquisition function on each of the q surrogate models in parallel. One concern with ParEGO and scalarizing-based solutions in general is that the uniformly sampled weights do not necessarily result in the best distribution of nondominated points. An advantage of ParEGO is that it can be easily adapted to focus the search on one objective by limiting the maximum weights of the others.

EHI. (or EHVI) Emmerich et al. [82] propose to use the expected improvement over the S-metric (i.e., hypervolume) as an acquisition function for MO-BO. Here, a surrogate model for each objective is fitted individually. The EHI is then calculated as the expectation of the hypervolume improvement over the distribution of outcomes as predicted by the surrogate models. A drawback of hypervolume-based BO is that a non-trivial multidimensional integral must be evaluated to calculate the expectation of hypervolume improvement. It is possible to use Monte-Carlo-based approximations [81], and the KMAC method [262] to efficiently calculate the EHI criterion with complexity $O(n \log n)$ in three dimensions and $O(n^{\lfloor m/2 \rfloor})$ in m dimensions. Additionally, Emmerich et al. [83] propose a more efficient means of calculating the EHI with a complexity of $O(n \log n)$ for $m = 2$. However, these methods are significantly more complex than the other presented MO-BO infill criteria. To obtain batch-proposals, Yang et al. [263] propose dividing the objective space into several sub-objective spaces and then search for the optimal solutions in each sub-objective space by using a truncated EHI.

SMS-EGO. The $\mathcal{S}$-Metric Selection-based Efficient Global Optimization (SMS-EGO) algorithm is another popular extension of MO-BO. This method was proposed by Ponweiser et al. [217] and extends the idea of the EHI by employing an infill criterion: In each BO iteration, SMS-EGO approximates each of the m objectives with a separate surrogate model. For each objective, we compute the LCB and denote the resulting m-dimensional outcome with $\boldsymbol{u}_{\text{LCB}}$. The desirability of a configuration $\boldsymbol{\lambda}$ is derived from the increment of the dominated hypervolume when $\boldsymbol{u}_{\text{LCB}}(\boldsymbol{\lambda})$ is added to the current Pareto front approximation $\hat{\mathcal{P}}$:

$$u_{\text{SMS}}(\boldsymbol{\lambda}) = \text{HV}_{\boldsymbol{r}}\left(\hat{\mathcal{P}} \cup \boldsymbol{u}_{\text{LCB}}(\boldsymbol{\lambda})\right) - \text{HV}_{\boldsymbol{r}}(\hat{\mathcal{P}}) - p, \tag{14}$$

with a penalty p and with the reference point $\boldsymbol{r}$ chosen as $\max(\hat{\mathcal{P}}) + \mathbf{1}_m$ (under the assumption that all objectives are to be minimized). The penalty p increases the worse $c(\boldsymbol{\lambda})$ is compared to the current Pareto front. To prevent the selection of configurations close to $\hat{\mathcal{P}}$, SMS-EGO assigns also a penalty to points within an ϵ-range of $\hat{\mathcal{P}}$. However, the penalty is in this case only influenced by the objectives inferior to the approximated Pareto front. If $\boldsymbol{u}_{\text{LCB}}(\boldsymbol{\lambda})$ is a dominated solution, $u_{\text{SMS}}(\boldsymbol{\lambda})$ would be zero without the penalty term p, making the optimization of the acquisition function more challenging. Therefore, to guide the search towards non-dominated solutions in areas of dominated solutions, a penalty p is added for each point that dominates the solution candidate. Otherwise, if $\boldsymbol{u}_{\text{LCB}}(\boldsymbol{\lambda})$ is a non-dominated solution, the penalty is zero.

Multi-EGO. In each BO iteration, Multi-EGO approximates each of the m objectives with a separate surrogate model from which a single-objective acquisition function is obtained. Jeong and Obayashi [133] use Gaussian processes for the surrogates and EI as the individual acquisition functions. The m acquisition functions are optimized jointly as an MOO problem itself using a multi-objective genetic algorithm (in principal any MOEA can be used here), resulting in a set of candidates with non-dominated acquisition function values. From this set, Multi-EGO then selects multiple points to be evaluated. This naturally lends to parallelization, as there are always multiple proposals generated in each iteration.

MESMO and PESMO. Common information-theoretic acquisition functions are multi-objective maximum entropy search (MESMO) and predictive entropy searc (PESMO) proposed by Belakaria et al. [21] and Hernández-Lobato et al. [110] respectively. Information theoretic multi-objective acquisition functions model each black-box with an independent surrogate model. The resulting Pareto set is modelled as a random variable. Hence, we can compute the entropy of the location of the Pareto set. The lower this entropy is, the more we know about the location of the Pareto set. These acquisition functions represent the expected reduction of entropy if a point is evaluated. Information theoretic acquisition functions are linear combinations of the expected entropy of the whole predictive distributions of every black-box. Hence, the utility of every point represents a global measure of the uncertainty of the input space and not a local and heuristic measure of the particular point, as in other acquisition functions. Thus, in theory they should explore the search space more efficiently [21, 110].

4.3.3 Extensions and open challenges of Bayesian Optimization. So far, we have seen the classical multi-objective scenario where a set of black-box functions is simultaneously optimized. However, there are extensions of BO to a wider range of MOO scenarios. MOO scenarios can, for example, be constrained. Such constraints, as mentioned in the previous section, might also be black-boxes that can be approximated by surrogate models. An example of such a scenario is the simultaneous optimization of the prediction error and time of prediction of a deep neural network constrained to some particular physical storage size required for its implementation on a chip. Another interesting and unsolved problem is to model the dependencies of various black-boxes. Intuitively, if expert knowledge suggests that there is a correlation between the black-boxes, we could infer the value of one black box by knowing values of another black-box. For example, if we are simultaneously optimizing the prediction error and prediction speed of a deep neural network, we can hypothesize that a negative correlation exists between them. By knowing a result with a low prediction error, it may be likely that prediction speed is slow. This could possibly allow forgoing evaluation of that black-box, especially if it will incur a significant loss of computational (or other) resources. Modeling independent surrogates for every black-box does not take into account potential correlations. We could model these dependencies by using a multi-output GP and a specific acquisition function that considers the information computed by the multi-output GP [172, 196]. Furthermore, the structure of the search space of HPO problems raises challenges to be addressed by optimizers. In particular in AutoML and NAS, the search spaces are often characterized by high dimensionality, the presence of both numerical and categorical variables, and hierarchies between hyperparameters. Modeling the relationship between HPs and model performance is particularly challenging in such scenarios. These issues have been increasingly discussed for single-objective HPO. To efficiently optimize over high-dimensional spaces, Wang et al. [257] propose random embeddings. To accommodate mixed-hierarchical spaces, there are several approaches reaching from GPs with special kernel functions [162] to other model classes that are inherently capable of representing mixed-hierarchical hyperparameters by linear models [64], random forests [125], or kernel density estimation [25]. These challenges equally apply to the MOO setting, and need to be addressed accordingly in future research. Modelling the causal relations of the input space variables in MOO potentially reduces the size of the input space [4]. Some optimization problems involve a mix of expensive black-boxes and cheap ones. In order to solve these scenarios, a hybrid methodology between decoupled MO-BO and metaheuristics can be suggested. In particular, cheaper black-boxes could be optimized with metaheuristics and more expensive ones with BO. The noise of the black-boxes can also vary across the input space. Other ideas are to extend automatic BO, i.e., approaches which try to adapt the control parameters of BO itself automatically, to the multi-objective setting [186] or implement asynchronous multi-objective BO to never leave any black-box or resource idle. Finally, non-myopic BO can be extended to the multi-objective setting [134].

4.3.4 Relevant software and implementations. Table 1 compares BO frameworks with multi objective capabilities. We identified as suitable well-established frameworks *Dragonfly* [142], *HyperMapper* [197], *Openbox* [168], *Ax*[7], *trieste*[8], BoTorch [15] and GPflowOpt [152]. We checked if they offer scalarization-based approaches (e.g., ParEGO), EHVI-based approaches (e.g., EHVI, QNEHVI) or MESMO, and if they can handle constraints and use computational resources in parallel.

Table 1. BO frameworks which offer MO support

	Dragonfly	HyperMapper	OpenBox	Ax	trieste	BoTorch	GPflowOpt
Scalarization-based	✓	✓	✓	✓	✓	✓	×
EHVI-based	×	×	✓	✓	✓	✓	✓
MESMO	×	×	✓	×	×	×	×
Constraints	×	✓	✓	✓	✓	✓	×
Parallel	✓	✓	✓	✓	✓	✓	×

4.4 Multi-fidelity optimization

Assuming the existence of cheaper approximate functions, multi-fidelity optimization is a popular choice for expensive-to-evaluate black-box optimization target functions in many application domains like aerodynamics [93] or industrial design [119]. These approximations tend to be noisy or less accurate in general, but present a cheaper option to estimate an evaluation of the original function, when a multitude of evaluations are infeasible due to the incurred computational cost. Multi-fidelity optimization has been established for single-objective hyperparameter tuning [164] and has become a desirable mode of optimization especially for deep learning models, that are very costly to fully train and evaluate [85, 164, 165]. Instead of optimizing configuration selection,

[7] https://ax.dev/

[8] https://secondmind-labs.github.io/trieste/

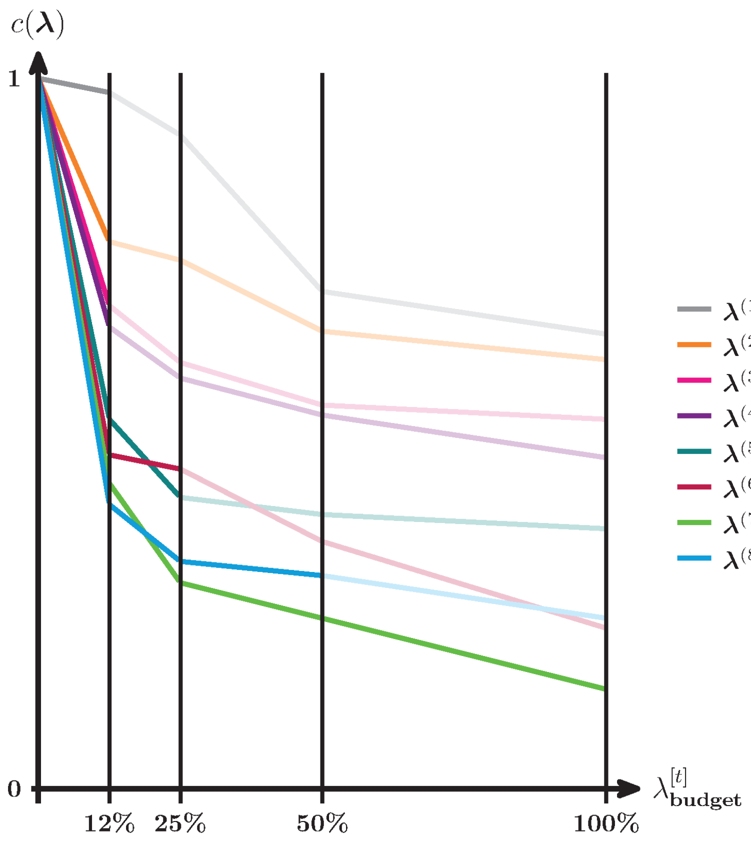


Fig. 6. Exemplary bracket run (figure inspired by Hutter et al. [126]). Faint lines represent future performance of HPCs that were discarded early.

as in model-based optimization, multi-fidelity methods for HPO aim at optimizing configuration evaluation, i.e., allocate resources to the different configurations chosen for evaluation in an efficient manner. In contrast to other applications, where one might have access to only one or few fidelities (e.g., one cheap computer simulation for an expensive aerodynamics optimization problem), ML presents a continuum of fidelities to choose from. Typical examples for this are the number of epochs a deep learning model is trained, how much training data is provided for model training or resolution of input images in computer vision. The idea is to allocate more resources (i.e., additional epochs, increased training data etc.) to promising configurations while discarding worse performing configurations without sacrificing a lot of resources. This is achieved by starting all configurations on a lower fidelity (i.e., less resources) and then promoting only the well-performing configurations to a higher fidelity. This idea is shown in Figure 6. Essentially, multi-fidelity approaches for single-criteria HPO were originally intended to build on top of the already quite competitive random search with improved resource allocation. These approaches have since been enhanced by the use of model-based optimization for drawing configurations instead of random sampling [85] and asynchronous execution [148]. The same holds true in the multi-objective setting as recent works have shown [230, 231]. Multi-fidelity extensions like Hyperband can be carried over to the MO setting by simply defining a suitable performance indicator to decide where increased resource allocation is desirable, which could be achieved in numerous ways. Existing approaches have done this via scalarization using random weights [230] or non-dominated sorting [227, 231]. Taking multi-fidelity one step further, it can also be combined with BO, which remains one of the open challenges for MOO: This has been done in single-objective HPO, where configurations are no longer sampled randomly (as in traditional Hyperband), but via BO [85]. Therefore, instead of enhancing random search, multi-fidelity methods are used to enhance BO in this case [85].

4.5 Further issues

4.5.1 Focusing optimization through user preferences. A growing body of literature is exploring the integration of decision maker (DM) preferences into multi-objective optimization [36, 256]. Depending on when the user preferences are elicited relative to the optimization process, these methods are generally divided into *a priori* (before optimization), *progressive* (during optimization), and *a posteriori* (after optimization) approaches [37]. The majority of literature on multi-objective optimization aims to find a good approximation of the entire Pareto front, providing the decision maker with a variety of alternatives to choose from *after* optimization, so falls into the *a posteriori* category. There are, however, at last three reasons for taking preference information into account earlier [36]:

(1) It will allow to provide the DM with a more relevant sample of Pareto optimal alternatives. This could either be a smaller set of only the most relevant (to the DM) alternatives, or a more fine-grained resolution of the most relevant parts of the Pareto frontier.
(2) By focusing the search onto the relevant part of the search space, we expect the optimization algorithm to find these solutions more quickly. This is particularly important in computationally expensive applications such as hyperparameter optimization.
(3) As the number of objectives increases, it becomes more and more difficult to create an approximation to the complete Pareto optimal frontier. This is partly because of the increasing number of Pareto optimal solutions, but also because with an increasing number of objectives, almost all solutions in a random sample of solutions become non-dominated [130], rendering dominance as selection criterion less useful. DM preference information can re-introduce the necessary order relation.

Several ways to specify preferences have been proposed, including reference points (an "ideal" solution), constraints (minimum acceptable qualities for each objective), maximal/minimal trade-offs (how much is the DM willing to sacrifice at most in one criterion to improve another criterion by one unit) and desirability functions (non-linear scaling of each objective to $[0, 1]$). Rather than asking the DM to specify preferences explicitly, preferences can also be learned [95] by asking the DM to rank pairs or small sets of solutions, or to pick the most preferred solution from a set. This has the advantage that the DM just has to compare *solutions*, which is something they should be comfortable doing. Finally, people have observed that DMs often select a solution from the Pareto frontier that “sticks out" in the sense that improving it slightly in either objective would lead to a significant deterioration in the other. These solutions are often called "knees", and it is possible to specifically search for them without having to ask the DM anything (e.g., [38]).

4.5.2 Noisy environments. If an MOO problem is noisy, we do not have direct access to the objective values $c_1, \dots, c_m$, but instead we only have measurements $\tilde{c}_1, \dots, \tilde{c}_m$ with

$$\tilde{c}_i = c_i + \epsilon_i \quad \forall i = 1, \dots, m \tag{15}$$

where ϵ_i is the observational noise in the i-th objective modeled as a random variable. Noise in the evaluation plays a major role also in most MOHPO problems, e.g., because generalization error estimates in Eq. (2) are based on a finite dataset sampled from a much larger universe, the random sequence of the data as it is presented during training has an impact on the performance, or a stochastic optimizer is used during training. Clearly, noise is challenging for optimization, whether single-objective or multi-objective. It may lead to false performance comparisons such as for example an incorrectly inferred dominance relationship between two HPCs. This may mean some dominated solutions are classified as non-dominated and thus incorrectly returned by the algorithm, while some Pareto-optimal solutions are discarded incorrectly because they are perceived as being dominated. It may also lead to an over-optimistic estimate of the Pareto frontier, as most solutions returned as non-dominated were “lucky" in their evaluation. One simple way to reduce the effect of noise is to evaluate each solution multiple times and optimize based on mean values. While this reduces the standard error, it is computationally very expensive and may be impractical for HPO. Researchers in the evolutionary computation community have developed a wealth of methods to cope with noisy evaluations, including the use of statistical tests (e.g., Syberfeldt et al. [243], Park and Ryu [205]), the use of surrogate models (e.g., Branke et al. [40]), probabilistic dominance [122], or integrating statistical ranking and selection techniques [161]. A relatively simple yet effective method seems to be the rolling tide EA [91] which alternates between sampling new candidates and refining the archive, i.e., re-evaluating promising HPCs, in each optimization iteration. For BO, the noise can be accounted for by using re-interpolation [153] or in a straightforward way by choosing GP regression (rather than interpolation) and appropriate acquisition functions, see, e.g., Astudillo and Frazier [11], Daulton et al. [62], Hernández-Lobato et al. [110], Horn et al. [115], Knowles et al. [150], Rojas Gonzalez et al. [223], Rojas-Gonzalez and Van Nieuwenhuyse [224]. Also heteroscedastic models to deal with input space noise in the presence of several objectives like the models presented in Villacampa-Calvo et al. [253] can be used for BO. A recent survey on multi-objective optimization methods under noise with a provable convergence to a local non-dominated set has been provided by Hunter et al. [123], older surveys with a somewhat broader scope can be found in Gutjahr and Pichler [103], Jin and Branke [136].

4.5.3 Realistic evaluation of multi-objective optimization methods. When applying MOHPO in a real world setting it is crucial to understand how a solution will behave on unseen data after deployment. In standard HPO this is achieved via a 3-way split of the data in train, validation and test sets, or, more generally, as nested resampling (c.f. Section 3). In Section 3.3.2 different measures for the quality of multi-objective optimization have been introduced. While these measures are generally useful to evaluate the performance over the whole objective space, decision makers are ultimately only interested in a single solution (selected from the Pareto set) for deployment and later use. This usually implies a human-in-the-loop. For a single train/validation/test split, this is easily achievable: The decision maker needs to look at the Pareto front computed on the validation set, choose the configuration they would like to use and then evaluate its performance on the test set. Extending this approach to nested resampling, multiple Pareto fronts are generated, one for each outer fold. Here, for a drill-down to to a single solution would imply that the decision maker needs to make these choices for each outer loop, which can become impractical and can make larger benchmark studies difficult to conduct efficiently. What can be implemented in an automatic fashion, is the evaluation of each generated Pareto front on its associated outer test set in an unbiased fashion. Here, similarly as in nested resampling in single objective HPO, each Pareto set candidate would be trained on the joint training and validation set, and evaluated on the test set. This results in a new unbiased Pareto front, for each outer iteration, and measures like hypervolume can also be calculated from the outer results in an unbiased fashion. Furthermore, attainment surfaces [92] have been introduced to study the performance of general multi-objective noisy optimization by visualizing the Pareto fronts of multiple runs simultaneously. They can be used to either visualize the set of all returned fronts evaluated on the validation sets, or, as described above, on the outer test sets. In general, proper MOHPO evaluation is understudied and an open challenge for further research.

4.6 Relevant benchmarks and results

In general, relevant benchmarks comparing different multi-objective optimizers for HPO mostly were conducted in the context of new optimizers being proposed (see, e.g., Guerrero-Viu et al. 99, Hernández-Lobato et al. 110, Schmucker et al. 230). We start by giving a brief summary of interesting findings in the literature. The experiments conducted in Horn [113] reveal the weaknesses of utilizing grid search in the context of multi-objective HPO when tuning hyperparameters of an SVM for binary classification (classification error and training time as targets) compared to sequential model-based optimization techniques in the sense that grid search fails to provide a good Pareto front approximation. Random search on the contrary has shown good results and even outperformed some model-based optimization methods (i.e., ParEGO, SMS-EGO and PESMO) when applied to a fairness-related multi-objective HPO task [230]. While an exhaustive benchmark comparing the performance of random and grid search on multi-objective HPO problems is missing at this point in time, random search can generally be recommended as a baseline when conducting experiments [230]. Horn and Bischl [114] found that ParEGO and SMS-EGO were able to outperform an NSGA-II variant and latin hypercube-sampling when tuning over multiple models with hierarchical structure in binary classification tasks (9 datasets) where the objectives have been FPR and FNR. In Horn et al. [115] the authors investigated the performance of noise-resistant SMS-EGO variants, the rolling tide EA (RTEA) and approaches based on random search while tuning an SVM on 9 binary classification tasks with FPR and FNR as objectives. Here, RTEA and a repeated variant of SMS-EGO were ranked best among all competitors. Hernández-Lobato et al. [110] compared PESMO to EHVI, ParEGO and SMS-EGO when tuning the hyperparameters of a feed-forward neural network on the MNIST dataset with classification error and prediction time as objectives. Here, PESMO outperformed all competitors, especially if few evaluations were available. Guerrero-Viu et al. [99] conducted a NAS+HPO benchmark tuning the architecture and hyperparameters of a CNN on classification tasks using the Oxford Flowers and MNIST Fashion datasets. They propose an EMOA extended by successive halving, a multi-objective BOHB variant, an EHVI variant, a multi-objective extension of BANANAS [258] and a BULK & CUT optimizer which first tries to find large and accurate models in the beginning and subsequently prunes them. In general, the EHVI variant and BULK & CUT optimizer slightly outperformed the other approaches [214]. Recent standardized HPO benchmark suites like *HPOBench* [76] and *YAHPO Gym* [214] include some support for multi-objective use cases, which emphasizes the trend towards MOHPO in the research community. In [214], the authors present *YAHPO Gym*, a benchmark suite for HPO, and illustrate the potential of *YAHPO Gym*'s MOHPO benchmark problems by comparing seven multi-objective optimizers on the YAHPO-MO benchmark (v1.0), a collection of 25 MOHPO problems given by tuning hyperparameters of an elastic net, random forest, gradient boosting tree, decision tree, funnel-shaped MLP net, or an ML pipeline on various OpenML tasks. The number of objectives ranges from two to four. Objectives are given by at least one performance metric (e.g., accuracy), and interpretability metrics (such as the number of features used or the interaction strength of features as introduced in Section 5) or metrics regarding computational efficiency such as memory required to store the model. The optimization budget (number of evaluations) is scaled with respect to the search space dimensionality and ranges from 77 to 267 objective evaluations. [214] compare random search, random search (x4), ParEGO, SMS-EGO, EHVI, Multi-EGO and a mixed integer evolution strategy (MIES; Li et al. 167) relying on Gaussian and discrete uniform mutation and uniform crossover. All model-based optimizers use random forests as surrogate models, for more details see [214]. Random search (x4) samples four configurations uniformly at random (in parallel) at each optimizer iteration. Regarding performance evaluation the anytime hypervolume indicator (computed on normalized objectives) was considered. Looking at mean ranks

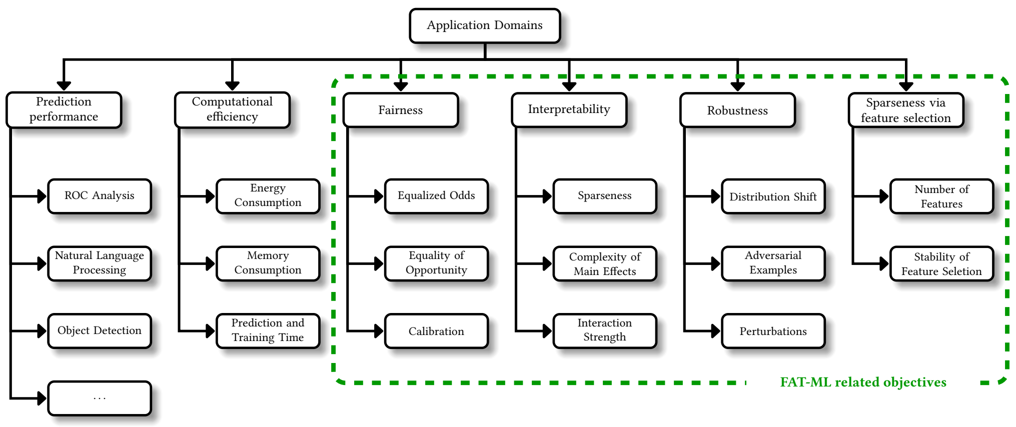


Fig. 7. Overview of application scenarios for MOHPO.

over all benchmark problems, random search (x4) outperforms all competitors after having used 25% of the optimization budget (which was expected due to random search (x4) having evaluated four times more configurations in total compared to the competitors). However, with respect to final performance, Multi-EGO, ParEGO and MIES are on par with or outperform random search (x4), while EHVI or SMS-EGO fail to consistently outperform the vanilla random search. Looking at each benchmark problem separately, results indicate strong performance differences of optimizers with respect to the hypervolume indicator. Especially MIES performs exceptionally well on some benchmark problems but only shows average performance on others. While this benchmark study is a first step towards an exhaustive multi-objective benchmark, plenty of work still needs to be done to be able to give general recommendations regarding when to use which multi-objective optimizer.

5 OBJECTIVES AND APPLICATIONS

In the following, we will give an overview of relevant metrics for ML models, interesting use cases and application domains for MOHPO. We organize this section by examining three aspects of ML model evaluation:

Prediction performance. In most cases, prediction performance is of primary importance. Which performance metric aligns best with the goals and costs associated with an ML task is not always readily apparent, especially if misprediction costs are hard to quantify or even unknown. We discuss the case where multiple performance metrics with unquantifiable trade-offs are relevant by the example of ROC analysis in classification.

Computational efficiency. Computational efficiency is a prime example for the difficulty of finding an appropriate metric. The desire for a memory or energy efficient model is often difficult to operationalize because it is not as straightforward to measure model efficiency. We will examine various possible ways and present a usecase where efficiency and prediction performance are optimized simultaneously.

Fairness, interpretability, robustness and sparseness. Many applications areas require ML models to fulfill higher standards than only high predictive performance. First, we explore Fairness and Interpretability as objectives. We further examine Robustness of ML models to domain shift, perturbations and adversarial attacks as a prerequisite/objective, as robust models inspire a higher degree of trust. Finally we will venture into the topic of sparse ML models, where the number of features is minimized in some manner. Sparseness itself is not necessarily a desirable quality, but may be a suitable proxy for model complexity/interpretability, data acquisition cost and/or even performance. The term FAT-ML (Fairness, Accountability and Transparency in Machine Learning) has been coined and has gained some traction recently[9]. There is considerable overlap between the objectives examined here and FAT-ML, but FAT-ML also entails external characteristics, such as e.g., responsibility which is encompassed in the accountability aspect.

A visual representation of the remaining sections can be found in Figure 7. It should be noted that interactions between different objectives are worth studying. Objectives may be positively correlated, for example a model with high predictive accuracy might be expected to also perform well measured in other performance metrics like AUC [49]. Similarly, a sparse model is expected to be more efficient as well as more interpretable. Objectives might also be conflicting. A more complex model might for example perform better in terms of accuracy, but might be less interpretable. It is hard to quantify these relationships without a comprehensive benchmark, but we will try to shed light on them wherever relevant.

5.1 Prediction performance

HPO has traditionally been focusing on optimizing for predictive performance measured by a single performance metric. It can, however, also be beneficial to optimize multiple different prediction metrics simultaneously; particularly, if a trade-off between different metrics cannot be specified *a priori*. An abundance of prediction metrics have been defined for various ML tasks and the appropriate choice depends on the specific use case and available data and we will therefore not provide a comprehensive overview. Good examples for the diversity in metrics can be found for classification, one of the most widely utilized ML tasks, in Figure 3.1 of Japkowicz and Shah [132] and for time series forecasting, a much more specific application, in Figure 2.3 of Svetunkov [242]. Different performance metrics may penalize prediction errors in distinct ways, and therefore prediction metrics are more or less correlated [49, 268]. These correlations have been extensively studied for some (widely used) metric pairs [58, 63, 121]. Here, as in general MOHPO, we argue it may be of merit to apply MOHPO in a case where the user is interested in two or more, possibly conflicting prediction metrics and/or trade-offs.

5.1.1 ROC analysis. Many binary classification models predict scores or probabilities that are then converted to predicted classes by applying a decision threshold. Different decision threshold values will result in different trade-offs between performance measures. A higher decision threshold may for example reduce the number of false positives, but may also reduce the number of true positives at the same time. This trade-off is often visualized by the receiver operating curve (ROC), where true positive rate (TPR) is plotted against false positive rate (FPR) for different threshold values. Similarly, the precision-recall curve visualizes TPR versus positive predicted value (PPV), respectively. Because improving one classification metric by varying the decision threshold is typically associated with deteriorating performance with respect to the other, choosing a decision threshold based on a ROC analysis a fundamentally multi-objective problem, where the decision threshold can be viewed as an additional hyperparameter. As the decision threshold has no impact on the model itself, it can be optimized post-hoc and tuning is therefore fairly cheap [30]. An approach to address this optimization problem is to formulate it as a single-objective problem by e.g., aggregating elements of the confusion matrix (*F-Measure*) or the ROC curve (AUC) into a single metric. As a result, by reducing the natural two-objective optimization problem to a single one, one only optimizes a classifier's general ability to discriminate both classes. However, not

[9]See e.g., https://facctconference.org/ or https://www.fatml.org/.

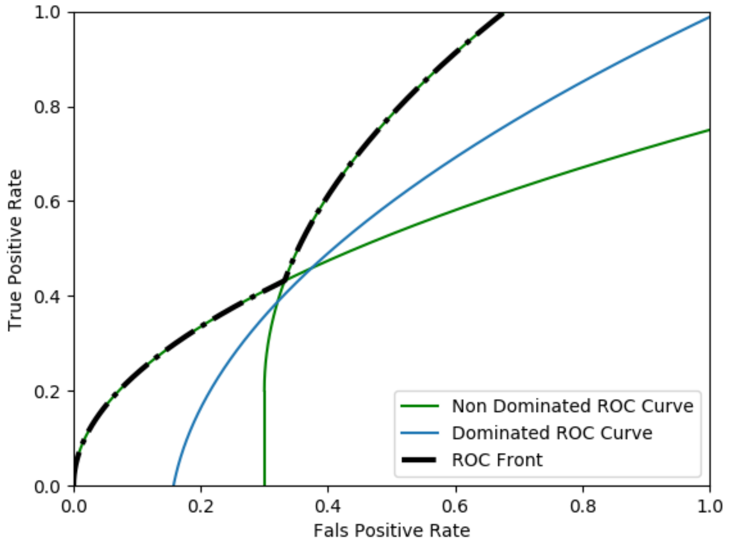


Fig. 8. Scheme showing two intersecting, non-dominant ROC curves with similar AUC in green and one ROC curve that is fully dominated in blue. The resulting Pareto front is marked through a dashed black line.

all information is preserved. An example can be seen in Figure 8, where two ROC curves lead to a similar AUC but present quite distinct shapes. It may be desirable to consider the ROC curves for different HP configurations with different ROC curves for a final solution. One can follow the approaches in e.g., Lévesque et al. [163], Chatelain et al. [52], and Bernard et al. [27] and combine the information from each iteration in one ROC front that displays all Pareto-optimal trade-offs between TPR and FPR. This preserves all relevant information from individual evaluations and allows for decisions in accordance to user- or case-specific preferences. In our example in Figure 8, the final combined model would then show the same performance, i.e., trade-offs, as the dashed black line. A similar approach has been introduced for regression: The Regression Error Characteristic (REC) Curves, which examines trade-offs between error tolerance and percentage of points within the tolerance [28].

Extending these ideas to a multi-class setting, where each class has unique associated misclassification costs, brings new challenges. The number of different misclassification errors grows quadratically with the number of classes g, leading to increasingly high-dimensional surfaces. Naturally, one wants to minimize all the different misclassification errors simultaneously. An in-depth exploration of this challenge and how to solve it can be found in Everson and Fieldsend [84]: The authors define the ROC surface for a g-class classification problem and try to optimize for the respective $g(g-1)$ misclassification rates by an EA. Multi-class ROC problems are therefore generally seen as a MOO problem and closely related to the core topic of this work [84, 90]. They are generally solved through (i) single-model approaches, where one classifier is identified, and once the costs are known at prediction time, a suitable trade-off on that classifier's ROC surface is then found, or (ii) multi-model approaches that produce a pool of suitable classifiers that are available at prediction time [27]. The optimal ROC surface can be viewed as a Pareto front; in this case the search space is the space of classifiers, and every classifier corresponds to a ROC curve. A classifier is non-dominated if any part of its ROC curve is non-dominated and the length of the non-dominated part of the ROC can be used as the crowding distance. It should be noted that the principle of AUC as a metric of the classifier's general ability to discriminate both classes in the binary case does not carry over to the multi-class setting [75]. Hand and Till [105] attempt to generalize and adapt AUC from a binary setting to a meaningful metric for the multi-class setting. In binary classification, the ROC curve is generated through varying the decision threshold, which is quite cheap and can be done after each iteration in a hyperparameter optimization problem. In a g-class setting during prediction, the classifier will generally output a vector of probabilities or some measure of confidence that a sample belongs to the corresponding class c_i Bernard et al. [27]:

$$h(x) = [h(c_1|x), h(c_2|x), \ldots, h(c_g|x)]$$

To generate a decision rule, a vector of weights $w = (w_1, w_2, ..., w_g)$ is applied, and the highest value is chosen. To obtain a ROC surface similar to the ROC curve in binary classification, a large number of different weight vectors would have to be evaluated, which incurs larger additional costs and makes it no longer trivial to obtain post-hoc for each configuration [27].

Applications and exemplary use case. An illustrative example is given by Chatelain et al. [52], who attempt to identify well-performing configurations of SVMs in a binary classification setting with unknown misclassification costs. Their use-case is based on digit recognition from incoming handwritten mail document images. While shown to be effective at similar tasks [198], SVMs are notorious for extreme effects of hyperparameters on performance [52]. Chatelain et al. [52] introduce two parameters C_- and C_+ as penalties for misclassifying the respective classes (namely, digit vs. non-digit) and use them during training of the SVM. They tune for these two hyperparameters along with γ, the kernel parameter for the radial basis function kernel. Using NSGA-II (see Section 4.2.3), they evolve a pool of non-dominated hyperparameter configurations, thus approximating the Pareto optimal set. Another application can be found in Horn and Bischl [114], where *ParEGO* and *SMS-EGO* (see Section 4.3) have been applied to jointly minimize the false-negative rate and false-positive rate of an SVM on a variety of binary classification tasks from different domains.

5.1.2 Other applications and examples for multiple prediction performance metrics. This section outlines a few examples where trade-offs between several prediction performance metrics are examined and MOHPO is applied accordingly.

Natural language processing. Language and human speech in general are quite complex and in turn natural language processing (NLP) tasks are notoriously hard to evaluate. One such example is the recent subfield of natural language generation (NLG): The quality of the resulting texts has to be quantified, but may depend on the use case and different aspects such as semantics, syntax, lexical overlap, fluency, coherency [143]. Sai et al. [225] show that across several popular metrics for NLG tasks, no single metric correlates with all desirable aspects. In a related application of MOHPO, Schmucker et al. [231] use successive halving to optimize perplexity and word error rate for transformer based language models as one of their application examples.

Object detection. Given an image, object detection is used to determine whether there are instances of a given type and where they are located [174]. This introduces aspects of both regression (how close is the proposed bounding box of the object compared to the ground truth?) and classification (are all objects identified correctly?) into the task. Widely used metrics include precision and recall, but this naturally only focuses on one aspect of the task, as one needs to formerly define when something is a true prediction or a false prediction. Liddle et al. [170] propose to specifically examine two aspects of object detection in a multi-objective manner during evaluation: (i) detection rate $DR = \frac{\text{\# correctly located objects}}{\text{\# objects in the image}}$ and (ii) false alarm rate $FAR = \frac{\text{\# falsely reported objects}}{\text{\# objects in the image}}$. Aside from these prediction performance metrics, detection speed is often crucial, which makes evaluation with a single metric even more challenging [174].

5.2 Computational efficiency

Technical constraints have always limited ML research and application, but with the increasing prominence of deep learning, efficiency in ML has become an important topic [228, 244, 245, 255]. This section will not address efficiency of the actual optimization process – be it model search, hyperparameter optimization or NAS [14, 173, 245], which is an active research area in its own right. In the context of this work, we see efficiency as a desirable quality of an ML model with a given HP configuration in terms of computational effort needed for training or prediction[10]. When taking efficiency into account, a common scenario is the existence of resource limitations which have to be respected, e.g., the memory consumption of the model has to be below the available memory to allow for deployment. In these scenarios it may be more useful to the practitioner to formulate a constrained (single-objective or multi-objective, depending on the remaining objectives) optimization problem. Another approach to interpret efficiency in the context of an ML model is *Feature Efficiency*, which will be addressed in Section 5.6. When looking at hardware implementation, we can roughly differentiate between energy-efficient and memory-efficient models. In the following, we will introduce several metrics that can be used to measure a model's efficiency.

5.2.1 Efficiency metrics and approaches. We will in the following give an introduction to three broad approaches to quantifying efficiency in the context of machine learning models and present one related use case each. A comprehensive overview of publications applying MOHPO with at least one objective related to efficiency can be found in Appendix A[11].

Energy consumption and computational complexity. Limiting computational complexity reduces the number of operations performed in a model and therefore generally leads to rather energy-efficient models. Lu et al. [182] give an overview of suitable measures for computational complexity of deep learning models: number of active nodes, number of active connections between the nodes, number of parameters and number of floating-point operations (FLOPs). From experiments, they conclude that FLOPs are the ideal metric and move on to optimize for it in a multi-objective NAS along with accuracy for image classification architectures. This matches with other research papers; FLOPs have long been used in ML – especially in deep learning publications over the last decade – to describe the complexity or even size of a model, for example, in the introductions of ResNet by He et al. [108] or ShuffleNet by Zhang et al. [267]. An alternative to FLOPs is to use Multiply-Accumulate (MAC) operations, but the relationship to FLOPs is roughly linear [118]. Another approach to measure energy consumption is through the use of an appropriate simulator like *Aladdin* as introduced in Shao et al. [235]. It is designed to simulate the energy consumption of NNs given the right information (C code describing the operations performed by the NN) and is used for evaluation of architectures [111, 222].
Example Application - Wang et al. [254] aim to tackle the issue of complex deep learning models for computer vision and the challenges these extremely deep architectures pose for deployment on

[10]The optimization process can become more efficient if it focuses on efficient candidates during optimization. An example of this correlation is the expected improvement per second acquisition function [239] in BO, which often prefers configurations that are quick to evaluate.

[11]While some applications of MOHPO such as prediction performance have too many publications to provide a reasonable and comprehensive overview in the scope of this work, others like e.g., interpretability have only recently been introduced and only few related publications exist. Efficiency in MOHPO is established and at the same time still somewhat novel. We therefore believe this comprehensive review – to the best of our knowledge the first of its kind – to be a worthwile contribution.

e.g., edge devices. They employ a multi-objective approach to identify models suitable for image classification that are not only highly accurate, but also cope with a minimal amount of FLOPs. Convolutional Neural Networks [160] are a popular choice of model for image classification today, and many complex architectures are used for various computer vision tasks. One such example is DenseNet [120] which consists of several blocks of dense layers connected via convolutional layers and pooling layers. Wang et al. [254] uses the the hyperparameters of its dense blocks as the search space for the architecture search. This includes the number of layers and the growth for the dense blocks as well as typical deep learning hyperparameters, such as maximum epochs or learning rate. From this search space, a population is initiated, and Particle Swarm Optimization [237] is used to find a Pareto front. In experiments on the CIFAR-10 dataset, some of the identified models outperform DenseNet-121 and other DenseNet configurations while being less complex with a smaller number of FLOPs.

Model size and memory consumption. Especially in deep learning models, model size and efficiency will often go hand in hand, as employing more parameters (i.e., greater model size) generally results in more FLOPs. The parameters are mostly weights, and their number can be straightforwardly derived in deep learning architectures. Several publications have used the number of parameters as proxy for the efficiency of a deep learning model. For example Howard et al. [117] introduced MobileNets, which are specifically designed for efficient deployment on edge devices. They introduce separable convolutions to reduce the number of parameters needed for a top-performing Convolutional Neural Network.
Example Application - Loni et al. [177] present their framework *DeepMaker* to identify efficient and accurate models for embedded devices. The objectives are classification accuracy and model size (number of trainable network weights); they also discovered a high correlation between the latter and prediction time. NSGA-II is used to search the space of discrete hyperparameters (activation function, number of condense blocks, number of convolution layers per block, learning rate, kernel size and optimizer). Their approach has some similarity to multi-fidelity optimization, as during optimization sampled architectures are only trained for 16 epochs to decide for the optimal choice, but the final performance is reported after training the selected architecture for 300 epochs. The method is tested on MNIST; CIFAR-10 and CIFAR-100.

Prediction and training time. The model is usually trained upon deployment, or the training is not as time-critical, so we are mostly interested in minimizing prediction time. However, some applications and deployment strategies require frequent retraining of the model. In which case, training time can be a crucial factor as well. While often a crucial metric, prediction time is very hard to measure reliably due to various differences in the computing environment [182]. Prediction time may also correlate strongly with energy efficiency metrics, Rajagopal et al. [220] use FLOPs as a proxy for inference latency.
Example Application - PESMO is introduced in Hernández-Lobato et al. [110] and applied to several MOO problems, one of which is finding fast and accurate NNs for image classification on MNIST. They tune a variety of hyperparameters: The number of hidden units per layer (between 50 and 300), the number of layers (between 1 and 3), the learning rate, the amount of dropout, and the level of ℓ_1 and ℓ_2 regularization [110]. The objectives are prediction error and prediction time - measured here as the time it takes for the network to predict 10.000 images. A ratio is then computed with the time the fastest network in the search space requires for this task. PESMO is compared against SMS-EGO, ParEGO and EHI among others and shows superior performance in terms of hypervolume. Their follow-up paper [111] deals with a similar use case.

Dong et al. [72] examine bi-objective NAS for image classification optimizing for classification accuracy and FLOPs, the number of parameters and prediction time respectively. It may sometimes be relevant to optimize a model with two efficiency metrics along with a primary performance metric; examples for this can be found in Chu et al. [55], Elsken et al. [78], Lu et al. [180]. Chu et al. [55] present a framework that allows incorporation of constraints with respect to the three objectives in a task of the super-resolution domain: peak signal-to-noise-ratio or structural similarity index as performance metrics, FLOPs, and the number of model parameters. Note, that this section already delves heavily into the topic of HW-NAS, but only from a multi-objective point of view. Single-objective and constrained approaches [24] are also widespread in addressing these challenges. Benmeziane et al. [24] provide a comprehensive survey on HW-NAS for a reader interested in this subtopic.

In context of the full software/hardware stack, Lokhmotov et al. [176] tunes the hyperparameters of MobileNets for test accuracy and prediction time on an image classification task.

5.3 Fairness

When algorithmic decisions made by ML models impact the lives of humans, it is important to avoid introducing bias that adversely affects sub-groups of the population [19, 112]. As an illustrating example, consider a bank's decision-making process for loan applications. Ethical and legal requirements dictate that a model should not discriminate applicants because of *protected attributes*, such as *gender* or *race*. Simultaneously, the bank wishes to prioritize predicting loan defaults as accurately as possible in order to maximize profits. Jointly optimizing those objectives can provide additional clarity to decision-makers by informing them of possible trade-offs while yielding models that satisfy fairness metrics, allowing them to make an informed decision regarding which models to deploy and enables the use of *fair* models.

For a survey on different fairness perspectives and metrics, we refer to two excellent surveys by Mehrabi et al. [189] and Pessach and Shmueli [212], as well as a benchmark by [94] for a more comprehensive take on fair ML than is possible in the scope of this paper. The general goal of fair ML is to *detect* and potentially *mitigate* biases arising due to the use of ML models. For this reason, several *fairness metrics* as well as *debiasing methods* have been proposed. The choice of the correct metric depends on the context a decision is made in [272], see e.g., [226] who propose a *Fairness Tree* that maps applicable metrics to different problem settings. While a variety of *causal* and *individual* fairness notions exist, so-called (statistical) group fairness metrics are widely used in practice, since they are easy to implement and do not require access to the underlying (causal) data generating mechanism. Given a protected attribute A (e.g., race), an outcome X, and a binary predictor Y_b, several criteria can be derived [19], and we provide three widely used examples:

- **Equalized Odds (Independence)**:

  $$Pr(Y_b = 1|A = 1, Y = y) = Pr(Y_b = 1|A = 0, Y = y), y \in \{0, 1\}$$

  Fulfilling this essentially requires equal true positive and false positive rates between subpopulations. More generally, independence on the conditional Y is required. *Equalized odds* for a binary predictor is the most relevant example of such an independence measure [107].
- **Equality of Opportunity (Sufficiency)**:

  $$Pr(Y_b = 1|A = 1, Y = 1) = Pr(Y_b = 1|A = 0, Y = 1)$$

  This criterion is a relaxation of equalized odds, as only independence on the event $Y = 1$, which denotes the advantageous outcome (e.g. getting approved for a loan) here, is required. It therefore constitutes *equality of opportunity* [107].

- **Calibration** is another desirable criterion for classifiers, especially in the context of fairness, where calibrated probabilities in all groups may be required. For a classifier $h(x)$ that yields predicted probability, *calibration* requires that:

$$\underset{a\in\{0,1\}}{\forall}\ \underset{p\in[0;1]}{\forall} Pr(Y_b = 1|A = a, h(x) = p) = p.$$

Many fairness metrics metrics operate on a Fairness Tensor [145] and essentially measure differences in performance metrics such as FPR or FOR between protected groups.

Applications and exemplary use case. Existing approaches towards improving a ML-algorithm's fairness mainly focus on decreasing discrepancies in classifier performance between sub-groups or individuals (e.g., in [19, 156]). This is achieved e.g., by pre-processing the data (e.g., [141]), imposing fairness constraints during model fits [156] or by post-processing model predictions [5, 107]. Those methods in turn often come with hyperparameters that can be further tuned in order to emphasize fairness during training [230, 231]. Several approaches towards optimizing (model-) hyperparameters for fairness in a multi-objective fashion have been proposed. While Pfisterer et al. [213] propose multi-objective BO to jointly optimize fairness criteria as well as prediction accuracy, [211] propose a *constrained* BO approach, where the fairness metric is constrained to a small deviation from optimal fairness while predictive accuracy is optimized. Pelegrina et al. [210] use a MOEA to optimize simultaneously for fairness metrics and performance in an attempt to find a fair PCA. Yet another approach is introduced in Martinez et al. [188], where each sensitive group risk is a separate objective, also leading to a MOHPO problem for choosing a classifier. It is interesting to note that fairness can be heavily influenced not only by parameters of the debiasing method, but also by the choice of ML algorithm and hyperparameters [213]. One popular dataset studied in the context of Fairness is the COMPAS (Correctional Offender Management Profiling for Alternative Sanctions) data [8]. The goal of COMPAS is to predict the risk that a criminal defendant will re-offend. The goal is to obtain a prediction that is accurate but simultaneously not biased towards individuals of any race. In the following example, the latter quantity is measured via $\tau_{FPR} = \frac{FPR_{S_0}}{FPR_{S_1}}$ (optimal for $\tau_{FPR} = 1$, where FPR_S is the false positive rate on group S with S_0 and S_1 is the advantaged and disadvantaged group, respectively). The effect of applying several debiasing strategies – interpolating between no debiasing and full debiasing – is shown in Figure 9. A random forest (*RF*) model is trained with different debiasing techniques: Reweighing [141], Equalized Odds [107], and Natural Language Processing [5]. As can be observed, these different strategies and debiasing strengths lead to different trade-offs, thus resulting in a MOO problem. It is important to note that existing approaches (including the ones we cover here) often propagate a solely technological solution to a *socio-technical* problem [112]. While MOO for fairness can solve the *technological* aspects of this problem, it cannot be forgotten that the social aspects must also be addressed [232]. Furthermore, practitioners must not only take into account the model itself, but also the data used to train the algorithm [202, 248], the process behind the collection and labeling of such data, and eventual feedback loops arising from use of potentially biased models.

5.4 Interpretability

In order to make an ML model's decisions more transparent, different methods that aim at providing human-understandable explanations have been proposed (c.f. [194]). Requirements for interpretability generally range from models derived from fully interpretable model classes to interpretability via post-hoc interpretation techniques. For example, the former is required to satisfy regulatory constraints in the banking sector [42, 50] and can be thought of as a constraint on the search space when selecting a model. The latter is often used to understand functional relationships between features and target variables in an ML context (i.e., understanding the model) or to understand

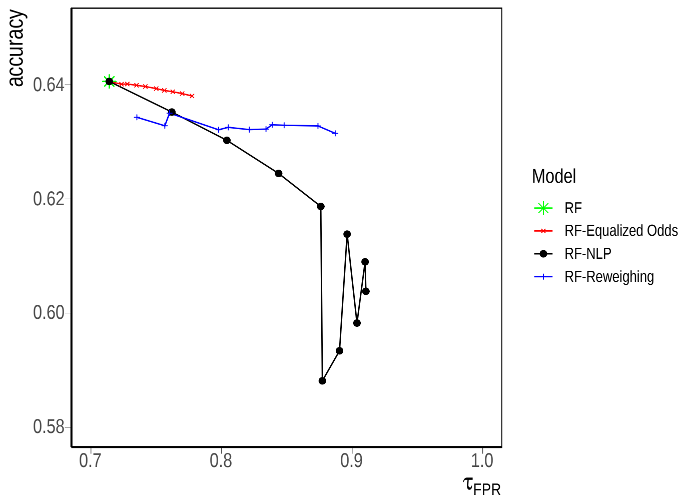


Fig. 9. Effect of 3 different debiasing strategies on the fairness-accuracy trade-off on the COMPAS dataset measured on test data. Figure obtained from [5]

single decisions made by a trained algorithm [194]. In addition, interpretability techniques can be helpful to debug errors made by models (e.g., errors stemming from mislabeled data or spuriously drawn correlations [155]). While many explanations can be either directly derived from the model class (e.g., decision trees and generalized linear models), interpretability techniques either focus on a single model class [204, 233] or are *model-agnostic* [9, 183]. The latter is especially desirable, as this allows the user to explain arbitrary models resulting from tuning over various model classes. On the other hand, interpretability methods can produce misleading results if a model is too complex or the explainability technique is unreliable due to them e.g., using additional features [147, 194]. Quantifying interpretability, i.e., *determining how complex the predictive decisions of a given model are*, could be a first step towards obtaining models that provide reliable explanations and interpretable models. Quantifying interpretability of a model is not straightforward, as terms like interpretability, explainability, and complexity are highly subjective expressions [171, 194, 195]. First approaches to metrics that can be used as a proxy for interpretability on tabular data have been proposed, as shown e.g., in Molnar et al. [195]. While explainability for non-tabular application domains may be equally interesting, no methods that allow quantifying the interpretability of e.g., computer vision models have been proposed to date. A popular proxy for model complexity (and therefore interpretability) is *sparseness*, i.e., the number of features. We explore this concept further in the context of MOHPO in Section 5.6. Molnar et al. [195] propose 2 additional metrics aside from sparseness to quantify interpretability on tabular data, which we will briefly summarize here[12]:

- **Complexity of main effects** Molnar et al. [195] propose to determine the average shape complexity of Accumulated Local Effects (ALE) [9] main effects by the number of parameters needed to approximate the curve with linear segments. The Main Effect Complexity (MEC) is defined as:

$$MEC = \frac{1}{\sum_{j=1}^{p} V_j} \sum_{j=1}^{p} V_j \cdot MEC_j, \tag{16}$$

[12]A detailed derivation and explanation can be found in Molnar [194], Molnar et al. [195]

where $V_j = \frac{1}{n}\sum_{i=1}^{n}(f_{j,ALE}(x^{(i)}))^2$ and $MEC_j = K + \sum_{k=1}^{K}\mathbb{I}_{\beta_{1,k}>0} - 1$. K is the number of linear segments needed for a good-enough approximation, p is the number of features, β_1, k the slope of the $k - th$ linear segment, and n the number of samples.

- **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 IAS can be measured as the fraction of variance that cannot be explained by main effects:

$$IAS = \frac{\mathbb{E}(L(f, f_{ALE1st}))}{\mathbb{E}(L(f, f_0))} \geq 0, \tag{17}$$

where f is the prediction function, f_{ALE1st} the sum of first order ALE effects, and f_0 the mean of predictions.

Applications and exemplary use case. Molnar et al. [195] aim to find an accurate and interpretable model to predict the quality of white wines (on a scale from 0 to 10) [59]. In order to accomplish this, they define a large search space of several models (SVM, gradient boosted trees, and random forest, among others) with a number of tunable hyperparameters and optimize for four objectives: cross-validated mean absolute error, number of features used, main effect complexity, and interaction strength. They conduct the optimization with ParEGO (see Section 4.3.2) over 500 iterations to find good trade-offs between these objectives. Carmichael et al. [48] perform MOHPO for deep learning architectures to find optimal trade-offs between accuracy and introspectability for image classification tasks on ImageNet-16-120, CIFAR-10 and MNIST.

5.5 Robustness

In many use cases, it is highly desirable to identify and deploy robust ML models, i.e., models that are able to maintain good performance despite variation in the input data [66, 96]. Robustness in the context of ML is only loosely defined. We broadly distinguish between robustness of the *training procedure* and robustness of the *fitted model*, which have both been described in literature. We focus on the latter, i.e., the susceptibility of a *trained model* to shifts of the data in the prediction step. While most research into robustness focuses on images, we aim to look at a general case. We mostly consider a *classification setting* in the following sections, but we also aim to provide information as to how this differs for *regression* where appropriate.

5.5.1 Robustness metrics and approaches. While generally seen as important and relevant, there are no tried and proven metrics to assess the robustness of ML models, nor a proper taxonomy. The taxonomy in [247] provides an overview of possible changes to the input data. While centered around image data, many ideas can be carried over to other types of data. They distinguish between *natural* and *synthetic* changes to the input data. *Natural* changes equate to changes to the dataset relying solely on unmodified data points, whereas *synthetic* changes allow for modification (e.g., perturbations) of data points. We differentiate between three types of changes:

Distribution shift. Our notion of a distribution shift refers to changes of either the marginal distribution of the target or the distribution of the features (conditional on the target) on a macro level. This concept is often associated with domain adaption (which is becoming increasingly popular as a research area, see Zhang et al. [265] for an introduction). Several typical real-world examples would be the increase in temperature of a manufacturing environment that leads to a change in sensor readings, or an image classification task previously trained and tested on only well-lit pictures suddenly exposed to darker pictures, or different weather conditions. Typical metrics used to assess robustness in the context of distribution shifts are *effective robustness*

$\rho(f) = acc_2(f) - \beta(acc_1(f))$, where acc_1, acc_2 are the accuracies pre- and post- domain shift, and β is the chosen baseline accuracy on the shifted test set. Duchi and Namkoong [74] show that optimizing a model for robustness in the context of distribution shift equates to optimizing it for performance on the tails. They then propose to reformulate the optimization problem from optimizing for average performance to instead using a metric that upweights the regions with the highest loss. They dub this reformulated optimization problem the *distributionally robust problem.* Indeed, the notion of *distributionally robust optimization* has existed for some time and has been applied to both data-driven and ML problems [70, 219].

Adversarial examples. Adversarial examples have recently generated substantial interest from the visual deep learning community [6, 98, 249], because a model that is very susceptible to adversarial attacks (the use of adversarial examples) is not as trustworthy. Prominent examples from image data present pictures with small perturbations – often undetectable to a human observer – that "trick" the ML model into false classification, whereas the original image was classified correctly. Adversarial examples are well-known in image data [259] but have been shown in other types of data, such as text [124], sequence [53], or tabular data [17]. Oftentimes, adversarial examples generated through perturbation are in focus, but different adversarial attacks generally lead to different measures [208]. A popular metric to assess robustness in the context of adversarial examples is adversarial accuracy [185, 187, 250]. Adversarial accuracy measures the percentage of samples that are (still) correctly classified after the adversarial attack [250]. In practice the user would conduct the adversarial attack and then calculate adversarial accuracy from the new predictions. For adversarial perturbations in an ε-ball around each point, a typical adversarial attack in classification, it can be defined as:

$$\mathbb{E}[\mathbb{1}(f(x^*) = c_x)], \ \ where\ x^* = \underset{d(x',x)\leq\epsilon}{\arg\max}\, L(x', c_x), \tag{18}$$

where c_x is the respective class label. Two metrics for adversarial robustness in computer vision were originally defined in Bastani et al. [20] and have also been employed by Buzhinsky et al. [43]: *adversarial frequency* and *adversarial severity.* Adversarial frequency ϕ is measured as the accuracy on a worst-case input in an ℓ_p ε-ball around each point x_*:

$$\phi(f, \varepsilon) := P(\rho(f, x_*) \leq \varepsilon),$$

where $\rho(f, x_*)$ is the minimum distance $\hat{\varepsilon}$ for some well-defined metric d, so that $\exists\ x,\ d(x, x_*) \leq \hat{\varepsilon} : f(x) \neq f(x_*)$ with f as a classifier. Adversarial severity μ is defined as the expected minimum distance to an adversarial example from the input for some ε [20]:

$$\mu(f, \varepsilon) := E[\rho(f, x_*) \mid \rho(f, x_*) \leq \varepsilon],$$

with ρ, f, x_* as before. While Bastani et al. [20] deem *adversarial frequency* to be generally the more important metric of the two, significant work centers around the minimum distance to creating an adversarial example, especially for neural networks [209]. In Rauber et al. [221], this subsequently prompted the unification of various adversarial attacks and measurement of robustness through minimum required perturbation.

Perturbations. Perturbations are oftentimes strongly linked to the construction of adversarial examples in deep learning [159]. This can be seen as a logical connection, as one would often look for an adversarial example within some ϵ-ball (see above). Laugros et al. [159] argue and show that robustness in the context of common perturbations and adversarial robustness are independent attributes of a model. Their work focuses on image data and includes some transformations – like a change in brightness – that we would classify as a domain shift. One of the most common ways to perturb data is the addition of Gaussian noise, which is also included in their work. A simple measure for robustness regarding such a perturbation is presented in Pfisterer et al. [213]. Adding random Gaussian noise $N(0,\epsilon)$ [13] to input data X is a typical way to introduce perturbations. One can then compare the loss derived from a relevant measure L on the changed input data to the loss on unperturbed data:

$$|L(X,Y) - L(X + N(0,\epsilon), Y)|.$$

Of course, the same procedure can be applied to other types of perturbations. In Gilmer et al. [97], the authors argue and provide evidence that susceptibility to adversarial examples and perturbations – at least for image data – are actually two symptoms of the same underlying problem, and thus, optimizing for robustness against adversarial attacks and more general corruption through perturbations should go hand-in-hand. In general, perturbations and adversarial examples are closely related, and – as seen above – substantial research in the computer vision community focuses on adversarial perturbations [6, 43, 97]. Niu et al. [199] give a comprehensive overview on perturbations to data in machine translation (mainly synthetic misspellings and letter case changing) and define a variety of suitable measures for model robustness in the context of those perturbations.

5.5.2 A note on robustness and uncertainty quantification. Uncertainty quantification, especially in the context for deep learning, has become a heavily researched topic and is often mentioned in conjunction with robustness of ML models [157]. To the best of our knowledge, optimizing an ML model's configuration for proper uncertainty quantification or minimal uncertainty is not an established method – and integrating a respective metric into MOHPO is even less so. We will therefore restrict this brief section to exploring the connection between uncertainty quantification and robustness as well as other relevant objectives. The total uncertainty of a model can be divided into *aleatoric uncertainty* and *epistemic uncertainty*. While the former is inherent to observations, the latter accounts for uncertainty in the model parameters and can be explained away with additional data [144]. Models that are better calibrated with respect to uncertainty tend to suffer less from adversarial examples [157, 238]. Along the same lines, robustness regarding domain shift and predictive performance on out-of-distribution samples are closely linked to (and even used as a measure of) predictive uncertainty [157]. As perturbations are often used to model noise in the data, a connection between this type of robustness and aleatoric uncertainty is easily conjectured. Indeed, Kendall and Gal [144] show that using the quantification of aleatoric uncertainty has a beneficial effect on the robustness in the context of noise, i.e., perturbations. Finally, uncertainty quantification can be relevant to the interpretability of an ML model. As argued in Kendall and Gal [144], a proper quantification of the model's uncertainty adds an extra layer of explainability and might aid in the prevention of critical mistakes, thus increasing the trustworthiness of the model.

5.5.3 Application and exemplary use case. The main goal of Guo et al. [102] is to identify network architectures that are robust with respect to adversarial attacks. To this end, the authors employ one-shot NAS [23]. Essentially, this entails defining a search space with several operations: An architecture is specified through hyperparameters $\alpha = \{\alpha_k^{(i,j)} \mid \alpha_k^{(i,j)} \in \{0,1\},\ i,j = 1,\dots,N,\ k = 1,\dots,n\}$ that represents the inclusion ($\alpha_k^{(i,j)} = 1$) or exclusion ($\alpha_k^{(i,j)} = 0$) of a transformation from a pool of n transformations[14] between nodes i, j of a graph representing a computational cell in the network. After initial training of a *supernet*, i.e., all hyperparameters set to 1, architectures are drawn through random search, fine-tuned and evaluated. While the optimization is not strictly multi-objective, this is not as important when conducting what is essentially a random search, because subsequent trials are not influenced by prior ones. The authors then examine different architectures

[13] ϵ is generally $0.001 - 0.01$ times the range of the numerical feature

[14] 3×3-separable convolution, identity, and zero in this case.

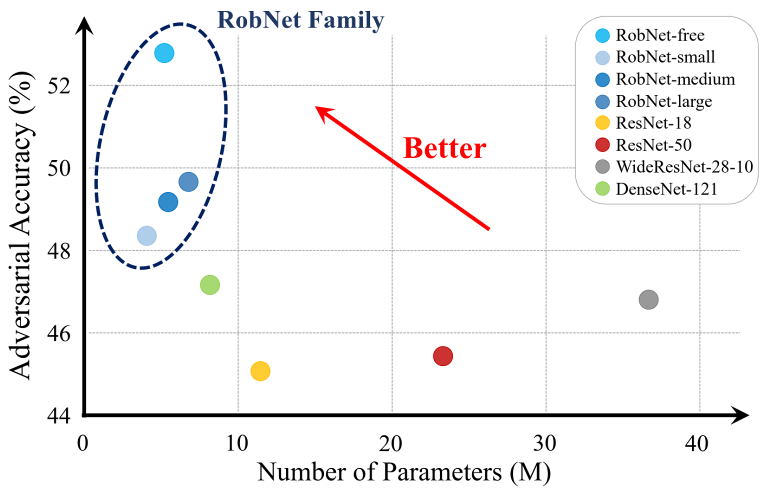


Fig. 10. Overview and classification of several architectures examined by Guo et al. [102] with respect to objectives *number of parameters* and *adversarial accuracy*.

and compare the desirable qualities of robustness in the context of adversarial examples and model size to identify suitable models. This is illustrated for a few select architectures in Figure 10.

5.6 Sparseness via Feature Selection

In ML, we often face high-dimensional tasks with a large number of features. However, frequently only a small fraction of all features is informative. *Sparse* models, i.e., models which use relatively few features, are often desirable. If too many features are considered by a model, ...

- ...the relationship between features and model prediction may be hard to understand and interpret [104] (see Section 5.4),
- ...the cost for model fitting or inference may increase, either in terms of storage, computation, or in terms of actual costs for measuring or acquiring the data [193],
- ...the predictive performance of a model might even suffer because of the the *curse of dimensionality* [22].

The severity of the effects depends on the machine learning algorithm used; some algorithms scale worse than others with regards to the number of features, and some (e.g., k-nearest neighbors algorithm) suffer more from the curse of dimensionality than others. Because of these potential drawbacks, it may often be desirable to perform *feature selection* before or during training. Feature selection is inherently an MOO problem, as model performance and sparsity tend to be conflicting objectives; a lower number of features often means a lower performance due to reduced information. However, for a certain model and configuration there will be a specific desirable quantity of features, that is probably not the maximum number of features [154]: Including less features will likely decrease the performance due to lack of information, whereas including too many features will create an abundance of information and the configuration suffers increasingly from the curse of dimensionality. In the following, we will provide an overview of common feature selection techniques with an emphasis on their connection and possible combination with MOHPO. Feature selection approaches are generally categorized into three sections in accordance to how the evaluate feature configurations [7]:

Embedded methods. methods perform feature selection as part of the model fitting process. For example, empirical risk minimization with L0 or L1 regularization shrinks irrelevant coefficients towards zero, and can therefore automatically produce sparse models during training. For example, Louizos et al. [179] train sparse neural networks via L0 penalization. As another example, trees and tree-based methods can inherently produce sparse models by limiting the number of used features through the maximal tree depth. As feature selection is performed as part of model fitting, no separate search technique must be used. The drawback of embedded methods is that they are specific to the model in use.

Filter methods. approaches use proxies to rank feature subsets by their likely explanatory power independent of the learning algorithm. Single measures – for instance, information theoretic measures – correlation measures, distance measures, or consistency measures [61] are calculated and used for evaluation of different feature subsets. Filtering is usually applied before model training.

Wrapper methods. search over the space of selected features to optimize model performance [154]. Because they take the learner algorithm performance into account directly, they often yield better performance than filter methods. However, they are computationally more expensive because model performance evaluations are generally noisy and relatively expensive. Additionally, exhaustively trying all possible combinations is usually computationally infeasible. Because wrapper methods can use general optimization algorithms, they are the most amenable to extensions to MOO [7]. Hybrid approaches can also be utilized, as in Bommert et al. [33] and Binder et al. [29], where hybrid filter-wrapper approaches are proposed to limit the complexity of the search spaces. This is desirable as BO surrogate models for feature selection must be modeled on a potentially high-dimensional and combinatorial search space (see Eq. (19)).

5.6.1 Sparseness Metrics. Aside from the simplest form, namely minimizing the number of used features, it may be of interest to formulate other objectives in relation to sparsity by e.g., weighting the features:

- **(Weighted) Number of Features** included in the model

$$\sum_{j=1}^{p} w_j s_j.$$

  If features have different costs, e.g., because they have different acquisition costs if a model is intended to be applied in real life, different weights can be introduced for this purpose.
- **Stability of Feature Selection**: In some applications the main goal of the analysis is the identification of important features. In bioinformatics, for example, performance plays a subordinate role in the analysis of omics data. The aim is to identify important genes for later examination in the laboratory. Besides predictive performance and the number of selected features, a third objective quantifying the stability of feature selection is introduced e.g., by comparing sets of selected features resulting from different resampling iterations.

We argue that it is reasonable and efficient to consider feature selection and hyperparameter optimization in a joint step and will briefly mention some approaches that do so: Binder et al. [29] examine an approach including feature selection and hyperparameter optimization of ML models and benchmark EAs as well as BO methods. Bouraoui et al. [35] propose a similar approach, unifying model and feature selection, restricted to SVMs. Several MOEAs are used for conjoint MOHPO and feature selection in Sopov and Ivanov [240] with application to emotion recognition. Liuliakov and Hammer [175] proposed a combined feature selection HPO AutoML tool based on TPOT.

5.6.2 Applications and exemplary use case. In general, the task of feature selection implies search over a binary space $\{0,1\}^p$ with $s_i = \mathbb{I}[\text{feature } i \text{ is included}]$. To simultaneously perform hyperparameter optimization, we formulate the joint search space of feature and hyperparameter configurations

$$\{0,1\}^p \times \tilde{\Lambda}. \tag{19}$$

The dimensionality of this space grows exponentially with the number of available features, which makes the optimization problem particularly challenging. Additionally, there may be complex interactions between features and hyperparameter configurations. Expensive evaluation like wrapper evaluation techniques raise the need for efficient search methods. Evolutionary algorithms (see Section 4.2) are widely used for feature selection due to their ability to handle complex search spaces, [1, 261], but recent works have presented BO methods (see Section 4.3) as a more efficient alternative [29, 33]. Aside from exploring one exemplary use case, we will highlight a few select examples of feature selection in the context of MOHPO. A comprehensive review with additional examples can be found in Al-Tashi et al. [7] and specifically for Evolutionary Computation methods in Xue et al. [261][15]. We highlight two works in the scope of multi-objective feature selection. Bommert et al. [32] state feature selection as a MOHPO problem with three objectives: predictive performance, number of features selected, and stability of feature selection. Furthermore, they provide a comprehensive comparison of stability measures. They tune hyperparameters of the underlying ML pipelines (feature filter plus classification learner) that are relevant to the sparseness objectives via random search to identify desirable trade-offs. All classification pipelines have been tuned w.r.t. the aforementioned three criteria. In the next step, all configurations that are not within a 5% tolerance of the best predictive performance identified so far are discarded. The remaining configurations are shown in Figure 11 for an exemplary dataset. The figure demonstrates one takeaway message of the benchmark: In comparison with the single-criterion tuning approach (marked by the triangle in Figure 11), the multi-objective approach additionally reveals hyperparameter settings that yield models with comparable or even better performance. At the same time, these additional identified model configurations require fewer features, and the feature selection is more stable. Binder et al. [29] recently showed that model-based optimization - even more than evolutionary approaches - offer great performance enhancement compared to random search, which was used in Bommert et al. [32].

6 DISCUSSION AND OPEN CHALLENGES

This paper has presented an overview of the concepts, methods and applications of MOHPO - as well as related concepts - and provide a guide to the ML practitioner delving into this particular topic. It is evident there is merit to formulating ML problems in a multi-objective manner, as many application examples support. Single-objective ML tasks, tuned to a pure prediction performance metric, are no longer in keeping with the state-of-the-art for many ML applications, as models have to meet certain standards with respect to secondary goals. While this presents new challenges to the ML expert with regard to optimization and algorithm selection, proper methods can provide the user with a suite of Pareto optimal trade-offs to choose a suitable model. As the topic of MOHPO and multi-objective pipeline creation and model selection is not fully established, the available software (not for MOO, but specifically MOHPO) is limited, but good implementations exist for several standard methods. Finally, we want to emphasize the lack of proper and extensive benchmarks for the field of MOHPO, which could shed some further light on strengths and weaknesses of different methods on a variety

[15]This work mainly includes works that reduce feature selection to a single-objective problem, but also several multi-objective approaches.

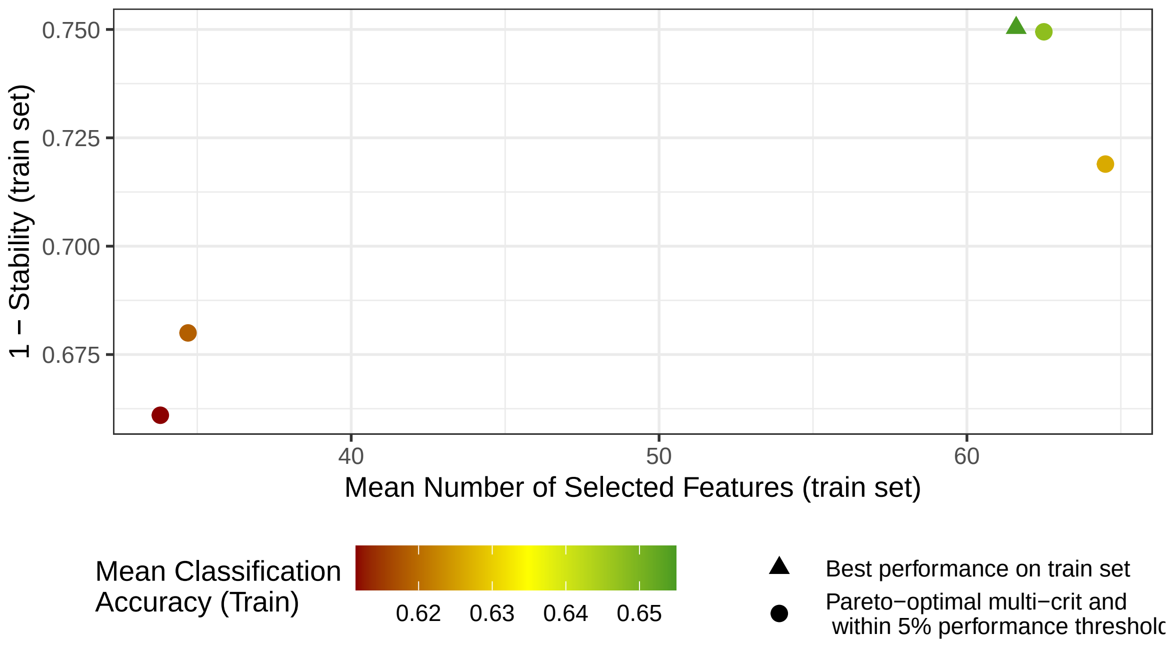


Fig. 11. Excerpt from results in Bommert et al. [32]: Different combinations of feature filters and classification learners tuned on the eating dataset (OpenML-ID 1233). Shown are points from the 3D Pareto front (projected into 2D space of stability measure vs. mean number of selected features) that are within the 5% threshold of the best predictive performance identified so far on the training set. Points that appear to be dominated in this 2D representation are in fact non-dominated due to higher accuracy. All three visualized criteria are evaluated on independent test sets.

of MOHPO tasks. The discrepancy in widespread vs. sparse use of MOHPO depending on the application at hand may also appear striking. ROC analysis and multi-objective feature selection are established and well researched areas; the body of literature for MOHPO including efficiency objectives has grown rapidly in the past few years with the ascent of deep learning and HW-NAS. With the recent trends to integrate FAT-ML related standards into the ML process, MOHPO with applications to interpretability and fairness is currently becoming increasingly relevant, but few works have been published that deal with these objectives. Integrating user preferences in a meaningful way either *a priori* or during the optimization process (see Section 4.5.1) remains a challenge that could help efficiency and transparency of MOHPO. It should be noted that MOHPO - compared to single-objective HPO - already improves in terms of transparency, simply by not having to reduce to a single metric and the result being a collection of trade-offs and not only a single HP configuration. Another challenge is MOHPO beyond supervised learning: We have focused in this work on supervised learning, but unsupervised methods like anomaly detection or clustering also depend heavily on HPs. As in single-objective HPO [30], the difficulty of performance evaluation and lack of standardized metrics complicate the application of the presented methods. Aside from "typical" performance measures, other objectives like e.g., efficiency can still be concretely evaluated and may be included in (MO)HPO of unsupervised methods.

ACKNOWLEDGMENTS

This work was supported 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) as well as the German Federal Ministry of Education and Research (BMBF) under Grant No. 01IS18036A.

REFERENCES

[1] Nadia Abd-Alsabour. 2014. A review on evolutionary feature selection. In *2014 European Modelling Symposium*. IEEE, 20–26. https://doi.org/10.1109/EMS.2014.28

[2] Majid Abdolshah, Alistair Shilton, Santu Rana, Sunil Gupta, and Svetha Venkatesh. 2019. Multi-objective Bayesian optimisation with preferences over objectives. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (Eds.). 12214–12224. https://dl.acm.org/doi/10.5555/3454287.3455383

[3] Ajith Abraham and Lakhmi Jain. 2006. *Evolutionary Multiobjective Optimization: Theoretical Advances and Applications*. Springer London. https://books.google.de/books?id=KHOQu7R_POoC

[4] Virginia Aglietti, Xiaoyu Lu, Andrei Paleyes, and Javier González. 2020. Causal Bayesian Optimization. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 3155–3164.

[5] Ashrya Agrawal, Florian Pfisterer, Bernd Bischl, Jiahao Chen, Srijan Sood, Sameena Shah, Francois Buet-Golfouse, Bilal A Mateen, and Sebastian Vollmer. 2020. Debiasing classifiers: is reality at variance with expectation? arXiv:2011.02407

[6] Naveed Akhtar and Ajmal Mian. 2018. Threat of Adversarial Attacks on Deep Learning in Computer Vision: A Survey. arXiv:1801.00553

[7] Qasem Al-Tashi, Said Jadid Abdulkadir, Helmi Md Rais, Seyedali Mirjalili, and Hitham Alhussian. 2020. Approaches to Multi-Objective Feature Selection: A Systematic Literature Review. *IEEE Access* 8 (2020), 125076–125096. https://doi.org/10.1109/ACCESS.2020.3007291

[8] Julia Angwin, Jeff Larson, Surya Mattu, and Lauren Kichner. 2016. Machine Bias. https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing

[9] Daniel W. Apley. 2016. Visualizing the effects of predictor variables in black box supervised learning models. arXiv:1612.08468

[10] Jasbir S. Arora. 2004. Multiobjective Optimum Design Concepts and Methods. In *Introduction to Optimum Design (Second Edition)* (second edition ed.), Jasbir S. Arora (Ed.). Academic Press, San Diego, 543 – 563.

[11] Raul Astudillo and Peter Frazier. 2017. Multi-attribute Bayesian optimization under utility uncertainty. In *NIPS workshop on Bayesian Optimization*, Vol. 172.

[12] Charles Audet and Warren Hare. 2017. *Derivative-Free and Blackbox Optimization*. Springer International Publishing.

[13] Nebojsa Bacanin, Timea Bezdan, Eva Tuba, Ivana Strumberger, and Milan Tuba. 2020. Optimizing Convolutional Neural Network Hyperparameters by Enhanced Swarm Intelligence Metaheuristics. *Algorithms* 13 (03 2020), 67. https://doi.org/10.3390/a13030067

[14] Bowen Baker, Otkrist Gupta, Ramesh Raskar, and Nikhil Naik. 2017. Accelerating neural architecture search using performance prediction. arXiv:1705.10823

[15] Maximilian Balandat, Brian Karrer, Daniel R. Jiang, Samuel Daulton, Benjamin Letham, Andrew Gordon Wilson, and Eytan Bakshy. 2020. BoTorch: A Framework for Efficient Monte-Carlo Bayesian Optimization. arXiv:1910.06403 [cs.LG]

[16] Maria Baldeon-Calisto and Susana K. Lai-Yuen. 2020. AdaResU-Net: Multiobjective adaptive convolutional neural network for medical image segmentation. *Neurocomputing* 392 (2020), 325–340.

[17] Vincent Ballet, Xavier Renard, Jonathan Aigrain, Thibault Laugel, Pascal Frossard, and Marcin Detyniecki. 2019. Imperceptible Adversarial Attacks on Tabular Data. arXiv:1911.03274

[18] Mohua Banerjee, Sushmita Mitra, and Ashish Anand. 2006. Feature Selection Using Rough Sets. In *Multi-Objective Machine Learning*, Yaochu Jin (Ed.). Studies in Computational Intelligence, Vol. 16. Springer, 3–20.

[19] Solon Barocas, Moritz Hardt, and Arvind Narayanan. 2018. *Fairness and Machine Learning*. fairmlbook.org. http://www.fairmlbook.org.

[20] Osbert Bastani, Yani Ioannou, Leonidas Lampropoulos, Dimitrios Vytiniotis, Aditya V. Nori, and Antonio Criminisi. 2016. Measuring Neural Net Robustness with Constraints. arXiv:1605.07262

[21] Syrine Belakaria, Aryan Deshwal, and Janardhan Rao Doppa. 2019. Max-value Entropy Search for Multi-Objective Bayesian Optimization. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (Eds.). 7823–7833. https://proceedings.neurips.cc/paper/2019/hash/82edc5c9e21035674d481640448049f3-Abstract.html

[22] Richard E. Bellman. 2015. *Adaptive Control Processes*. Princeton University Press, Princeton. 94 pages. https://doi.org/10.1515/9781400874668

[23] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. 2018. Understanding and simplifying one-shot architecture search. In *International Conference on Machine Learning*. PMLR, 550–559.

[24] Hadjer Benmeziane, Kaoutar El Maghraoui, Hamza Ouarnoughi, Smaïl Niar, Martin Wistuba, and Naigang Wang. 2021. A Comprehensive Survey on Hardware-Aware Neural Architecture Search. arXiv:2101.09336 https://arxiv.org/abs/2101.09336

[25] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for Hyper-Parameter Optimization. In *Advances in Neural Information Processing Systems 24*, J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 2546–2554.

[26] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of machine learning research* 13, 2 (2012).

[27] Simon Bernard, Clément Chatelain, Sébastien Adam, and Robert Sabourin. 2016. The multiclass roc front method for cost-sensitive classification. *Pattern Recognition* 52 (2016), 46–60.

[28] Jinbo Bi and Kristin P. Bennett. 2003. Regression error characteristic curves. In *Proceedings of the 20th international conference on machine learning (ICML-03)*. 43–50.

[29] Martin Binder, Julia Moosbauer, Janek Thomas, and Bernd Bischl. 2020. Multi-Objective Hyperparameter Tuning and Feature Selection Using Filter Ensembles. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference* (Cancún, Mexico) *(GECCO '20)*. Association for Computing Machinery, New York, NY, USA, 471–479. https://doi.org/10.1145/3377930.3389815

[30] 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. 2021. Hyperparameter Optimization: Foundations, Algorithms, Best Practices and Open Challenges. arXiv:2107.05847 [stat.ML]

[31] Bernd Bischl, Olaf Mersmann, Heike Trautmann, and Claus Weihs. 2012. Resampling methods for meta-model validation with recommendations for evolutionary computation. *Evolutionary computation* 20, 2 (2012), 249–275.

[32] Andrea Bommert, Jörg Rahnenführer, and Michel Lang. 2017. A Multicriteria Approach to Find Predictive and Sparse Models with Stable Feature Selection for High-Dimensional Data. *Computational and Mathematical Methods in Medicine* 2017 (2017), 1–18. https://doi.org/10.1155/2017/7907163

[33] Andrea Bommert, Xudong Sun, Bernd Bischl, Jörg Rahnenführer, and Michel Lang. 2020. Benchmark for filter methods for feature selection in high-dimensional classification data. *Computational Statistics & Data Analysis* 143 (March 2020), 106839. https://doi.org/10.1016/j.csda.2019.106839

[34] Jakob Bossek, Carola Doerr, and Pascal Kerschke. 2020. Initial Design Strategies and Their Effects on Sequential Model-Based Optimization: An Exploratory Case Study Based on BBOB. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference (GECCO '20)*. Association for Computing Machinery, New York, NY, USA, 778–786. https://doi.org/10.1145/3377930.3390155

[35] Amal Bouraoui, Salma Jamoussi, and Yassine BenAyed. 2018. A multi-objective genetic algorithm for simultaneous model and feature selection for support vector machines. *Artificial Intelligence Review* 50, 2 (2018), 261–281.

[36] Jürgen Branke. 2016. *MCDA and Multiobjective Evolutionary Algorithms*. Springer New York, New York, NY, 977–1008.

[37] Jürgen Branke and Kalyanmoy Deb. 2005. Integrating user preferences into evolutionary multi-objective optimization. In *Knowledge incorporation in evolutionary computation*. Springer, 461–477.

[38] Jürgen Branke, Kalyanmoy Deb, Henning Dierolf, and Matthias Osswald. 2004. Finding knees in multi-objective optimization. In *Parallel Problem Solving from Nature (LNCS, 3242)*. Springer, 722–731.

[39] Juergen Branke, Kalyan Deb, Kaisa Miettinen, and Roman Słowiński. 2008. *Multiobjective Optimization: Interactive and Evolutionary Approaches*. Springer. https://books.google.de/books?id=N-1hWMNUa2EC

[40] Jürgen Branke, Christian Schmidt, and Hartmut Schmeck. 2001. Efficient Fitness Estimation in Noisy Environments. In *Genetic and Evolutionary Computation Conference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 243–250.

[41] Carla E. Brodley, Umaa Rebbapragada, Kevin Small, and Byron Wallace. 2012. Challenges and opportunities in applied machine learning. *Ai Magazine* 33, 1 (2012), 11–24.

[42] Michael Bücker, Gero Szepannek, Alicja Gosiewska, and Przemyslaw Biecek. 2020. Transparency, Auditability and eXplainability of Machine Learning Models in Credit Scoring. arXiv:2009.13384

[43] Igor Buzhinsky, Arseny Nerinovsky, and Stavros Tripakis. 2020. Metrics and methods for robustness evaluation of neural networks with generative models. arXiv:2003.01993 [cs.LG]

[44] Han Cai, Ligeng Zhu, and Song Han. 2019. ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

[45] Maria Baldeon Calisto and Susana K. Lai-Yuen. 2020. AdaEn-Net: An ensemble of adaptive 2D–3D Fully Convolutional Networks for medical image segmentation. *Neural Networks* 126 (2020), 76–94.

[46] Maria G. Baldeon Calisto and Susana K. Lai-Yuen. 2021. EMONAS-Net: Efficient multiobjective neural architecture search using surrogate-assisted evolutionary algorithm for 3D medical image segmentation. *Artif. Intell. Medicine* 119 (2021), 102154. https://doi.org/10.1016/j.artmed.2021.102154

[47] Matt Carlyle, John W. Fowler, Esma Gel, and Bosun Kim. 2003. Quantitative Comparison of Approximate Solution Sets for Bi-criteria Optimization Problems. *Decision Sciences* 34 (02 2003). https://doi.org/10.1111/1540-5915.02254

[48] Zachariah Carmichael, Tim Moon, and Sam A. Jacobs. 2021. Learning Interpretable Models Through Multi-Objective Neural Architecture Search. arXiv:2112.08645

[49] Rich Caruana and Alexandru Niculescu-Mizil. 2004. Data mining in metric space: an empirical analysis of supervised learning performance criteria. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. 69–78.

[50] Diogo V. Carvalho, Eduardo M. Pereira, and Jaime S. Cardoso. 2019. Machine learning interpretability: A survey on methods and metrics. *Electronics* 8, 8 (2019), 832.

[51] Akshay Chandrashekaran and Ian Lane. 2016. Automated optimization of decoder hyper-parameters for online LVCSR. In *2016 IEEE Spoken Language Technology Workshop (SLT)*. IEEE, 454–460.

[52] Clément Chatelain, Sébastien Adam, Yves Lecourtier, Laurent Heutte, and Thierry Paquet. 2010. A multi-model selection framework for unknown and/or evolutive misclassification cost problems. *Pattern Recognition* 43, 3 (2010), 815–823.

[53] Minhao Cheng, Jinfeng Yi, Pin-Yu Chen, Huan Zhang, and Cho-Jui Hsieh. 2020. Seq2Sick: Evaluating the Robustness of Sequence-to-Sequence Models with Adversarial Examples. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*. AAAI Press, 3601–3608. https://aaai.org/ojs/index.php/AAAI/article/view/5767

[54] Ting-Wu Chin, Ari S. Morcos, and Diana Marculescu. 2021. Joslim: Joint Widths and Weights Optimization for Slimmable Neural Networks. In *Machine Learning and Knowledge Discovery in Databases. Research Track - European Conference, ECML PKDD 2021, Bilbao, Spain, September 13-17, 2021, Proceedings, Part III (Lecture Notes in Computer Science, Vol. 12977)*, Nuria Oliver, Fernando Pérez-Cruz, Stefan Kramer, Jesse Read, and José A. Lozano (Eds.). Springer, 119–134. https://doi.org/10.1007/978-3-030-86523-8_8

[55] Xiangxiang Chu, Bo Zhang, and Ruijun Xu. 2020. Multi-objective Reinforced Evolution in Mobile Neural Architecture Search. In *Computer Vision - ECCV 2020 Workshops - Glasgow, UK, August 23-28, 2020, Proceedings, Part IV (Lecture Notes in Computer Science, Vol. 12538)*, Adrien Bartoli and Andrea Fusiello (Eds.). Springer, 99–113. https://doi.org/10.1007/978-3-030-66823-5_6

[56] Carlos A. Coello Coello, Gary B. Lamont, and David A. van Veldhuizen. 2007. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Springer US. https://books.google.de/books?id=rXIuAMw3lGAC

[57] Carlos A. Coello Coello and Margarita R. Sierra. 2004. A Study of the Parallelization of a Coevolutionary Multi-objective Evolutionary Algorithm. In *MICAI 2004: Advances in Artificial Intelligence, Third Mexican International Conference on Artificial Intelligence, Mexico City, Mexico, April 26-30, 2004, Proceedings (Lecture Notes in Computer Science, Vol. 2972)*, Raúl Monroy, Gustavo Arroyo-Figueroa, Luis Enrique Sucar, and Juan Humberto Sossa Azuela (Eds.). Springer, 688–697.

[58] Corinna Cortes and Mehryar Mohri. 2003. AUC optimization vs. error rate minimization. *Advances in neural information processing systems* 16 (2003), 313–320.

[59] Paulo Cortez, António Cerdeira, Fernando Almeida, Telmo Matos, and José Reis. 2009. Modeling wine preferences by data mining from physicochemical properties. *Decis. Support Syst.* 47, 4 (2009), 547–553. https://doi.org/10.1016/j.dss.2009.05.016

[60] Piotr Czyzżak and Adrezej Jaszkiewicz. 1998. Pareto simulated annealing—a metaheuristic technique for multiple-objective combinatorial optimization. *Journal of Multi-Criteria Decision Analysis* 7, 1 (1998), 34–47.

[61] Manoranjan Dash and Huan Liu. 1997. Feature selection for classification. *Intelligent Data Analysis* 1, 3 (1997), 131–156.

[62] Samuel Daulton, Maximilian Balandat, and Eytan Bakshy. 2021. Parallel Bayesian Optimization of Multiple Noisy Objectives with Expected Hypervolume Improvement. In *Advances in Neural Information Processing Systems*.

[63] Jesse Davis and Mark Goadrich. 2006. The relationship between Precision-Recall and ROC curves. In *Proceedings of the 23rd international conference on Machine learning*. 233–240.

[64] Erik A. Daxberger, Anastasia Makarova, Matteo Turchetta, and Andreas Krause. 2020. Mixed-Variable Bayesian Optimization. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, Christian Bessiere (Ed.). ijcai.org, 2633–2639. https://doi.org/10.24963/ijcai.2020/365

[65] Kalyanmoy Deb. 2001. *Multi-Objective Optimization using Evolutionary Algorithms*. Wiley. https://books.google.de/books?id=OSTn4GSy2uQC

[66] Kalyan Deb and Himanshu Gupta. 2006. Introducing Robustness in Multi-Objective Optimization. *Evolutionary computation* 14 (2006), 463–94.

[67] Kalyanmoy Deb and Himanshu Jain. 2014. An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems With Box Constraints. *Evolutionary Computation, IEEE Transactions on* 18 (08 2014), 577–601. https://doi.org/10.1109/TEVC.2013.2281535

[68] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6, 2 (2002), 182–197.

[69] Dwyer S. Deighan, Scott E. Field, Collin D. Capano, and Gaurav Khanna. 2021. Genetic-algorithm-optimized neural networks for gravitational wave classification. *Neural Computing and Applications* 33, 20 (2021), 13859–13883.

[70] Erick Delage and Yinyu Ye. 2010. Distributionally Robust Optimization Under Moment Uncertainty with Application to Data-Driven Problems. *Operations Research* 58 (06 2010), 595–612. https://doi.org/10.1287/opre.1090.0741

[71] Janez Demšar. 2006. Statistical Comparisons of Classifiers over Multiple Data Sets. *Journal of Machine Learning Research* 7, 1 (2006), 1–30. http://jmlr.org/papers/v7/demsar06a.html

[72] Jin-Dong Dong, An-Chieh Cheng, Da-Cheng Juan, Wei Wei, and Min Sun. 2018. PPP-Net: Platform-aware Progressive Search for Pareto-optimal Neural Architectures. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=B1NT3TAIM

[73] Jin-Dong Dong, A. Cheng, Da-Cheng Juan, Wei Wei, and Min Sun. 2018. PPP-Net: Platform-aware Progressive Search for Pareto-optimal Neural Architectures. In *ICLR*.

[74] John Duchi and Hongseok Namkoong. 2020. Learning Models with Uniform Performance via Distributionally Robust Optimization. arXiv:1810.08750 [stat.ML]

[75] Darrin C. Edwards, Charles E. Metz, and Robert M. Nishikawa. 2005. The hypervolume under the ROC hypersurface of" near-guessing" and" near-perfect" observers in N-class classification tasks. *IEEE transactions on medical imaging* 24, 3 (2005), 293–299.

[76] Katharina Eggensperger, Philipp Müller, Neeratyoy Mallik, Matthias Feurer, René Sass, Aaron Klein, Noor H. Awad, Marius Lindauer, and Frank Hutter. 2021. HPOBench: A Collection of Reproducible Multi-Fidelity Benchmark Problems for HPO. *CoRR* abs/2109.06716 (2021).

[77] Matthias Ehrgott. 2005. *Multicriteria Optimization (2. ed.)*. Springer. https://doi.org/10.1007/3-540-27659-9

[78] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. 2019. Efficient Multi-Objective Neural Architecture Search via Lamarckian Evolution. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

[79] Michael T. M. Emmerich, Nicola Beume, and Boris Naujoks. 2005. An EMO algorithm using the hypervolume measure as selection criterion. In *International Conference on Evolutionary Multi-Criterion Optimization*. Springer, 62–76.

[80] Michael T. M. Emmerich and André H. Deutz. 2018. A tutorial on multiobjective optimization: fundamentals and evolutionary methods. *Natural computing* 17, 3 (2018), 585–609.

[81] Michael T. M. Emmerich, André H. Deutz, and Jan Willem Klinkenberg. 2011. Hypervolume-based expected improvement: Monotonicity properties and exact computation. In *2011 IEEE Congress of Evolutionary Computation (CEC)*. IEEE, 2147–2154.

[82] Michael T. M. Emmerich, Kyriakos C. Giannakoglou, and Boris Naujoks. 2006. Single-and multiobjective evolutionary optimization assisted by Gaussian random field metamodels. *IEEE Transactions on Evolutionary Computation* 10, 4 (2006), 421–439.

[83] Michael T. M. Emmerich, Kaifeng Yang, André H. Deutz, Hao Wang, and Carlos M. Fonseca. 2016. A multicriteria generalization of bayesian global optimization. In *Advances in Stochastic and Deterministic Global Optimization*. Springer, 229–242.

[84] Richard M. Everson and Jonathan E. Fieldsend. 2006. Multi-class ROC analysis from a multi-objective optimisation perspective. *Pattern Recognition Letters* 27, 8 (2006), 918–927.

[85] Stefan Falkner, Aaron Klein, and Frank Hutter. 2018. BOHB: Robust and Efficient Hyperparameter Optimization at Scale. In *Proceedings of the 35th International Conference on Machine Learning, ICML (Proceedings of Machine Learning Research, Vol. 80)*, Jennifer G. Dy and Andreas Krause (Eds.). PMLR, 1436–1445.

[86] Xinjie Fan, Yuguang Yue, Purnamrita Sarkar, and YX Rachel Wang. 2020. On hyperparameter tuning in general clustering problemsm. In *International Conference on Machine Learning*. PMLR, 2996–3007.

[87] Tom Fawcett. 2006. An introduction to ROC analysis. *Pattern recognition letters* 27, 8 (2006), 861–874.

[88] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. 2015. Efficient and robust automated machine learning. In *Advances in neural information processing systems*. 2962–2970.

[89] Jonathan E. Fieldsend. 2009. *Optimizing Decision Trees Using Multi-objective Particle Swarm Optimization*. Springer Berlin Heidelberg, Berlin, Heidelberg, 93–114.

[90] Jonathan E. Fieldsend and Richard M. Everson. 2005. Formulation and comparison of multi-class ROC surfaces. (2005).

[91] Jonathan E. Fieldsend and Richard M. Everson. 2015. The Rolling Tide Evolutionary Algorithm: A Multiobjective Optimizer for Noisy Optimization Problems. *IEEE Transactions on Evolutionary Computation* 19, 1 (2015), 103–117. https://doi.org/10.1109/TEVC.2014.2304415

[92] Carlos M. Fonseca and Peter J. Fleming. 1996. On the performance assessment and comparison of stochastic multiobjective optimizers. In *International Conference on Parallel Problem Solving from Nature*. Springer, 584–593.

[93] Alexander I. J. Forrester, András Sóbester, and Andy J. Keane. 2007. Multi-fidelity optimization via surrogate modelling. *Proceedings of the royal society a: mathematical, physical and engineering sciences* 463, 2088 (2007), 3251–3269.

[94] Sorelle A. Friedler, Carlos Scheidegger, Suresh Venkatasubramanian, Sonam Choudhary, Evan P. Hamilton, and Derek Roth. 2019. A comparative study of fairness-enhancing interventions in machine learning. In *Proceedings of the Conference on Fairness, Accountability, and Transparency, FAT* 2019, Atlanta, GA, USA, January 29-31, 2019*, danah boyd and Jamie H. Morgenstern (Eds.). ACM, 329–338. https://doi.org/10.1145/3287560.3287589

[95] Johannes Fürnkranz and Eyke Hüllermeier. 2010. *Preference learning*. Springer.

[96] António Gaspar-Cunha and José A. Covas. 2006. Robustness using Multi-Objective Evolutionary Algorithms. In *Applications of Soft Computing*. Springer Berlin Heidelberg, Berlin, Heidelberg, 353–362.

[97] Justin Gilmer, Nicolas Ford, Nicholas Carlini, and Ekin Cubuk. 2019. Adversarial Examples Are a Natural Consequence of Test Error in Noise *(Proceedings of Machine Learning Research, Vol. 97)*. PMLR, Long Beach, California, USA, 2280–2289. http://proceedings.mlr.press/v97/gilmer19a.html

[98] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1412.6572

[99] Julia Guerrero-Viu, Sven Hauns, Sergio Izquierdo, Guilherme Miotto, Simon Schrodi, Andre Biedenkapp, Thomas Elsken, Difan Deng, Marius Lindauer, and Frank Hutter. 2021. Bag of Baselines for Multi-objective Joint Neural Architecture Search and Hyperparameter Optimization. arXiv:2105.01015

[100] Ayla Gülcü and Zeki Kuş. 2021. Multi-objective simulated annealing for hyper-parameter optimization in convolutional neural networks. *PeerJ Computer Science* 7 (2021), e338.

[101] Vassil Guliashki, Hristo Toshev, and Chavdar Korsemov. 2008. Survey of Evolutionary Algorithms used in multiobjective optimization. 60 (11 2008).

[102] Minghao Guo, Yuzhe Yang, Rui Xu, Ziwei Liu, and Dahua Lin. 2020. When nas meets robustness: In search of robust architectures against adversarial attacks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 631–640.

[103] Walter J. Gutjahr and Alois Pichler. 2016. Stochastic multi-objective optimization: a survey on non-scalarizing methods. *Annals of Operations Research* 236, 2 (2016), 475–499.

[104] Isabelle Guyon and André Elisseeff. 2003. An Introduction to Variable and Feature Selection. *Journal of Machine Learning Research* 3 (2003), 1157–1182. http://jmlr.org/papers/v3/guyon03a.html

[105] David Hand and Robert Till. 2001. A Simple Generalisation of the Area Under the ROC Curve for Multiple Class Classification Problems. *Hand, The* 45 (11 2001), 171–186. https://doi.org/10.1023/A:1010920819831

[106] Michael P. Hansen and Andrzej Jaszkiewicz. 1998. Evaluating the quality of approximations to the non-dominated set.

[107] Moritz Hardt, Eric Price, Nati Srebro, et al. 2016. Equality of opportunity in supervised learning. In *Advances in neural information processing systems*. 3315–3323.

[108] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[109] Xin He, Kaiyong Zhao, and Xiaowen Chu. 2020. AutoML: A Survey of the State-of-the-Art. arXiv:1908.00709 [cs.LG]

[110] Daniel Hernández-Lobato, José Miguel Hernández-Lobato, Amar Shah, and Ryan P. Adams. 2016. Predictive entropy search for multi-objective bayesian optimization. In *International Conference on Machine Learning*. 1492–1501.

[111] José Miguel Hernández-Lobato, Michael A. Gelbart, Brandon Reagen, Robert Adolf, Daniel Hernández-Lobato, Paul N. Whatmough, David Brooks, Gu-Yeon Wei, and Ryan P. Adams. 2016. Designing neural network hardware accelerators with decoupled objective evaluations. In *NIPS workshop on Bayesian Optimization*. 10.

[112] Kenneth Holstein, Jennifer Wortman Vaughan, Hal Daumé III, Miroslav Dudík, and Hanna M. Wallach. 2019. Improving Fairness in Machine Learning Systems: What Do Industry Practitioners Need?. In *Proceedings of the Conference on Human Factors in Computing Systems, CHI 2019*, Stephen A. Brewster, Geraldine Fitzpatrick, Anna L. Cox, and Vassilis Kostakos (Eds.). ACM, 600. https://doi.org/10.1145/3290605.3300830

[113] Daniel Horn. 2010. *Multi-objective analysis of machine learning algorithms using model-based optimization techniques*. Ph. D. Dissertation. Technische Universität Dortmund.

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

[115] Daniel Horn, Melanie Dagge, Xudong Sun, and Bernd Bischl. 2017. First Investigations on Noisy Model-Based Multi-objective Optimization. *Lecture Notes in Computer Science* 10173, 298–313. https://doi.org/10.1007/978-3-319-54157-0_21

[116] Daniel Horn, Tobias Wagner, Dirk Biermann, Claus Weihs, and Bernd Bischl. 2015. Model-based multi-objective optimization: taxonomy, multi-point proposal, toolbox and benchmark. In *International Conference on Evolutionary Multi-Criterion Optimization*. Springer, 64–78.

[117] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv:1704.04861

[118] Chi-Hung Hsu, Shu-Huan Chang, Jhao-Hong Liang, Hsin-Ping Chou, Chun-Hao Liu, Shih-Chieh Chang, Jia-Yu Pan, Yu-Ting Chen, Wei Wei, and Da-Cheng Juan. 2018. Monas: Multi-objective neural architecture search using reinforcement learning. *arXiv preprint arXiv:1806.10332* (2018).

[119] Deng Huang, Theodore T. Allen, William I. Notz, and R. Allen Miller. 2006. Sequential kriging optimization using multiple-fidelity evaluations. *Structural and Multidisciplinary Optimization* 32, 5 (2006), 369–382.

[120] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. 2017. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4700–4708.

[121] Jin Huang and Charles X. Ling. 2005. Using AUC and accuracy in evaluating learning algorithms. *IEEE Transactions on knowledge and Data Engineering* 17, 3 (2005), 299–310.

[122] Evan J. Hughes. 2001. Evolutionary multi-objective ranking with uncertainty and noise. In *International Conference on Evolutionary Multi-Criterion Optimization*. Springer, 329–343.

[123] Susan R. Hunter, Eric A. Applegate, Viplove Arora, Bryan Chong, Kyle Cooper, Oscar Rincón-Guevara, and Carolina Vivas-Valencia. 2019. An Introduction to Multiobjective Simulation Optimization. 29, 1 (2019).

[124] Aminul Huq and Mst. Tasnim Pervin. 2020. Adversarial Attacks and Defense on Texts: A Survey. arXiv:2005.14108

[125] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2011. Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*. Springer, 507–523.

[126] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren (Eds.). 2018. *Automated Machine Learning: Methods, Systems, Challenges*. Springer. In press, available at http://automl.org/book..

[127] Beatriz Iglesia, M.S. Philpott, Anthony Bagnall, and Victor Rayward-Smith. 2004. Data mining rules using multi-objective evolutionary algorithms. 1552 – 1559 Vol.3. https://doi.org/10.1109/CEC.2003.1299857

[128] Md Shahriar Iqbal, Jianhai Su, Lars Kotthoff, and Pooyan Jamshidi. 2020. FlexiBO: Cost-Aware Multi-Objective Optimization of Deep Neural Networks. *CoRR* abs/2001.06588 (2020).

[129] Hisao Ishibuchi, Ryo Imada, Yu Setoguchi, and Yusuke Nojima. 2018. How to Specify a Reference Point in Hypervolume Calculation for Fair Performance Comparison. *Evolutionary Computation* 26, 3 (2018), 411–440. https://doi.org/10.1162/evco_a_00226

[130] Hisao Ishibuchi, Noritaka Tsukamoto, and Yusuke Nojima. 2008. Evolutionary many-objective optimization: A short review. In *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*. IEEE, 2419–2426.

[131] Himanshu Jain and Kalyanmoy Deb. 2014. An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point Based Nondominated Sorting Approach, Part II: Handling Constraints and Extending to an Adaptive Approach. *Evolutionary Computation, IEEE Transactions on* 18 (08 2014), 602–622. https://doi.org/10.1109/TEVC.2013.2281534

[132] Nathalie Japkowicz and Mohak Shah (Eds.). 2011. *Evaluating Learning Algorithms: A Classification Perspective*. Cambridge University Press.

[133] Shinkyu Jeong and Shigeru Obayashi. 2005. Efficient global optimization (EGO) for multi-objective problem and data mining. In *2005 IEEE congress on evolutionary computation*, Vol. 3. IEEE, 2138–2145.

[134] Shali Jiang, Henry Chai, Javier González, and Roman Garnett. 2019. Efficient nonmyopic Bayesian optimization and quadrature. arXiv:1909.04568

[135] Yaochu Jin. 2006. *Multi-objective machine learning*. Vol. 16. Springer Science & Business Media.

[136] Yaochu Jin and Jürgen Branke. 2005. Evolutionary optimization in uncertain environments-a survey. *IEEE Transactions on Evolutionary Computation* 9, 3 (2005), 303–317.

[137] Yaochu Jin and Bernhard Sendhoff. 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 (2008), 397–415.

[138] Donald R. Jones. 2001. A Taxonomy of Global Optimization Methods Based on Response Surfaces. *Journal of Global Optimization* 21, 4 (2001), 345–383. https://doi.org/10.1023/A:1012771025575

[139] Donald R. Jones, Matthias Schonlau, and William J. Welch. 1998. Efficient Global Optimization of Expensive Black-Box Functions. *Journal of Global Optimization* 13, 4 (Dec. 1998), 455–492. https://doi.org/10.1023/A:1008306431147

[140] Chia-Feng Juang and Chia-Hung Hsu. 2014. Structure and parameter optimization of FNNs using multi-objective ACO for control and prediction. In *2014 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. IEEE, 928–933.
[141] Faisal Kamiran and Toon Calders. 2012. Data Preprocessing Techniques for Classification without Discrimination. *Knowl. Inf. Syst.* 33, 1 (Oct. 2012), 1–33. https://doi.org/10.1007/s10115-011-0463-8
[142] Kirthevasan Kandasamy, Karun Raju Vysyaraju, Willie Neiswanger, Biswajit Paria, Christopher R. Collins, Jeff Schneider, Barnabás Póczos, and Eric P. Xing. 2020. Tuning Hyperparameters without Grad Students: Scalable and Robust Bayesian Optimisation with Dragonfly. *J. Mach. Learn. Res.* 21 (2020), 81:1–81:27.
[143] Marvin Kaster, Wei Zhao, and Steffen Eger. 2021. Global Explainability of BERT-Based Evaluation Metrics by Disentangling along Linguistic Factors. arXiv:2110.04399
[144] Alex Kendall and Yarin Gal. 2017. What uncertainties do we need in bayesian deep learning for computer vision? arXiv:1703.04977
[145] Joon Sik Kim, Jiahao Chen, and Ameet Talwalkar. 2020. FACT: A Diagnostic for Group Fairness Trade-offs. In *Proceedings of the 37th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 119)*, Hal Daumé III and Aarti Singh (Eds.). PMLR, 5264–5274. http://proceedings.mlr.press/v119/kim20a.html
[146] Ye-Hoon Kim, Bhargava Reddy, Sojung Yun, and Chanwon Seo. 2017. Nemo: Neuro-evolution with multiobjective optimization of deep neural network for speed and accuracy. In *ICML 2017 AutoML Workshop*.
[147] Pieter-Jan Kindermans, Sara Hooker, Julius Adebayo, Maximilian Alber, Kristof T Schütt, Sven Dähne, Dumitru Erhan, and Been Kim. 2019. The (un) reliability of saliency methods. In *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*. Springer, 267–280.
[148] Aaron Klein, Louis C. Tiao, Thibaut Lienart, Cedric Archambeau, and Matthias Seeger. 2020. Model-based Asynchronous Hyperparameter and Neural Architecture Search. arXiv:2003.10865
[149] Joshua Knowles. 2006. ParEGO: A Hybrid Algorithm with on-Line Landscape Approximation for Expensive Multiobjective Optimization Problems. 10, 1 (2006), 50–66. https://doi.org/10.1109/TEVC.2005.851274
[150] Joshua Knowles, David Corne, and Alan Reynolds. 2009. Noisy Multiobjective Optimization on a Budget of 250 Evaluations. In *Evolutionary Multi-Criterion Optimization*, Matthias Ehrgott, Carlos M. Fonseca, Xavier Gandibleux, Jin-Kao Hao, and Marc Sevaux (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 36–50.
[151] Joshua Knowles, Richard Watson, and David Corne. 2001. Reducing Local Optima in Single-Objective Problems by Multi-objectivization. *Lecture Notes in Computer Science*. https://doi.org/10.1007/3-540-44719-9_19
[152] Nicolas Knudde, Joachim van der Herten, Tom Dhaene, and Ivo Couckuyt. 2017. GPflowOpt: A Bayesian Optimization Library using TensorFlow. arXiv:1711.03845
[153] Patrick Koch, Tobias Wagner, Michael T. M. Emmerich, Thomas Bäck, and Wolfgang Konen. 2015. Efficient multi-criteria optimization on noisy machine learning problems. *Applied Soft Computing* 29 (2015), 357–370.
[154] Ron Kohavi and George H. John. 2002. Wrappers for feature subset selection. *Artificial Intelligence* 97, 1-2 (2002), 273–324.
[155] Todd Kulesza, Margaret Burnett, Weng-Keen Wong, and Simone Stumpf. 2015. Principles of explanatory debugging to personalize interactive machine learning. In *Proceedings of the 20th international conference on intelligent user interfaces*. 126–137.
[156] Preethi Lahoti, Alex Beutel, Jilin Chen, Kang Lee, Flavien Prost, Nithum Thain, Xuezhi Wang, and Ed Chi. 2020. Fairness without Demographics through Adversarially Reweighted Learning. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (Eds.). https://proceedings.neurips.cc/paper/2020/hash/07fc15c9d169ee48573edd749d25945d-Abstract.html
[157] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. 2017. Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles. In *Advances in Neural Information Processing Systems*, Vol. 30. Curran Associates, Inc.
[158] Adriana Lara, Gustavo Sanchez, Carlos A. Coello Coello, and Oliver Schutze. 2010. HCS: A New Local Search Strategy for Memetic Multiobjective Evolutionary Algorithms. *IEEE Transactions on Evolutionary Computation* 14, 1 (2010), 112–132. https://doi.org/10.1109/TEVC.2009.2024143
[159] Alfred Laugros, Alice Caplier, and Matthieu Ospici. 2019. Are adversarial robustness and common perturbation robustness independant attributes?. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*.
[160] Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. 1999. Object recognition with gradient-based learning. In *Shape, contour and grouping in computer vision*. Springer, 319–345.
[161] Loo-Hay Lee, Chew Ek Peng, Teng Suyan, and Li Juxin. 2009. Application of evolutionary algorithms for solving multi-objective simulation optimization problems. In *Multi-objective memetic algorithms*. Springer, 91–110.
[162] Julien-Charles Levesque, Audrey Durand, Christian Gagné, and Robert Sabourin. 2017. Bayesian optimization for conditional hyperparameter spaces. In *2017 International Joint Conference on Neural Networks, IJCNN 2017, Anchorage,*

AK, USA, May 14-19, 2017. IEEE, 286–293. https://doi.org/10.1109/IJCNN.2017.7965867

[163] Julien-Charles Lévesque, Audrey Durand, Christian Gagné, and Robert Sabourin. 2012. Multi-objective evolutionary optimization for generating ensembles of classifiers in the ROC space. In *Proceedings of the 14th annual conference on Genetic and evolutionary computation*. 879–886.

[164] Lisha Li, Kevin G. Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. 2017. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. *J. Mach. Learn. Res.* 18 (2017), 185:1–185:52. http://jmlr.org/papers/v18/16-558.html

[165] Liam Li, Kevin G. Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Jonathan Ben-tzur, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. 2020. A System for Massively Parallel Hyperparameter Tuning. In *Proceedings of Machine Learning and Systems 2020, MLSys 2020, Austin, TX, USA, March 2-4, 2020*, Inderjit S. Dhillon, Dimitris S. Papailiopoulos, and Vivienne Sze (Eds.). mlsys.org.

[166] Miqing Li and Xin Yao. 2019. Quality Evaluation of Solution Sets in Multiobjective Optimisation: A Survey. *Comput. Surveys* 52 (03 2019), 1–38. https://doi.org/10.1145/3300148

[167] Rui Li, Michael T. M. Emmerich, Jeroen Eggermont, Thomas Bäck, Martin Schütz, Jouke Dijkstra, and Johan H. C. Reiber. 2013. Mixed Integer Evolution Strategies for Parameter Optimization. *Evolutionary Computation* 21, 1 (2013), 29—-64.

[168] Yang Li, Yu Shen, Wentao Zhang, Yuanwei Chen, Huaijun Jiang, Mingchao Liu, Jiawei Jiang, Jinyang Gao, Wentao Wu, Zhi Yang, Ce Zhang, and Bin Cui. 2021. OpenBox: A Generalized Black-box Optimization Service. In *KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, Singapore, August 14-18, 2021*, Feida Zhu, Beng Chin Ooi, and Chunyan Miao (Eds.). ACM, 3209–3219. https://doi.org/10.1145/3447548.3467061

[169] Jason Liang, Elliot Meyerson, Babak Hodjat, Dan Fink, Karl Mutch, and Risto Miikkulainen. 2019. Evolutionary neural automl for deep learning. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 401–409.

[170] Thomas Liddle, Mark Johnston, and Mengjie Zhang. 2010. Multi-objective genetic programming for object detection. In *IEEE Congress on Evolutionary Computation*. IEEE, 1–8.

[171] Zachary C. Lipton. 2018. The mythos of model interpretability. *Commun. ACM* 61, 10 (2018), 36–43. https://doi.org/10.1145/3233231

[172] Haitao Liu, Jianfei Cai, and Yew-Soon Ong. 2018. Remarks on multi-output Gaussian process regression. *Knowl. Based Syst.* 144 (2018), 102–121. https://doi.org/10.1016/j.knosys.2017.12.034

[173] Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. 2018. Hierarchical Representations for Efficient Architecture Search. In *International Conference on Learning Representations*.

[174] Li Liu, Wanli Ouyang, Xiaogang Wang, Paul Fieguth, Jie Chen, Xinwang Liu, and Matti Pietikäinen. 2020. Deep learning for generic object detection: A survey. *International journal of computer vision* 128, 2 (2020), 261–318.

[175] Aleksei Liuliakov and Barbara Hammer. 2021. AutoML Technologies for the Identification of Sparse Models. In *International Conference on Intelligent Data Engineering and Automated Learning*. Springer, 65–75.

[176] Anton Lokhmotov, Nikolay Chunosov, Flavio Vella, and Grigori Fursin. 2018. Multi-objective autotuning of mobilenets across the full software/hardware stack. In *Proceedings of the 1st on Reproducible Quality-Efficient Systems Tournament on Co-designing Pareto-efficient Deep Learning*. 1.

[177] Mohammad Loni, Sima Sinaei, Ali Zoljodi, Masoud Daneshtalab, and Mikael Sjödin. 2020. DeepMaker: A multi-objective optimization framework for deep neural networks in embedded systems. *Microprocess. Microsystems* 73 (2020), 102989. https://doi.org/10.1016/j.micpro.2020.102989

[178] Mohammad Loni, Ali Zoljodi, Sima Sinaei, Masoud Daneshtalab, and Mikael Sjödin. 2019. NeuroPower: Designing Energy Efficient Convolutional Neural Network Architecture for Embedded Systems. In *Artificial Neural Networks and Machine Learning - ICANN 2019: Theoretical Neural Computation - 28th International Conference on Artificial Neural Networks, Munich, Germany, September 17-19, 2019, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 11727)*, Igor V. Tetko, Vera Kurková, Pavel Karpov, and Fabian J. Theis (Eds.). Springer, 208–222. https://doi.org/10.1007/978-3-030-30487-4_17

[179] Christos Louizos, Max Welling, and Diederik P. Kingma. 2017. Learning Sparse Neural Networks through L_0 Regularization. arXiv:1712.01312

[180] Zhichao Lu, Kalyanmoy Deb, and Vishnu Naresh Boddeti. 2020. Muxconv: Information multiplexing in convolutional neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 12044–12053.

[181] Zhichao Lu, Kalyanmoy Deb, Erik Goodman, Wolfgang Banzhaf, and Vishnu Naresh Boddeti. 2020. Nsganetv2: Evolutionary multi-objective surrogate-assisted neural architecture search. In *European Conference on Computer Vision*. Springer, 35–51.

[182] Zhichao Lu, Ian Whalen, Vishnu Boddeti, Yashesh Dhebar, Kalyanmoy Deb, Erik Goodman, and Wolfgang Banzhaf. 2019. Nsga-net: neural architecture search using multi-objective genetic algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 419–427.

[183] Scott M. Lundberg and Su-In Lee. 2017. A Unified Approach to Interpreting Model Predictions. In *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.). Curran Associates, Inc., 4765–4774. http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf

[184] Gang Luo. 2016. A review of automatic selection methods for machine learning algorithms and hyper-parameter values. *Network Modeling Analysis in Health Informatics and Bioinformatics* 5, 1 (2016).

[185] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2018. Towards Deep Learning Models Resistant to Adversarial Attacks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=rJzIBfZAb

[186] Gustavo Malkomes and Roman Garnett. 2018. Automating Bayesian optimization with Bayesian optimization. In *Advances in Neural Information Processing Systems*. 5984–5994.

[187] Chengzhi Mao, Ziyuan Zhong, Junfeng Yang, Carl Vondrick, and Baishakhi Ray. 2019. Metric learning for adversarial robustness. *Advances in Neural Information Processing Systems* 32 (2019).

[188] Natalia Martinez, Martin Bertran, and Guillermo Sapiro. 2020. Minimax Pareto Fairness: A Multi Objective Perspective. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event (Proceedings of Machine Learning Research, Vol. 119)*. PMLR, 6755–6764. http://proceedings.mlr.press/v119/martinez20a.html

[189] Ninareh Mehrabi, Fred Morstatter, Nripsuta Saxena, Kristina Lerman, and Aram Galstyan. 2021. A survey on bias and fairness in machine learning. *ACM Computing Surveys (CSUR)* 54, 6 (2021), 1–35.

[190] Ingo Mierswa. 2009. *Non-Convex and Multi-Objective Optimization in Data Mining*. Ph. D. Dissertation. Technische Universität Dortmund.

[191] Ingo Mierswa and Michael Wurst. 2006. Information preserving multi-objective feature selection for unsupervised learning. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*. 1545–1552.

[192] Kaisa Miettinen. 2012. *Nonlinear Multiobjective Optimization*. Springer US. https://books.google.de/books?id=bnzjBwAAQBAJ

[193] Fan Min, Qinghua Hu, and William Zhu. 2014. Feature selection with test cost constraint. *International Journal of Approximate Reasoning* 55, 1 (2014), 167–179.

[194] Christoph Molnar. 2019. *Interpretable Machine Learning*. https://christophm.github.io/interpretable-ml-book/.

[195] Christoph Molnar, Giuseppe Casalicchio, and Bernd Bischl. 2019. Quantifying Interpretability of Arbitrary Machine Learning Models Through Functional Decomposition. arXiv:1904.03867

[196] Pablo Moreno-Muñoz, Antonio Artés, and Mauricio Alvarez. 2018. Heterogeneous multi-output gaussian process prediction. In *Advances in neural information processing systems*. 6711–6720.

[197] Luigi Nardi, David Koeplinger, and Kunle Olukotun. 2019. Practical Design Space Exploration. In *27th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, MASCOTS 2019, Rennes, France, October 21-25, 2019*. IEEE Computer Society, 347–358. https://doi.org/10.1109/MASCOTS.2019.00045

[198] Renata F. P. Neves, Alberto N. G. Lopes Filho, Carlos A. B. Mello, and Cleber Zanchettin. 2011. A SVM based off-line handwritten digit recognizer. In *2011 IEEE International Conference on Systems, Man, and Cybernetics*. IEEE, 510–515.

[199] Xing Niu, Prashant Mathur, Georgiana Dinu, and Yaser Al-Onaizan. 2020. Evaluating Robustness to Input Perturbations for Neural Machine Translation. arXiv:2005.00580 [cs.CL]

[200] Randal S. Olson and Jason H. Moore. 2016. TPOT: A tree-based pipeline optimization tool for automating machine learning. In *Workshop on automatic machine learning*. PMLR, 66–74.

[201] Randal S. Olson, Ryan J. Urbanowicz, Peter C. Andrews, Nicole A. Lavender, La Creis Kidd, and Jason H. Moore. 2016. *Applications of Evolutionary Computation: 19th European Conference, EvoApplications 2016, Porto, Portugal, March 30 – April 1, 2016, Proceedings, Part I*. Springer International Publishing, Chapter Automating Biomedical Data Science Through Tree-Based Pipeline Optimization, 123–137. https://doi.org/10.1007/978-3-319-31204-0_9

[202] Alexandra Olteanu, Carlos Castillo, Fernando Diaz, and Emre Kiciman. 2019. Social data: Biases, methodological pitfalls, and ethical boundaries. *Frontiers in Big Data* 2 (2019), 13.

[203] Julio-Omar Palacio-Niño and Fernando Berzal. 2019. Evaluation metrics for unsupervised learning algorithms. *arXiv preprint arXiv:1905.05667* (2019).

[204] Anna Palczewska, Jan Palczewski, Richard Marchese Robinson, and Daniel Neagu. 2014. Interpreting random forest classification models using a feature contribution method. In *Integration of reusable systems*. Springer, 193–218.

[205] Taejin Park and Kwang Ryel Ryu. 2011. Accumulative Sampling for Noisy Evolutionary Multi-Objective Optimization. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation* (Dublin, Ireland) *(GECCO '11)*. Association for Computing Machinery, New York, NY, USA, 793–800. https://doi.org/10.1145/2001576.2001684

[206] Maryam Parsa, Aayush Ankit, Amirkoushyar Ziabari, and Kaushik Roy. 2019. Pabo: Pseudo agent-based multi-objective bayesian hyperparameter optimization for efficient neural accelerator design. In *2019 IEEE/ACM International*

Conference on Computer-Aided Design (ICCAD). IEEE, 1–8.

[207] Maryam Parsa, John P. Mitchell, Catherine D. Schuman, Robert M. Patton, Thomas E. Potok, and Kaushik Roy. 2020. Bayesian multi-objective hyperparameter optimization for accurate, fast, and efficient neural network accelerator design. *Frontiers in neuroscience* 14 (2020), 667.

[208] Magdalini Paschali, Sailesh Conjeti, Fernando Navarro, and Nassir Navab. 2018. Generalizability vs. Robustness: Adversarial Examples for Medical Imaging. arXiv:1804.00504 [cs.CV]

[209] Jonathan Peck, Joris Roels, Bart Goossens, and Yvan Saeys. 2017. Lower bounds on the robustness to adversarial perturbations. In *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.). Curran Associates, Inc., 804–813. http://papers.nips.cc/paper/6682-lower-bounds-on-the-robustness-to-adversarial-perturbations.pdf

[210] Guilherme D. Pelegrina, Renan DB Brotto, Leonardo T. Duarte, Romis Attux, and João M. T. Romano. 2020. A novel multi-objective-based approach to analyze trade-offs in Fair Principal Component Analysis. arXiv:2006.06137

[211] Valerio Perrone, Michele Donini, Krishnaram Kenthapadi, and Cédric Archambeau. 2020. Fair Bayesian Optimization. arXiv:2006.05109 [stat.ML]

[212] Dana Pessach and Erez Shmueli. 2020. Algorithmic fairness. arXiv:2001.09784

[213] Florian Pfisterer, Stefan Coors, Janek Thomas, and Bernd Bischl. 2019. Multi-Objective Automatic Machine Learning with AutoxgboostMC. arXiv:1908.10796 [stat.ML]

[214] Florian Pfisterer, Lennart Schneider, Julia Moosbauer, Martin Binder, and Bernd Bischl. 2021. YAHPO Gym - Design Criteria and a new Multifidelity Benchmark for Hyperparameter Optimization. *CoRR* abs/2109.03670 (2021).

[215] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. 2018. Efficient neural architecture search via parameter sharing. arXiv:1802.03268

[216] Iana S Polonskaia, Nikolay O Nikitin, Ilia Revin, Pavel Vychuzhanin, and Anna V Kalyuzhnaya. 2021. Multi-objective evolutionary design of composite data-driven models. In *2021 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 926–933.

[217] Wolfgang Ponweiser, Tobias Wagner, Dirk Biermann, and Markus Vincze. 2008. 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)*. Springer, Berlin, Heidelberg, 784–794. https://doi.org/10.1007/978-3-540-87700-4_78

[218] Hao Qin, Takahiro Shinozaki, and Kevin Duh. 2017. Evolution strategy based automatic tuning of neural machine translation systems. In *Proc. 14th Int. Workshop Spoken Language Transl.* 120–128.

[219] Hamed Rahimian and Sanjay Mehrotra. 2019. Distributionally Robust Optimization: A Review. arXiv:1908.05659 [math.OC]

[220] Aghila Rajagopal, Gyanendra Prasad Joshi, A. Ramachandran, R. T. Subhalakshmi, Manju Khari, Sudan Jha, K. Shankar, and Jinsang You. 2020. A Deep Learning Model Based on Multi-Objective Particle Swarm Optimization for Scene Classification in Unmanned Aerial Vehicles. *IEEE Access* 8 (2020), 135383–135393. https://doi.org/10.1109/ACCESS.2020.3011502

[221] Jonas Rauber, Wieland Brendel, and Matthias Bethge. 2018. Foolbox: A Python toolbox to benchmark the robustness of machine learning models. arXiv:1707.04131 [cs.LG]

[222] Brandon Reagen, José Miguel Hernández-Lobato, Robert Adolf, Michael Gelbart, Paul Whatmough, Gu-Yeon Wei, and David Brooks. 2017. A case for efficient accelerator design space exploration via bayesian optimization. In *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, 1–6.

[223] Sebastian Rojas Gonzalez, Hamed Jalali, and Inneke Van Nieuwenhuyse. 2020. A multiobjective stochastic simulation optimization algorithm. *European Journal of Operational Research* 284, 1 (2020), 212–226. https://doi.org/10.1016/j.ejor.2019.12.014

[224] Sebastian Rojas-Gonzalez and Inneke Van Nieuwenhuyse. 2020. A survey on kriging-based infill algorithms for multiobjective simulation optimization. *Computers & Operations Research* 116 (2020), 104869.

[225] Ananya B. Sai, Tanay Dixit, Dev Yashpal Sheth, Sreyas Mohan, and Mitesh M. Khapra. 2021. Perturbation checklists for evaluating NLG evaluation metrics. arXiv:2109.05771

[226] Pedro Saleiro, Benedict Kuester, Abby Stevens, Ari Anisfeld, Loren Hinkson, Jesse London, and Rayid Ghani. 2018. Aequitas: A Bias and Fairness Audit Toolkit. arXiv:1811.05577

[227] David Salinas, Valerio Perrone, Olivier Cruchant, and Cédric Archambeau. 2021. A multi-objective perspective on jointly tuning hardware and hyperparameters. arXiv:2106.05680

[228] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4510–4520.

[229] Iqbal H. Sarker. 2021. Machine learning: Algorithms, real-world applications and research directions. *SN Computer Science* 2, 3 (2021), 1–21.

[230] Robin Schmucker, Michele Donini, Valerio Perrone, Muhammad Bilal Zafar, and Cedric Archambeau. 2020. Multi-Objective Multi-Fidelity Hyperparameter Optimization with Application to Fairness. *NeurIPS Workshop on Meta-Learning* (2020).

[231] Robin Schmucker, Michele Donini, Muhammad Bilal Zafar, David Salinas, and Cédric Archambeau. 2021. Multi-objective Asynchronous Successive Halving. arXiv:2106.12639

[232] Andrew D. Selbst, Danah Boyd, Sorelle A. Friedler, Suresh Venkatasubramanian, and Janet Vertesi. 2019. Fairness and abstraction in sociotechnical systems. In *Proceedings of the Conference on Fairness, Accountability, and Transparency.* 59–68.

[233] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. 2017. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision.* 618–626.

[234] Amar Shah and Zoubin Ghahramani. 2016. Pareto frontier learning with expensive correlated objectives. In *International conference on machine learning*. PMLR, 1919–1927.

[235] Yakun Sophia Shao, Brandon Reagen, Gu-Yeon Wei, and David Brooks. 2014. Aladdin: A pre-rtl, power-performance accelerator simulator enabling large design space exploration of customized architectures. In *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA).* IEEE, 97–108.

[236] Takahiro Shinozaki, Shinji Watanabe, and Kevin Duh. 2020. Automated Development of DNN Based Spoken Language Systems Using Evolutionary Algorithms. In *Deep Neural Evolution*. Springer, 97–129.

[237] Margarita R. Sierra and Carlos A. Coello Coello. 2005. Improving PSO-based multi-objective optimization using crowding, mutation and epsilon-dominance. In *International conference on evolutionary multi-criterion optimization.* Springer, 505–519.

[238] Lewis Smith and Yarin Gal. 2018. Understanding measures of uncertainty for adversarial example detection. arXiv:1803.08533

[239] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. 2012. Practical Bayesian Optimization of Machine Learning Algorithms. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou, and Kilian Q. Weinberger (Eds.). 2960–2968.

[240] Evgenii Sopov and Ilia Ivanov. 2015. Self-Configuring Ensemble of Neural Network Classifiers for Emotion Recognition in the Intelligent Human-Machine Interaction. In *IEEE Symposium Series on Computational Intelligence, SSCI 2015, Cape Town, South Africa, December 7-10, 2015*. IEEE, 1808–1815. https://doi.org/10.1109/SSCI.2015.252

[241] Thorsten Suttorp and Christian Igel. 2006. Multi-Objective Optimization of Support Vector Machines. In *Multi-Objective Machine Learning*, Yaochu Jin (Ed.). Studies in Computational Intelligence, Vol. 16. Springer, 199–220.

[242] Ivan Svetunkov. 2022. *Forecasting and Analytics with ADAM.* Retrieved May 18, 2008 from https://openforecast.org/adam/

[243] Anna Syberfeldt, Amos Ng, Robert John, and Philip Moore. 2010. Evolutionary optimisation of noisy multi-objective problems using confidence-based dynamic resampling. *European Journal of Operational Research* 204 (08 2010), 533–544. https://doi.org/10.1016/j.ejor.2009.11.003

[244] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew G. Howard, and Quoc V. Le. 2019. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.* 2820–2828.

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

[246] Tomohiro Tanaka, Takafumi Moriya, Takahiro Shinozaki, Shinji Watanabe, Takaaki Hori, and Kevin Duh. 2016. Automated structure discovery and parameter tuning of neural network language model based on evolution strategy. In *2016 IEEE Spoken Language Technology Workshop (SLT)*. IEEE, 665–671.

[247] Rohan Taori, Achal Dave, Vaishaal Shankar, Nicholas Carlini, Benjamin Recht, and Ludwig Schmidt. 2020. Measuring Robustness to Natural Distribution Shifts in Image Classification. arXiv:2007.00644 [cs.LG]

[248] Antonio Torralba and Alexei A. Efros. 2011. Unbiased look at dataset bias. In *CVPR 2011*. 1521–1528. https://doi.org/10.1109/CVPR.2011.5995347

[249] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. 2017. Ensemble adversarial training: Attacks and defenses. arXiv:1705.07204

[250] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. 2019. Robustness May Be at Odds with Accuracy. In *7th International Conference on Learning Representations, ICLR*. OpenReview.net. https://openreview.net/forum?id=SyxAb30cY7

[251] Tea Tušar and Bogdan Filipič. 2014. Visualization of Pareto Front Approximations in Evolutionary Multiobjective Optimization: A Critical Review and the Prosection Method. *IEEE Transactions on Evolutionary Computation* 19 (01 2014), 1–1. https://doi.org/10.1109/TEVC.2014.2313407

[252] Vimal Vachhani, Vipul Dabhi, and Harshadkumar Prajapati. 2015. Survey of Multi Objective Evolutionary Algorithms. https://doi.org/10.1109/ICCPCT.2015.7159422
[253] Carlos Villacampa-Calvo, Bryan Zaldivar, Eduardo C. Garrido-Merchán, and Daniel Hernández-Lobato. 2020. Multi-class Gaussian Process Classification with Noisy Inputs. arXiv:2001.10523
[254] Bin Wang, Yanan Sun, Bing Xue, and Mengjie Zhang. 2019. Evolving Deep Neural Networks by Multi-objective Particle Swarm Optimization for Image Classification. arXiv:1904.09035 [cs.NE]
[255] Chunnan Wang, Hongzhi Wang, Guocheng Feng, and Fei Geng. 2020. Multi-Objective Neural Architecture Search Based on Diverse Structures and Adaptive Recommendation. arXiv:2007.02749
[256] Handing Wang, Markus Olhofer, and Yaochu Jin. 2017. A mini-review on preference modeling and articulation in multi-objective optimization: current status and challenges. *Complex & Intelligent Systems* 3, 4 (2017), 233–245.
[257] Ziyu Wang, Frank Hutter, Masrour Zoghi, David Matheson, and Nando de Freitas. 2016. Bayesian Optimization in a Billion Dimensions via Random Embeddings. *J. Artif. Intell. Res.* 55 (2016), 361–387. https://doi.org/10.1613/jair.4806
[258] Colin White, Willie Neiswanger, and Yash Savani. 2021. BANANAS: Bayesian Optimization with Neural Architectures for Neural Architecture Search. In *AAAI.*
[259] Rey Reza Wiyatno, Anqi Xu, Ousmane Dia, and Archy de Berker. 2019. Adversarial Examples in Modern Machine Learning: A Review. arXiv:1911.05268
[260] Bing Xue, Mengjie Zhang, and Will N. Browne. 2012. Particle swarm optimization for feature selection in classification: A multi-objective approach. *IEEE transactions on cybernetics* 43, 6 (2012), 1656–1671.
[261] Bing Xue, Mengjie Zhang, Will N Browne, and Xin Yao. 2015. A survey on evolutionary computation approaches to feature selection. *IEEE Transactions on Evolutionary Computation* 20, 4 (2015), 606–626.
[262] Kaifeng Yang, Michael T. M. Emmerich, André H. Deutz, and Thomas Bäck. 2019. Efficient computation of expected hypervolume improvement using box decomposition algorithms. *Journal of Global Optimization* 75, 1 (2019), 3–34.
[263] Kaifeng Yang, Pramudita Satria Palar, Michael T. M. Emmerich, Koji Shimoyama, and Thomas Bäck. 2019. A Multi-Point Mechanism of Expected Hypervolume Improvement for Parallel Multi-Objective Bayesian Global Optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference.* ACM, Prague Czech Republic, 656–663. https://doi.org/10.1145/3321707.3321784
[264] Quanming Yao, Mengshuo Wang, Yuqiang Chen, Wenyuan Dai, Hu Yi-Qi, Li Yu-Feng, Tu Wei-Wei, Yang Qiang, and Yu Yang. 2018. Taking human out of learning applications: A survey on automated machine learning. arXiv:1810.13306
[265] Kun Zhang, Bernhard Schölkopf, Krikamol Muandet, and Zhikun Wang. 2013. Domain Adaptation Under Target and Conditional Shift. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28* (Atlanta, GA, USA) *(ICML'13).* JMLR.org, III–819–III–827. http://dl.acm.org/citation.cfm?id=3042817.3043028
[266] Qingfu Zhang and Hui Li. 2007. MOEA/D: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on evolutionary computation* 11, 6 (2007), 712–731.
[267] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. 2018. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition.* 6848–6856.
[268] Yangming Zhou and Yangguang Liu. 2014. Correlation analysis of performance metrics for classifier. 487–492. https://doi.org/10.1142/9789814619998_0081
[269] Eckart Zitzler, Dimo Brockhoff, and Lothar Thiele. 2006. The Hypervolume Indicator Revisited: On the Design of Pareto-compliant Indicators Via Weighted Integration. In *Evolutionary Multi-Criterion Optimization, 4th International Conference, EMO 2007, Matsushima, Japan, March 5-8, 2007, Proceedings (Lecture Notes in Computer Science, Vol. 4403)*, Shigeru Obayashi, Kalyanmoy Deb, Carlo Poloni, Tomoyuki Hiroyasu, and Tadahiko Murata (Eds.). Springer, 862–876. https://doi.org/10.1007/978-3-540-70928-2_64
[270] Eckart Zitzler and Lothar Thiele. 1998. Multiobjective Optimization Using Evolutionary Algorithms - A Comparative Case Study. In *Parallel Problem Solving from Nature - PPSN V, 5th International Conference, Amsterdam, The Netherlands, September 27-30, 1998, Proceedings (Lecture Notes in Computer Science, Vol. 1498)*, A. E. Eiben, Thomas Bäck, Marc Schoenauer, and Hans-Paul Schwefel (Eds.). Springer, 292–304. https://doi.org/10.1007/BFb0056872
[271] Eckart Zitzler, Lothar Thiele, Marco Laumanns, Carlos M. Fonseca, and Viviane Grunert da Fonseca. 2003. Performance assessment of multiobjective optimizers: an analysis and review. *IEEE Trans. Evol. Comput.* 7, 2 (2003), 117–132. https://doi.org/10.1109/TEVC.2003.810758
[272] Indrė Žliobaitė. 2017. Measuring discrimination in algorithmic decision making. *Data Mining and Knowledge Discovery* 31, 4 (2017), 1060–1089.

4.2. Multi-Objective Hyperparameter Tuning and Feature Selection using Filter Ensembles

This work presents a multi-objective HPO method for simultaneously optimizing the performance of an ML model and sparsity. Understanding model sparsity as a proxy for model explainability (Molnar et al., 2019), this work methodologically contributes to model explainability in the context of HPO and AutoML (see Section 3.1). It methodically fits into the subcategory *Multi-objective AutoML (C)*.

Contributing article:

Binder*, M., Moosbauer*, J., Thomas, J., Bischl, B. (2020). Multi-objective hyperparameter tuning and feature selection using filter ensembles. *Proceedings of the 2020 Genetic and Evolutionary Computation Conference (GECCO '20), pp. 471–479. ACM.*

Author contributions:

Julia Moosbauer and Martin Binder designed research questions. The literature review was performed by Julia Moosbauer. The ideas for algorithmic components were developed jointly by Martin Binder and Julia Moosbauer. Martin Binder has implemented an R package. The benchmark was designed jointly by Martin Binder and Julia Moosbauer and implemented, executed, and analyzed by Julia Moosbauer with the support of Martin Binder. The manuscript was written jointly by Julia Moosbauer and Martin Binder. Bernd Bischl and Janek added valuable input and suggested several notable modifications. All authors contributed to revisions of the paper.

Supplementary material available at:

- Supplementary material: `https://dl.acm.org/doi/10.1145/3377930.3389815`
- R package: `https://github.com/cran/mosmafs`

Multi-Objective Hyperparameter Tuning and Feature Selection using Filter Ensembles

Martin Binder*
Ludwig-Maximilians-Universität München
Munich, Germany
martin.binder@stat.uni-muenchen.de

Julia Moosbauer*
Ludwig-Maximilians-Universität München
Munich, Germany
julia.moosbauer@stat.uni-muenchen.de

Janek Thomas
Fraunhofer Institute for Integrated Circuits IIS
Erlangen, Germany
janek.thomas@scs.fraunhofer.de

Bernd Bischl
Ludwig-Maximilians-Universität München
Munich, Germany
bernd.bischl@stat.uni-muenchen.de

ABSTRACT

Both feature selection and hyperparameter tuning are key tasks in machine learning. Hyperparameter tuning is often useful to increase model performance, while feature selection is undertaken to attain sparse models. Sparsity may yield better model interpretability and lower cost of data acquisition, data handling and model inference. While sparsity may have a beneficial or detrimental effect on predictive performance, a small drop in performance may be acceptable in return for a substantial gain in sparseness. We therefore treat feature selection as a multi-objective optimization task. We perform hyperparameter tuning and feature selection simultaneously because the choice of features of a model may influence what hyperparameters perform well.

We present, benchmark, and compare two different approaches for multi-objective joint hyperparameter optimization and feature selection: The first uses multi-objective model-based optimization. The second is an evolutionary NSGA-II-based wrapper approach to feature selection which incorporates specialized sampling, mutation and recombination operators. Both methods make use of parameterized filter ensembles.

While model-based optimization needs fewer objective evaluations to achieve good performance, it incurs computational overhead compared to the NSGA-II, so the preferred choice depends on the cost of evaluating a model on given data.

CCS CONCEPTS

• **Theory of computation → Evolutionary algorithms**; Nonconvex optimization; *Mixed discrete-continuous optimization*; • **Computing methodologies →** *Supervised learning*; **Feature selection**;

KEYWORDS

Feature Selection, Hyperparameter Optimization, Multiobjective Optimization, Evolutionary Algorithms, Model-based Optimization

ACM Reference Format:
Martin Binder*, Julia Moosbauer*, Janek Thomas, and Bernd Bischl. 2020. Multi-Objective Hyperparameter Tuning and Feature Selection using Filter Ensembles. In *Genetic and Evolutionary Computation Conference (GECCO '20), July 8–12, 2020, Cancún, Mexico.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3377930.3389815

1 INTRODUCTION

Machine learning models often need to satisfy multiple objectives simultaneously to accomodate the nature of a practical setting. Usually the main goal is predictive performance. Especially on large and complex datasets this necessitates highly nonlinear algorithms, which have hyperparameters that need to be chosen carefully. *Hyperparameter optimization* poses a substantial challenge in machine learning: Besides few model-specific methods [16, 30], there are no general analytic representations of model performance w.r.t. hyperparameter settings. Performance therefore needs to be estimated using test-set evaluation or cross-validation. Hyperparameter optimization is therefore an expensive black-box optimization problem.

Besides predictive performance, model sparsity is frequently another desirable objective. According to Guyon and Elisseeff [17], sparser models help with interpretability, i.e. a better understanding of the underlying process that generated the data. In addition to that, predictions can be made faster and more cost-effectively. Sparser models may even have better predictive performance, since they regularize against overfitting.

The process of *feature selection* aims to select a small subset of relevant features while still constructing models with sufficient or even optimal predictive performance. There are two distinct model-agnostic approaches to feature selection [17]: Filters and wrappers. *Filters* use proxy measures to rank features by their estimated explanatory power, independently of the learning algorithm being employed. These include information theoretic measures, correlation measures, distance measures or consistency measures [8]. In contrast, *wrappers* [28] optimize model test-set performance directly over the space of feature subsets. Because every feature subset evaluation requires either one or multiple model fits, exhaustive search is usually infeasible, and a black-box discrete optimization search strategy is necessary. Commonly used are simple greedy methods like forward or backward search. More advanced methods like evolutionary algorithms can improve upon this [37]. Because

*These authors contributed equally to this work.

they directly optimize learner performance, wrappers often yield better results [37].

Feature selection is often considered as a single-objective task. Sometimes the feature selection step is only used to optimize performance [28]. However, in many applications it is desirable to forego a small drop in performance for a substantial gain in sparseness. This leads to a natural treatment of the feature selection problem as a *multi-objective optimization problem*: Maximize predictive performance while minimizing the number of features selected. Feature selection methods may aggregate model performance and number of features into a single objective function through a penalization term [37]. However, this implies a trade-off between performance and sparsity must be specified a-priori, which may be difficult.

Multi-objective optimization methods try to find a *set* of solutions that represent different trade-offs between the goals, enabling the user to consider the possible alternatives and to choose a fitting solution a-posteriori. This is beneficial for feature selection [36].

Hyperparameter optimization and feature selection are often performed in separate steps. We argue that jointly optimizing over the combined spaces of hyperparameters and feature subsets is beneficial and appropriate: The optimal choice of hyperparameter configuration is very likely to depend on the specific features that are included and vice-versa. Also, it is likely more computationally efficient to explore the joint spaces simultaneously. When using the wrapper approach this combination is not trivial: The exponentially large binary search space of selected features now has to be fused with the mixed numeric-categorical space of hyperparameters.

We present and adapt model-agnostic holistic approaches for both aspects discussed above: multi-objective and joint optimization of hyperparameters and feature sets. To guide the search, our methods make use of combinations of feature filter scores which are combined in a *filter ensemble*.

Our approaches can be considered *hybrid* filter-wrappers: they are fundamentally "wrapper"-based, because they optimize model-performance, but also make use of filters.

2 RELATED WORK AND CONTRIBUTIONS

In recent literature, *Bayesian optimization* ("BO"), also referred to as *model-based optimization*, has become a popular method for hyperparameter tuning of learning algorithms [33], often outperforming simple grid search or random search. Originally, BO generally relied on a Gaussian process (GP) model, such as in the Efficient Global Optimization (EGO) algorithm [24]. However, the GP does not typically scale well to high dimensions and large numbers of data points. Random forests have been proposed as an alternative surrogate model, as used in the popular *SMAC* hyperparameter optimization method [22].

Automated machine learning (AutoML) deals with the configuration and optimization of complete machine learning pipelines, often encompassing data pre-processing, ML models, ensembling, and possibly post-processing steps. AutoML may encompass choosing among the many different possibilities of what method to use at each stage of the pipeline, optimizing the hyperparameters for these methods, and combining them in ways that yield well-performing ensembles. Feature selection by filter or wrapper methods is an important pre-processing step already part of some AutoML frameworks. Consequently, AutoML has the potential to jointly optimize hyperparameters and included features. *autosklearn* [14] for example integrates a filter-based feature selector, parameterized by a filter measure and a percentage indicating the fraction of highest-ranked features to be included. Bayesian optimization is used to find the optimal filter (among a set of possible filters) and the best feature selection rate. The AutoML framework *TPOT* [31] uses an evolutionary algorithm to search over different machine learning pipelines, and also provides different feature selection strategies in its search space. TPOT can perform multi-objective optimization with regard to model performance and model complexity, measured as the number of processing steps, but it does not take model sparsity into account.

However, even though hyperparameter optimization and feature selection are both present in some AutoML frameworks, it is not their goal to find a good trade-off between predictive performance and sparseness. Feature selection is merely used to improve predictive performance, without considering the preference for sparse models in light of better interpretability or other benefits. In fact, these frameworks often tend to produce considerably complex models. They may even introduce additional features through feature engineering in pursuit of increasing predictive performance as the only goal. The results are often large and heterogeneous ensembles that are hard to interpret and deploy [32].

Evolutionary algorithms are especially well suited for multi-objective optimization. According to the survey by Xue et al. [37], *genetic algorithms* (GAs) are among the most commonly applied techniques for multi-objective feature selection. Inspired by natural evolution, GAs apply recombination and mutation operators to iteratively improve the population of solution candidates. Through techniques like non-dominated sorting [9, 25], genetic algorithms have become a powerful tool for multi-objective optimization. Several GA-based methods have been proposed for the task of wrapper-based feature selection [13, 18, 35].

There have been first investigations on simultaneous multi-objective hyperparameter optimization and feature selection using GAs: Bouraoui et al. [4] proposed an SVM-wrapper approach based on the NSGA-II, using a shared representation of the feature configuration and algorithm hyperparameters. While their method of combining the search spaces is similar to our GA-based approach, Bouraoui et al. [4] do not use specialized initialization and mutation operators that we have found to be necessary for good performance[1]. Another limitation is that their approach is not model agnostic but limited to SVMs only.

Numerous methods have been proposed for *feature filtering*, and there are known ways of combining filters into *filter ensembles*. These may rely on applying the same filter on diverse datasets ("homogeneous approach"), or different filters on the same dataset ("heterogeneous approach"), and there are different ways of aggregating the filter rankings [3]. We extend the heterogeneous approach into a hybrid filter-wrapper selection method by optimizing a parameterized ranking aggregator.

[1]See the Ablation Study in the Supplement, where their approach, "Variant (1)", is outperformed by all our methods on every dataset and learning algorithm with no exception.

There has been a lot of research on feature selection and hyperparameter optimization. However, we found that there is no *general* algorithm or framework to perform model-agnostic simultaneous hyperparameter optimization and feature selection for predictive performance and model sparsity in a multi-objective fashion. We therefore choose to tackle this problem from two directions. First we adapt a standard evolutionary multi-objective optimization method, the NSGA-II, for the particular problem of feature selection and hyperparameter tuning. We contrast this with a classical hyperparameter optimization algorithm, based on BO, extended to perform feature selection.

Our main contributions are:

(1) We adapt the NSGA-II to the problem at hand by introducing specialized sampling and mutation operators that make use of filter ensembles to enhance optimization performance.
(2) We investigate how multi-objective Bayesian optimization (MOBO) can be used for this problem and propose an effective method that uses a filter ensemble for feature selection.
(3) By conducting a benchmark of these approaches on a variety of tasks for different machine learning algorithms, we provide a comparison between our approaches and show how these algorithms perform compared to suitable baselines.

3 PROBLEM STATEMENT

We consider the machine learning problem of a given feature space $\mathcal{X}$ of vectors with p components (“features”), an arbitrary outcome space $\mathcal{Y}$ (for example $\{-1, 1\}$ for a binary classification task), and a performance measure $L : \mathbb{R}^g \times \mathcal{Y} \to \mathbb{R}$ measuring the quality of predictions (in $\mathbb{R}^g$) given ground truth values. g is 1 for regression tasks, and equal to the number possible outcome classes for classification. Data samples $\mathcal{D} = \left\{\left(\mathbf{x}^{(1)}, y^{(1)}\right), \dots, \left(\mathbf{x}^{(n)}, y^{(n)}\right)\right\} \in (\mathcal{X} \times \mathcal{Y})^n$ are assumed to be n i.i.d. realizations of random variables (X, Y) which follow a joint distribution $\mathbb{P}_{XY}$. A dataset is thus characterized by n, the number of samples available, and p, the number of *features*.

Let $\mathcal{A}_{\boldsymbol{\lambda},\boldsymbol{s}}$ be a learning algorithm that takes the given dataset $\mathcal{D}$ and constructs a model $f : \mathcal{X} \to \mathbb{R}^g$. The learning algorithm is parameterized by the *feature configuration vector* $\boldsymbol{s} \in \{0, 1\}^p$, where $s_j = 1$ denotes that feature j is included in the model. The sparseness of the resulting model is thus determined by the *Hamming weight* of $\boldsymbol{s}$, i.e. the number of components of $\boldsymbol{s}$ that are 1. *Hyperparameters* of the learning algorithm are summarized in a vector $\boldsymbol{\lambda} \in \Lambda$. Λ is a (possibly mixed) bounded space and may contain numeric, integer, and categorical values[2].

In general we are trying to construct a model $f = \mathcal{A}_{\boldsymbol{\lambda},\boldsymbol{s}}(\mathcal{D})$ which minimizes the *generalization error* $\mathrm{GE}[f] = \mathbb{E}_{\mathbb{P}_{XY}}[L(f(\mathbf{x}), y)]$. However, the generalization error can only be estimated using in-sample data $\widehat{\mathrm{GE}}[\mathcal{A}_{\boldsymbol{\lambda},\boldsymbol{s}}, \mathcal{D}]$ through a resampling technique such as cross-validation.

This setup gives rise to the bi-objective hyperparameter optimization and feature selection problem:

$$\min_{\boldsymbol{\lambda} \in \Lambda, \boldsymbol{s} \in \{0,1\}^p} \left(\widehat{\mathrm{GE}}\left[\mathcal{A}_{\boldsymbol{\lambda},\boldsymbol{s}}, \mathcal{D}\right], \sum_{i=1}^{p} c_i s_i,\right).$$

[2]Hierarchical dependencies of hyperparameters, e.g. kernel hyperparameters that appear only if a specific kernel is chosen, are a common extension but not considered in this work.

The setup regards estimated generalization error as one objective and the cost of features considered as another.

It is possible, and a trivial extension of our method, to consider arbitrary costs c_i for each feature i. However, in our benchmarks we limit ourselves to equal costs $c_i = 1/p$. The resulting measure corresponds to the *fraction of selected features*, ffrac $= \frac{1}{p}\sum_i s_i$, ranging from 0 to 1.

4 MULTI-OBJECTIVE HYPERPARAMETER TUNING AND FEATURE SELECTION: TWO APPROACHES

Our two methods are based on two popular multi-objective optimization methods, adapted for the particular task of hyperparameter tuning and feature selection: A model-based approach, and an approach based on an evolutionary algorithm. Both approaches make use of feature filters to accelerate the search.

4.1 Feature Filters and Filter Ensembles

The number of possible feature configurations $\boldsymbol{s}$ is exponential in p, so a large number of performance evaluations would be necessary to explore the performance space. Therefore, both our optimization approaches make use of *feature filters*. These are methods that heuristically score individual features according to their apparent relevance for the outcome variable. Because there are various methods to estimate this relevance, we consider a collection of M feature filters, which generate scores $F^m(\mathcal{D})_j$, $m = 1, \dots, M$, $j = 1, \dots, p$ for each feature j of a training dataset $\mathcal{D}$. Different methods may generate scores on different scales, but we rank-transform and scale these scores to values ranging in equi-distant steps from least (value of 0) to most (value of 1) relevant for the outcome variable.

We propose combining multiple filter methods into *filter ensembles* similarly to Dittman et al. [12], but extending their method by using *weighted* average rank aggregation. The ensemble filter score EF_j for feature j is calculated as the weighted average according to weight vector $\boldsymbol{w} \in [0, 1]^M$, $\sum_i w_i = 1$:

$$\mathrm{EF}_j(\boldsymbol{w}) = \sum_{m=1}^{M} w_m F^m(\mathcal{D})_j. \quad (1)$$

The weighting parameter $\boldsymbol{w}$ can be optimized, extending the filter into a hybrid filter-wrapper approach.

4.2 Bayesian Optimization Approach

Bayesian optimization (BO), also often referred to as *sequential model-based optimization*, has been successfully used for machine learning hyperparameter optimization in many applications [33]. The principle of BO is based on two steps, which are performed in turn. First, a so-called *surrogate model* is fitted to model the relationship between decision variables (e.g. hyperparameter values) and the objective value (e.g. estimated generalization performance). This model generates cheap approximations of the (generally expensive to evaluate) objective function values. In a second step, an *infill criterion* is used to find promising decision values to be evaluated on this expensive function. The infill criterion has to face a trade-off between “exploitation”—evaluating points for which the surrogate

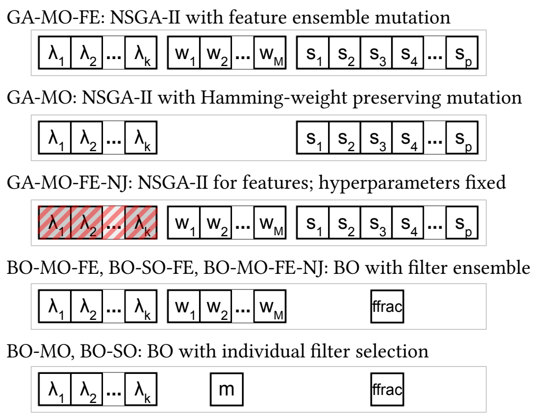

GA-MO-FE: NSGA-II with feature ensemble mutation
λ1 λ2 ... λk w1 w2 ... wM s1 s2 s3 s4 ... sp
GA-MO: NSGA-II with Hamming-weight preserving mutation
λ1 λ2 ... λk s1 s2 s3 s4 ... sp
GA-MO-FE-NJ: NSGA-II for features; hyperparameters fixed
λ1 λ2 ... λk w1 w2 ... wM s1 s2 s3 s4 ... sp
BO-MO-FE, BO-SO-FE, BO-MO-FE-NJ: BO with filter ensemble
λ1 λ2 ... λk w1 w2 ... wM ffrac
BO-MO, BO-SO: BO with individual filter selection
λ1 λ2 ... λk m ffrac

Figure 1: Representation of individuals in different variants of our proposed optimization methods as described in Section 5.3; λ, w, and s are vectors as described in Section 4.

predicts good performance—and "exploration"—evaluating points where predictive uncertainty is high.

There are different ways of adapting BO to perform multi-objective Bayesian optimization ("MOBO") [21], which has been applied successfully to hyperparameter tuning [19]. We chose the *Parego* method [27]. Parego scalarizes the different objectives through the Chebyshev norm using a random weight vector, which is sampled again for every point being proposed. Parego has the advantage over many other MOBO methods of only requiring a single surrogate model fit for a proposed point [19], but other MOBO approaches can trivially be used as a substitute within our method.

The number of possible feature configurations is exponential in p, so a large number of evaluations would be necessary to fit an accurate surrogate model on performance values. We therefore investigate the use of prior knowledge from *feature filter* scores to reduce the dimensionality of the search space. Instead of searching over the discrete space of possible feature configuration vectors $\mathbf{s} \in \{0, 1\}^p$, we restrict the search space to the *fraction of selected features* ffrac $\in [0, 1]$. The explicit feature configuration $\mathbf{s}$ results from selecting the ffrac most highly ranked features according to an individual filter (4.2.1) or according to the filter ensemble (4.2.2). This re-parametrization reduces the set of possible feature configurations being considered, and it may miss features that work well in combination but get a poor filter score because they perform poorly individually. Ultimate performance therefore depends on the quality of the filters being used, and we investigate methods that use combinations of filters to lessen the dependency on individual filter performance and therefore boost overall performance.

Two possible methods for simultaneous use of multiple feature filter methods are considered (see also Figure 1):

4.2.1 Individual filter selection. In this method, we introduce a discrete *filter selection hyperparameter m*, as well as a *feature fraction hyperparameter* ffrac $\in [0, 1]$. For each model evaluation, only the most relevant $\lceil p \cdot \text{ffrac} \rceil$ features, according to filter with index m, are included in the model. This approach is similar to the one taken in auto-sklearn [14], although there the feature selection problem was not considered as a multi-objective problem.

4.2.2 Filter ensemble selection. This method uses the filter ensemble as shown in Equation 1 and introduces the vector $\boldsymbol{w}$, as well as the aforementioned feature fraction ffrac, as hyperparameters. The most relevant $\lceil p \cdot \text{ffrac} \rceil$ features, according to $\text{EF}_j(\boldsymbol{w})$, are included in the model.

4.3 Evolutionary Approach

The *Nondominated Sorting Genetic Algorithm II* (NSGA-II) [10] is an evolutionary multi-objective algorithm that uses nondominated-sorting to preferably select individuals close to the Pareto-front of the problem. It iterates through generations of each a *reproduction*, a *crossover*, a *mutation*, and a *survival* step that generate the population of the next generation.

GAs often represent individuals as vectors of binary, discrete, or continuous values, depending on the optimization problem. Because hyperparameters generally have various types, we use the Cartesian products of operators that operate in different ways on the various types, following Li et al. [29]. This means e.g. that numeric hyperparameters undergo Gaussian mutation, while categorical hyperparameters undergo uniform mutation etc. Table 4 in the supplement summarizes the chosen recombination and mutation operators for the respective hyperparameter types. For the hyperparameter mutations, we use self-adapting step sizes and mutation probabilities as suggested by Li et al. [29].

The feature configuration vector parameter $\mathbf{s}$ plays a special role, because $\mathbf{s}$ maps to the objective of the fraction of selected features, which is being optimized, in a straightforward manner. We therefore consider the initialization and mutation of $\mathbf{s}$ in detail.

4.3.1 Geometric initialization. A naive approach for feature configuration initialization would be Bernoulli-sampling of each feature selection bit s_i individually, possibly biased towards a low expected number of selected features to favor relatively sparse solutions [1]. However, this gives rise to a binomial distribution of the number of selected features $\sum_i s_i$ with standard deviation $\sim O\left(\sqrt{p}\right)$. For even moderately large values of p, this fails to cover the objective space evenly along the dimension of the selected feature fraction. Figure 6 in the supplement illustrates this. We propose to sample values of s such that the sum of selected features covers the whole feature fraction objective. Therefore, we elect to use a truncated geometric distribution of number of included features to encode our preference for sparse models. This is achieved by sampling the desired number of included features S as a truncated geometrically distributed random integer between 0 and p, and then uniformly sampling from all vectors s that satisfy $\sum_i s_i = S$.

The method introduces the success probability of the geometric distribution as a configuration parameter. It can be set by the user to encode a relative preference for sparsity. We chose to use an empirically determined value by fitting decision trees on 100 random subsets of 90% of the data set and determining the average number of distinct split variables.

4.3.2 Filter-ensemble based initialization. It is possible to enhance the geometric initialization by including prior knowledge gained from feature filter methods. The goal is to select the features that have high filter scores with larger probability than the ones with lower ranking, while still having an approximately geometric distribution over the number of total features selected. We make the initial distribution of bit j dependent on the filter ensemble value as described in equation 1, with $\boldsymbol{w}$ uniformly randomly sampled from the simplex $\boldsymbol{w} \in [0,1]^M$, $\sum_{m=1}^{M} w_m = 1$. For each individual to be initialized, we first sample S as in 4.3.1. Each bit j is then sampled from a Bernoulli-distribution with parameter:

$$\pi_{\mathrm{B}}(\boldsymbol{w}, S) = \frac{\mathrm{EF}_j(\boldsymbol{w})\,(S+1)}{\mathrm{EF}_j(\boldsymbol{w})S + \left(1 - \mathrm{EF}_j(\boldsymbol{w})\right)(p-S) + 1}. \quad (2)$$

4.3.3 Hamming-weight preserving mutation. Performing random bit-flip mutation on the feature selection vector entails similar problems to Bernoulli-initialization: a bit-flip with probability $\pi_{\mathrm{Mut_{bin}}}$ is equivalent to erasing a bit with probability $2\pi_{\mathrm{Mut_{bin}}}$ and sampling it anew from a $\frac{1}{2}$-Bernoulli distribution. This biases the mutation result towards the $\frac{1}{2}$ point of the feature fraction objective. Instead, we choose to approximately preserve the Hamming weight. This is done by sampling erased bits from a Bernoulli-distribution with parameter $\pi_{\mathrm{B}} = (S+1)/(p+2)$, where $S = \sum_i s_i$ is the Hamming weight of the original vector.

4.3.4 Filter-ensemble based mutation. Just as for initialization, mutation can be made dependent on feature filter ensemble ranks to preferentially include more relevant features. As in the Hamming-weight preserving mutation, each bit is erased with a probability of $2\pi_{\mathrm{Mut_{bin}}}$ and then drawn from a Bernoulli-distribution. The parameter is as in equation 2 for filter-based initialization, with S the Hamming weight of the original vector, and $\boldsymbol{w}$ a weighting vector. This weighting vector itself becomes part of the search space to achieve self-adaption, as described in Li et al. [29], which results in $\mathbf{w}$ being optimized during the NSGA-II run. This way, the Hamming-weight is (approximately) preserved.

4.4 Implementation and Reproducibility

All proposed methods are implemented and publicly accessible through the `mosmafs` R package published on CRAN[3], using the `mlr` [2] machine learning framework as backend. For full reproducibility of the results we publish the code used to perform the benchmark experiments on Github[4].

5 EXPERIMENTS

We conduct experiments to answer the following research questions empirically:

(1) *Evolutionary vs. Bayesian Optimization*: How do the proposed methods—the evolutionary and the Bayesian optimization approach—perform relative to each other?
(2) *Effect of Filter Ensembles*: Do the methods benefit from using *filter ensembles*?
(3) *Multi-Objective vs. Single-Objective*: Does multi-objective optimization find much sparser solutions without a major loss in predictive performance compared to single-objective optimization?
(4) *Simultaneous Hyperparameter Tuning and Feature Selection*: Is it beneficial to perform hyperparameter optimization and feature selection *simultaneously* compared to performing the tasks sequentially?

[3]https://CRAN.R-project.org/package=mosmafs
[4]https://github.com/compstat-lmu/mosmafs/tree/master

Table 1: Description of the datasets being used. n denotes the number of observations and p the total number of features. The class ratio gives the proportional size of the smaller outcome class. The dataset id (did) is the unique identifier for the dataset on the OpenML platform [34].

name	n	p	class ratio	n/p	did
wdbc	569	30	0.37	18.97	1510
ionosphere	351	33	0.36	10.64	59
sonar	208	60	0.47	3.47	40
hill-valley	1212	100	0.50	12.12	1479
tecator	240	124	0.43	1.94	851
semeion	319	256	0.50	1.25	41973
madeline	3140	259	0.50	12.12	41144
lsvt	126	307	0.33	0.41	1484
madelon	2600	500	0.50	5.20	1485
isolet	600	617	0.50	0.97	41966
cnae-9	240	282	0.50	0.85	41967
arcene	200	9961	0.44	0.02	1458
AP_Breast_Colon	630	10935	0.45	0.06	1145
AP_Colon_Kidney	546	10935	0.48	0.05	1137

5.1 Benchmark Datasets

Because our implementation is based off the `mlr` framework, it can work with a wide variety of regression and classification tasks, even in conditions of missing values or class imbalance. However, we conduct our experiments on a set of well-behaved problems to limit the scope of the experiments necessary to show effectiveness.

We consider real-world binary classification tasks that are publicly accessible through the OpenML platform [34] (see Table 1). To eliminate algorithmic factors that might influence the result (like class imbalance correction, feature encoding, or handling of missing values), we included datasets fulfilling the following criteria: (roughly) balanced outcome classes, a purely numeric feature space, and no missing values. Further, datasets have been chosen to represent a large variety in terms of dimensionality ($30 \leq p \leq 10937$)[5] and in terms of instances per dimension ($0.02 \leq n/p \leq 18.97$).

5.2 Learning Algorithms and their Hyperparameters

We consider three different classifiers that are tuned by the optimization algorithms proposed in Section 4: The support vector machine classifier (SVM) [6] with Gaussian kernel, extreme gradient boosting (xgboost) [5] and the kernelized k-nearest-neighbor classifier (kknn) [39]. We decided to consider these learners because they represent three very distinct learning paradigms, and because they are generally regarded responsive to hyperparameter tuning. The hyperparameter spaces that are being tuned over are presented

[5]To limit the computational resources necessary for our experiments we did not investigate even larger datasets.

Table 2: Hyperparameter spaces over which tuning was performed for the three learning algorithms *xgboost*, *support vector machine* and *kernelized k-Nearest-Neighors* respectively. Ranges marked with * were tuned on a logarithmic scale. For the large datasets madeline, madelon, arcene, AP_Breast_Colon and AP_Colon_Kidney, the xgboost *nrounds* parameter was set fixed to 2000 and early stopping after 10 rounds was enabled.

SVM		xgboost	
kernel	rbfdot	nrounds	$\{1, 2, \ldots, 2000\}$
sigma	$[2^{-10}, 2^{10}]$ *	eta	$[0.01, 0.2]$
C	$[2^{-10}, 2^{10}]$ *	gamma	$[2^{-7}, 2^{6}]$ *
		max_depth	$\{3, \ldots, 20\}$
kknn		colsample_bytree	$[0.5, 1]$
k	$\{1, 2, \ldots, 50\}$	colsample_bylevel	$[0.5, 1]$
distance	$[1, 100]$	lambda	$[2^{-10}, 2^{10}]$ *
kernel	{rectangular, optimal, triangular, biweight}	alpha	$[2^{-10}, 2^{10}]$ *
		subsample	$[0.5, 1]$

in Table 2. They also react differently to high dimensionality: kknn is vulnerable to the "curse of dimensionality", the SVM is a regularized modeling algorithm, and the xgboost algorithm does implicit feature selection because it is a tree-based learner.

5.3 Algorithms

We perform hyperparameter optimization and feature selection for the learners described in Section 5.2. The datasets used are presented in Section 5.1. We use different configurations of our methods for comparisons described in the following. See also Figure 1 for a schematic representation of individual configuration vectors used in each method.

To answer research question 1, comparing the evolutionary and Bayesian optimization methods, we consider:

GA-MO-FE The NSGA2 with filter ensemble based initialization (as in 4.3.2) *and* mutation (as in 4.3.4).

BO-MO-FE Multi-objective Bayesian optimization with filter ensemble selection as described in 4.2.2.

To study research question 2, i.e. the contribution of filter ensembles to the methods, we compare both methods to these algorithms without filter ensembles:

GA-MO The NSGA2 with filter ensemble based initialization (as in 4.3.2) but *without* filter-based mutation, instead only using Hamming-weight preserving mutation as in 4.3.3.

BO-MO Multi-objective Bayesian optimization with individual filter selection, as described in 4.2.1.

We illuminate research question 3 by considering feature selection and hyperparameter optimization as a single-objective task, run with the following methods:

BO-SO Single-objective Bayesian optimization with individual filter selection (see 4.2.1)—this is a method similar to auto-sklearn's approach [14] which we consider to be a state-of-the-art approach.

BO-SO-FE Single-objective Bayesian optimization with filter ensemble selection (see 4.2.2), which is our strongest single objective baseline.

To address research question 4, we consider variants of our algorithms that perform feature selection and hyperparameter optimization in separate steps. We construct two algorithms to approximate the most straightforward simplifications to non-joint optimization. Note that the GA-approach has a focus on feature selection, while the BO-approach is more directed at hyperparameter tuning.

GA-MO-FE-NJ We initially optimize hyperparameters for the learning algorithms on each dataset by running standard single objective BO on the *full* datasets without performing feature selection, through 500 model evaluations. The resulting hyperparameter values are then fixed, while our full NSGA2-variant (using both filter ensemble initialization and mutation) only performs multi-objective feature selection.

BO-MO-FE-NJ To optimize hyperparameters and filter weights we use the BO-SO-FE method, and then evaluate the final model with the fixed hyperparameters. We iterate over equally spaced ffrac values and evaluate the model with all filters. This constructs a set of models with different trade-offs between sparsity and model performance, from which we construct a Pareto-set.

The methods have access to $M = 5$ filter methods[6]: Information Gain, Random Forest Feature Importance [23], Joint Mutual Information (JMI) [38], Minimal Conditional Mutual Information Maximization (CMIM) [15], Area Under the Curve (AUC).

All NSGA2 variants use $\mu = 80$ as population and $\lambda = 15$ as offspring size which Khan and Baig [26] found to perform well in a feature selection setting. We choose an overall per-individual mutation probability of 0.3 and an overall per-pair-of-individuals crossover probability of 0.7.

Our Bayesian optimization methods use a random forest as surrogate to model the mixed discrete and continuous hyperparameter space similar to the state-of-the-art hyperparameter optimization toolbox SMAC [22]. The infill criterion used is LCB [7]. In each iteration, a batch of 15 configurations is proposed in parallel as described in Horn et al. [20].

5.4 Evaluation

We measure the performance of resulting models by their mean misclassification error (mmce $= \frac{1}{n} \sum_i \mathbb{I}_{y^{(i)} \neq \hat{y}^{(i)}}$) on a validation set with ground truth values $y^{(i)}$ and model predictions $\hat{y}^{(i)}$.

To get an estimate of the optimization performance that is unbiased by potential overtuning, we performed *nested resampling*: During the whole optimization run, each optimization algorithm is only allowed to assess model performance $\text{mmce}_{\text{optim}}$ through (*inner*) cross-validation (CV) on a optimization set $\mathcal{D}_{\text{optim}}$. The final performance of the algorithm is reported as the performance of

[6]The methods were chosen by running many filters on a range of datasets and performing hierarchical clustering on the differences of their feature rankings; see the Supplement for more details.

the solution candidates trained on $\mathcal{D}_{\text{optim}}$ and evaluated on a test set $\mathcal{D}_{\text{test}}$. This procedure is repeated 10 times on *outer* CV folds $\left(\mathcal{D}^{(k)}_{\text{optim}}, \mathcal{D}^{(k)}_{\text{test}}\right), k = 1, ..., 10$.

As a proper multi-objective performance measure we consider the *dominated hypervolume* [40] (domHV) with reference point $W = (1, 1)$, which corresponds to the worst possible values w.r.t. the two objectives. To prevent overtuning effects from skewing our results, we proceed as follows:

(1) We let each optimization algorithm report its Pareto set, taking into account only $\mathcal{D}_{\text{optim}}$. This corresponds to returning non-dominated individuals w.r.t. ($\text{mmce}_{\text{optim}}$, ffrac). GA-based methods take only candidates from their current generation, while BO-based methods get the Pareto set of all candidates seen so far.
(2) We calculate the *generalization* dominated hypervolume $\text{domHV}_{\text{gen}}$, i.e. the hypervolume that is dominated by this Pareto set of candidate configurations w.r.t. the ($\text{mmce}_{\text{test}}$, ffrac) evaluated on $\mathcal{D}_{\text{test}}$.[7]

For each of the experiments, we allow 2000 model evaluations, with one evaluation corresponding to computing a single inner 10-fold CV on $\mathcal{D}_{\text{optim}}$. This corresponds to approximately 130 iterations for both the model-based and the evolutionary approach. We test performance differences for significance using the critical difference test of Demšar [11], which is a non-parametric test for comparison of different methods on different datasets.

6 RESULTS

6.1 Evolutionary vs. Bayesian Optimization, Effect of Filter Ensembles

The global rank analysis as well as a critical difference test [11] presented in Figure 2 show that our most advanced BO-based method, BO-MO-FE, significantly outperforms all GA methods. The optimization trace seems to indicate that including the feature ensemble also confers an advantage, possibly because linear combinations of filters can perform better than any individual filter. This difference is not statistically significant after 2000 evaluations, however.

Figure 5 shows an additional aspect to take into consideration: overall runtime. BO-MO-FE usually has a slight performance advantage over GA-MO-FE in absolute terms, but comes with computational overhead that may have to be considered if individual model performance evaluations are relatively fast. The overhead comes mostly from surrogate model fitting and infill optimization, with runtime dependent on the number of hyperparameters, but independent of dataset size. The overhead becomes less relevant when optimizing large datasets or slow models.

6.2 Multi-Objective vs. Single-Objective

The goal of our multi-objective methods is not to optimize predictive performance by itself, and instead to explore the optimal trade-offs between performance and sparseness. It is still interesting to look at the best performing model configurations being found by the multi-objective methods, and to compare them to the best models found by single-objective methods, as done in Figure 4. Here, again, the BO methods outperform GA methods. Except for the *kknn* learning algorithm, the MO-method is on par with the SO methods.

[7]Note that $\text{domHV}_{\text{gen}}$ is not the same as the dominated hypervolume of a population on the test set. Instead, only the individuals that are non-dominated according to $\mathcal{D}_{\text{optim}}$ are used to calculate their dominated hypervolume on $\mathcal{D}_{\text{test}}$.

Table 3: $\text{domHV}_{\text{gen}}$ after 2000 evaluations of simultaneous hyperparameter tuning / feature selection methods compared to corresponding non-simultaneous methods. Best results for GA and BO in bold. Values averaged over datasets, see supplement for results by dataset.

Learner	BO-MO-FE	BO-MO-FE-NJ	GA-MO-FE	GA-MO-FE-NJ
kknn	**0.9217**	0.8670	0.9108	**0.9134**
SVM	**0.9331**	0.8719	**0.9241**	0.8892
xgboost	**0.9164**	0.8940	**0.9165**	0.9121

The BO-SO method employs a feature filtering step, and the models it chooses will often be sparse to some degree, for regularization. Figure 3 compares the BO-SO method to the multi-objective methods in terms of both sparseness and predictive performance: For each learning algorithm, dataset, and CV fold of BO-SO, the best performing models from BO-MO-FE and GA-MO-FE were chosen that are *at least* as sparse. Note that the *length* of arrows in this figure depends mainly on the size of the population returned by the methods and is not directly of interest. Instead the *slope* should be considered, with steeper arrows pointing to the right indicating a cheaper trade for sparseness vs. performance (and arrows pointing to the left indicating results outperforming BO-SO). The plot shows that both MO methods often found much sparser models than the SO baseline while giving up very little in predictive performance.

6.3 Simultaneous Hyperparameter Tuning and Feature Selection

Table 3 shows the $\text{domHV}_{\text{gen}}$ of our joint optimization methods, compared to their non-joint correspondents. In most cases the joint optimization confers a considerable performance advantage, especially compared to the BO-SO-FE-NJ method. We assume that the advantage of joint over non-joint methods depends on the interaction strength of hyperparameter performance and sparseness.

7 CONCLUSION

Our results show that both evolutionary approaches and model-based approaches can efficiently perform model-agnostic multi-objective optimization to simultaneously tune hyperparameters and select features. We have also shown that performing these tasks simultaneously has an advantage over running them separately.

Our experiments suggest that using multi-objective model-based optimization to tune feature inclusion based on single filter works well, although a parameterized filter ensemble performs better.

Our other proposed method based on an NSGA-II enhanced with specialized initialization and mutation sampling seems to perform almost as well, although it does not reach the level of the best Bayesian optimization based approach. The advantage of the NSGA-II, however, is that it does not introduce as much computational overhead, which can make up a considerable part of overall runtime if model performance evaluations themselves are cheap.

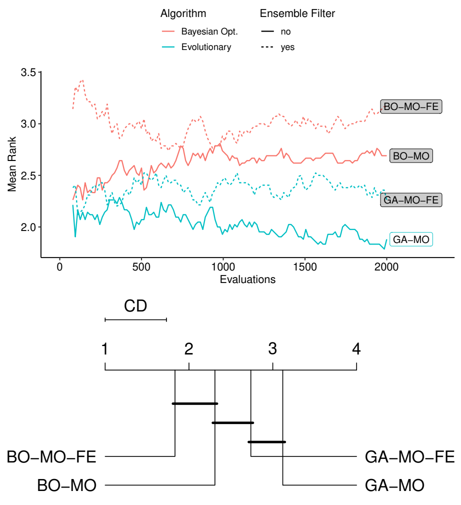


Figure 2: Comparison of different methods. Top: the results of a global rank analysis based on $\text{domHV}_{\text{gen}}$ to compare the proposed methods BO-MO(-FE) and GA-MO(-FE), i.e. both the BO and GA with and without filter ensemble. Ranks are computed per dataset and algorithm (ties are ranked by their average rank) and then averaged. Higher values are better. Bottom: Non-parametric critical difference test [11] performed after the full budget of 2000 evaluations at significance level $\alpha = 0.05$, based on $\text{domHV}_{\text{gen}}$ rank as above. The BO-MO-FE method statistically significantly outperforms all GA methods, although the absolute difference is small (see Figure 5).

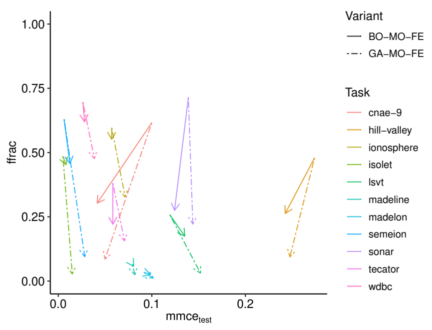


Figure 3: Comparison of multi-objective and single-objective methods applied to the SVM learning algorithm: Performance $\text{mmce}_{\text{test}}$ and the fraction of included features ffrac found after 2000 evaluations by baseline BO-SO (tail end of arrows), BO-MO-FE (head of solid arrows) and GA-MO-FE (head of dashed arrows). Each dataset (Table 1) is shown, values are averaged over 10 outer CV runs. Choice of individuals for MO methods described in Section 6.2.

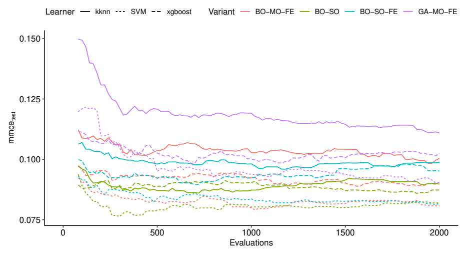


Figure 4: Comparison of multi-objective and single-objective methods: $\text{mmce}_{\text{test}}$ of the best $\text{mmce}_{\text{optim}}$ individual found until a particular evaluation, averaged over 10 outer CV folds and over all datasets. Lower values are better. Note that $\text{mmce}_{\text{test}}$ is not monotonically improving because individuals with better $\text{mmce}_{\text{optim}}$ may be found that perform worse on the test set than previous individuals.

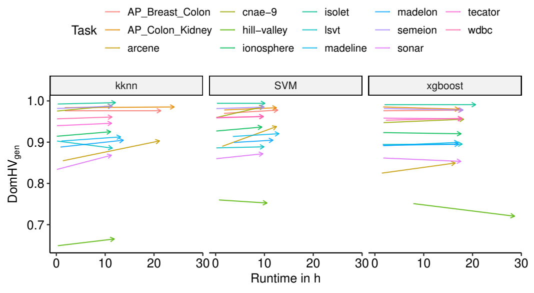


Figure 5: Performance ($\text{domHV}_{\text{gen}}$) and runtime of GA-MO-FE (tail end of arrow) and BO-MO-FE (head of arrow) on each dataset (Table 1) and learning algorithm (Table 2), averaged over 10 outer CV runs. BO-MO-FE has moderately improved performance over GA-MO-FE, but at a cost in runtime.

Our final recommendation is therefore to use the Bayesian optimization approach in combination with parameterized Filter ensembles if model evaluations are expensive, if computational resources are cheap, and if it is important to get configurations that perform very close to optimal, given their sparseness. The NSGA-II approach is suitable if model evaluations are cheap and if marginal degradation of performance are acceptable.

ACKNOWLEDGMENTS

This work has been partially 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.

This work was supported by the Bavarian Ministry for Economic Affairs, Infrastructure, Transport and Technology through the Center for Analytics-Data-Applications (ADA-Center) within the framework of "BAYERN DIGITAL II".

REFERENCES

[1] Bernd Bischl, Igor Vatolkin, and Mike Preuss. Selecting small audio feature sets in music classification by means of asymmetric mutation. In *Proceedings of the 11th International Conference on Parallel Problem Solving from Nature*, volume 6238, pages 314–323, 2010. ISBN 3642158439.

[2] Bernd Bischl, Michel Lang, Jakob Bossek, Leonard Judt, Jakob Richter, Tobias Kuehn, and Erich Studerus. mlr: Machine Learning in R. *Journal of Machine Learning Research*, 17(170):1–5, 2014.

[3] Amparo Bolón-Canedo, Verónica; Alonso-Betanzos. Ensembles for feature selection: A review and future trends. *Information Fusion*, 11 2018. doi: 10.1016/j.inffus.2018.11.008.

[4] Amal Bouraoui, Salma Jamoussi, and Yassine BenAyed. A multi-objective genetic algorithm for simultaneous model and feature selection for support vector machines. *Artificial Intelligence Review*, 50(2):261–281, 2018. ISSN 15737462.

[5] Tianqi Chen and Carlos Guestrin. *XGBoost: A Scalable Tree Boosting System.* ACM, 2016. ISBN 9781450342322.

[6] Corinna Cortes and Vladimir Vapnik. Support-Vector Networks. *Machine Learning*, 20(3):273–297, 1995. ISSN 15730565.

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

[8] M. Dash and H Liu. Feature selection for classification. *Intelligent Data Analysis*, 1(3):131–156, 1997. ISSN 1088467X.

[9] Kalyanmoy Deb and R B Agrawal. Simulated binary crossover for continuous search space. *Complex Systems*, 9(2):115–148, 1994. ISSN 08912513.

[10] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002. ISSN 1089778X.

[11] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.*, 7:1–30, December 2006. ISSN 1532-4435.

[12] David J. Dittman, Taghi M. Khoshgoftaar, Randall Wald, and Amri Napolitano. Comparing two new gene selection ensemble approaches with the commonly-used approach. In *2012 11th International Conference on Machine Learning and Applications*. IEEE, December 2012. doi: 10.1109/icmla.2012.175. URL https://doi.org/10.1109/icmla.2012.175.

[13] C. Emmanouilidis, A. Hunter, and J. Macintyre. A multiobjective evolutionary setting for feature selection and a commonality-based crossover operator. In *Proceedings of the 2000 Congress on Evolutionary Computation. CEC00 (Cat. No.00TH8512)*, volume 1, pages 309–316. IEEE, 2000. ISBN 0-7803-6375-2.

[14] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and Robust Automated Machine Learning, 2015. URL http://papers.nips.cc/paper/5872-efficient-and-robust-automated-machine-learning.

[15] Francois Fleuret. Fast Binary Feature Selection with Conditional Mutual Information. *Journal of Machine Learning Research*, 5:1531–1555, 2004.

[16] Luca Franceschi, Paolo Frasconi, Saverio Salzo, Riccardo Grazzi, and Massimiliano Pontil. Bilevel programming for hyperparameter optimization and meta-learning. In *Proceedings of the 35th International Conference on Machine Learning (ICML 2018*, Proceedings of Machine Learning Research. PMLR, 2018.

[17] Isabelle Guyon and Andre Elisseeff. An Introduction to Variable and Feature Selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003. ISSN 00032670.

[18] Tarek M. Hamdani, Jin-Myung Won, Adel M. Alimi, and Fakhri Karray. Multi-objective Feature Selection with NSGA II. In *Adaptive and Natural Computing Algorithms*, pages 240–247. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[19] 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. IEEE, 2016.

[20] Daniel Horn, Tobias Wagner, Dirk Biermann, Claus Weihs, and Bernd Bischl. Model-based multi-objective optimization: Taxonomy, multi-point proposal, toolbox and benchmark. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 9018, pages 64–78, 2015. ISBN 9783319159331.

[21] Daniel Horn, Tobias Wagner, Dirk Biermann, Claus Weihs, and Bernd Bischl. Model-based multi-objective optimization: Taxonomy, multi-point proposal, toolbox and benchmark. In *Lecture Notes in Computer Science*, pages 64–78. Springer International Publishing, 2015. doi: 10.1007/978-3-319-15934-8_5. URL https://doi.org/10.1007/978-3-319-15934-8_5.

[22] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential Model-Based Optimization for General Algorithm Configuration. In *International conference on learning and intelligent optimization*. Springer, 2011. URL http://www.cs.ubc.ca/labs/beta/Projects/SMAC/papers/11-LION5-SMAC.pdf.

[23] Hemant Ishwaran, Udaya B Kogalur, Eiran Z Gorodeski, Andy J Minn, and Michael S Lauer. High-dimensional variable selection for survival data. *Journal of the American Statistical Association*, 105(489):205–217, 2010. ISSN 01621459.

[24] Donald R Jones, Matthias Schonlau, and William J Welch. Efficient Global Optimization of Expensive Black-Box Functions. *Journal of Global Optimization*, 13:455–492, 1998. URL http://www.ressources-actuarielles.net/EXT/ISFA/1226.nsf/0/f84f7ac703bf5862c12576d8002f5259/{\protect\T1\textdollar}FILE/Jones98.pdf.

[25] T. Meyarivan K. Deb, A. Pratap, S. Agarwal. A fast and elitist multi-objective genetic algorithm: NSGA-II, IEEE Transactions on Evolutionary Computation. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[26] A. Khan and A. R. Baig. Multi-objective feature subset selection using non-dominated sorting genetic algorithm. *Journal of Applied Research and Technology*, 13(1):145–159, 2015. ISSN 16656423.

[27] 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, feb 2006. ISSN 1089-778X. doi: 10.1109/TEVC.2005.851274. URL http://ieeexplore.ieee.org/document/1583627/.

[28] Ron Kohavi and George H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324, 2002. ISSN 00043702.

[29] 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):29–64, 2013. ISSN 10636560.

[30] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

[31] Randal S. Olson and Jason H. Moore. TPOT: A Tree-Based Pipeline Optimization Tool for Automating Machine Learning. pages 151–160. Springer, Cham, 2019. doi: 10.1007/978-3-030-05318-5_8. URL http://link.springer.com/10.1007/978-3-030-05318-5{_}8.

[32] Florian Pfisterer, Stefan Coors, Janek Thomas, and Bernd Bischl. Multi-objective automatic machine learning with autoxgboostmc. *arXiv preprint arXiv:1908.10796*, 2019.

[33] 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, 2012.

[34] 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. ISSN 19310145. URL http://dl.acm.org/citation.cfm?doid=2641190.2641198.

[35] Kashif Waqas, Rauf Baig, and Shahid Ali. Feature subset selection using multi-objective genetic algorithms. In *IEEE 13th International Multitopic Conference*, pages 1–6, 2009. ISBN 9781424448722.

[36] Bing Xue, Mengjie Zhang, and Will N Browne. Particle swarm optimization for feature selection in classification: A multi-objective approach. *IEEE Transactions on Cybernetics*, 43(6):1656–1671, 2013. ISSN 21682267.

[37] Bing Xue, Mengjie Zhang, Will N. Browne, and Xin Yao. A Survey on Evolutionary Computation Approaches to Feature Selection. *IEEE Transactions on Evolutionary Computation*, 20(4):606–626, 2016. ISSN 1089778X.

[38] H Yang. Data visualization and feature selection: New algorithms for nongaussian data. *Advances in Neural Information Processing Systems*, pages 687—-693, 1999.

[39] Kai Yu, Liang Ji, and Xuegong Zhang. Kernel Nearest-Neighbor Algorithm. *Neural Processing Letters*, 15(2):147–156, 2002. ISSN 13704621. doi: 10.1023/A:1015244902967. URL http://link.springer.com/10.1023/A:1015244902967.

[40] Eckart Zitzler and Lothar Thiele. Multiobjective Optimization Using Evolutionary Algorithms — a Comparative Case Ctudy. In Agoston E. Eiben, Thomas Bäck, Marc Schoenauer, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature — PPSN V*, pages 292–301, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg. ISBN 978-3-540-49672-4.

5. Contributions to explainability of the learning algorithm

5.1. Explaining Hyperparameter Optimization via Partial Dependence Plots

The paper presents a method based on PDPs that allows for visualizing the effects of hyperparameters on model performance in the presence of a sampling bias in a more accurate way through a partitioning approach. It can be classified as methodological contribution to ML algorithm explainability via an interpretability method, see Section 3.2 – *Interpretability methods (B).*

Contributing article:

Moosbauer*, J., Herbinger*, J., Casalicchio, G., Lindauer, M., Bischl, B. (2021). Explaining Hyperparameter Optimization via Partial Dependence Plots. *Advances in Neural Information Processing Systems 34 (NeurIPS 2021), pp. 2280–2291.*

Author contributions:

Julia Moosbauer and Julia Herbinger developed the project idea with continuous support from Giuseppe Casalicchio. Julia Moosbauer has developed the idea of leveraging an uncertainty estimate to improve interpretability measures and derived the uncertainty estimate for PDPs. Julia Herbinger has formed the initial core idea for the partitioning method to identify subregions based on uncertainty estimates with support from Giuseppe Casalicchio. The algorithm was developed by Julia Moobauer based on an initial tree-splitting algorithm implemented by Giuseppe Casalicchio and has been substantially improved by Julia Herbinger. Julia Moosbauer and Julia Herbinger jointly designed and conducted the experiments. Evaluation metrics for the benchmark were defined by Julia Herbinger and improved by Julia Moosbauer. Julia Moosbauer has implemented the benchmark on synthetic functions, and Julia Herbinger has implemented the deep learning benchmark. The manuscript was drafted jointly by Julia Moosbauer and Julia Herbinger with overall equal contributions. All authors contributed to revisions of the paper. Giuseppe Casalicchio, Marius Lindauer, and Bernd Bischl gave valuable input throughout the project and suggested several notable modifications.

Supplementary material available at:

- Appendix: `https://papers.nips.cc/paper/2021/file/12ced2db6f0193dda91ba86224ea1cd8-Supplemental.pdf`

Explaining Hyperparameter Optimization via Partial Dependence Plots

Julia Moosbauer[*]**, Julia Herbinger**[*]**, Giuseppe Casalicchio, Marius Lindauer, Bernd Bischl**
Department of Statistics, Ludwig-Maximilians-University Munich, Munich, Germany
Institute of Information Processing, Leibniz University Hannover, Hannover, Germany
`{julia.moosbauer, julia.herbinger, giuseppe.casalicchio, bernd.bischl}@stat.uni-muenchen.de`
`lindauer@tnt.uni-hannover.de`

Abstract

Automated hyperparameter optimization (HPO) can support practitioners to obtain peak performance in machine learning models. However, there is often a lack of valuable insights into the effects of different hyperparameters on the final model performance. This lack of explainability makes it difficult to trust and understand the automated HPO process and its results. We suggest using interpretable machine learning (IML) to gain insights from the experimental data obtained during HPO with Bayesian optimization (BO). BO tends to focus on promising regions with potential high-performance configurations and thus induces a sampling bias. Hence, many IML techniques, such as the *partial dependence plot* (PDP), carry the risk of generating biased interpretations. By leveraging the posterior uncertainty of the BO surrogate model, we introduce a variant of the PDP with estimated confidence bands. We propose to partition the hyperparameter space to obtain more confident and reliable PDPs in relevant sub-regions. In an experimental study, we provide quantitative evidence for the increased quality of the PDPs within sub-regions.

1 Introduction

Most machine learning (ML) algorithms are highly configurable. Their hyperparameters must be chosen carefully, as their choice often impacts the model performance. Even for experts, it can be challenging to find well-performing hyperparameter configurations. Automated machine learning (AutoML) systems and methods for automated HPO have been shown to yield considerable efficiency compared to manual tuning by human experts [Snoek et al., 2012]. However, these approaches mainly return a well-performing configuration and leave users without insights into decisions of the optimization process. Questions about the importance of hyperparameters or their effects on the resulting performance often remain unanswered. Not all data scientists trust the outcome of an AutoML system due to the lack of transparency [Drozdal et al., 2020]. Consequently, they might not deploy an AutoML model, despite all performance gains. Providing insights into the search process may help increase trust and facilitate interactive and exploratory processes: A data scientist could monitor the AutoML process and make changes to it (e.g., restricting or expanding the search space) already *during* optimization to anticipate unintended results.

Transparency, trust, and understanding of the inner workings of an AutoML system can be increased by interpreting the internal surrogate model of an AutoML approach. For example, BO trains a surrogate model to approximate the relationship between hyperparameter configurations and model performance. It is used to guide the optimization process towards the most promising regions of the hyperparameter space. Hence, surrogate models implicitly contain information about the influence of

[*]These authors contributed equally to this work.

35th Conference on Neural Information Processing Systems (NeurIPS 2021).

hyperparameters. If the interpretation of the surrogate matches with a data scientist's expectation, confidence in the correct functioning of the system may be increased. If these do not match, it provides an opportunity to look either for bugs in the code or for new theoretical insights.

We propose to analyze surrogate models with methods from IML to provide insights into the results of HPO. In the context of BO, typical choices for surrogate models are flexible, probabilistic black-box models, such as Gaussian processes (GP) or random forests. Interpreting the effect of single hyperparameters on the performance of the model to be tuned is analogous to interpreting the feature effect of the black-box surrogate model. We focus on the PDP [Friedman, 2001], which is a widely-used method[2] to visualize the average marginal effect of single features on a black-box model's prediction. When applied to surrogate models, they provide information on how a specific hyperparameter influences the estimated model performance. However, applying PDPs out of the box to surrogate models might lead to misleading conclusions. Efficient optimizers such as BO tend to focus on exploiting promising regions of the hyperparameter space while leaving other regions less explored. Therefore, a sampling bias in input space is introduced, which in turn can lead to a poor fit and biased interpretations in underexplored regions of the space.

Contributions: We study the problem of sampling bias in experimental data produced by AutoML systems and the resulting bias of the surrogate model and assess its implications on PDPs. We then derive an uncertainty measure for PDPs of probabilistic surrogate models. In addition, we propose a method that splits the hyperparameter space into interpretable sub-regions of varying uncertainty to obtain sub-regions with more reliable and confident PDP estimates. In the context of BO, we provide evidence for the usefulness of our proposed methods on a synthetic function and in an experimental study in which we optimize the architecture and hyperparameters of a deep neural network. Our Supplementary Material provides (A) more background related to uncertainty estimates, (B) notes on how our methods are applied to hierarchical hyperparameter spaces, (C) details on the experimental setup and more detailed results, (D) a link to the source code.

Reproducibility and Open Science: The implementation of the proposed methods as well as reproducible scripts for the experimental analysis are provided in a public git-repository[3].

2 Background and Related Work

Recent research has begun to question whether the evaluation of an AutoML system should be purely based on the generated models' predictive performance without considering interpretability [Hutter et al., 2014a, Pfisterer et al., 2019, Freitas, 2019, Xanthopoulos et al., 2020]. Interpreting AutoML systems can be categorized as (1) interpreting the resulting ML model on the underlying dataset, or (2) interpreting the HPO process itself. In this paper, we focus on the latter.

Let $c : \Lambda \to \mathbb{R}$ be a *black-box* cost function, mapping a hyperparameter configuration $\boldsymbol{\lambda} = (\lambda_1, ..., \lambda_d)$ to the model error[4] obtained by a learning algorithm with configuration $\boldsymbol{\lambda}$. The hyperparameter space may be mixed, containing categorical and continuous hyperparameters. The goal of HPO is to find $\boldsymbol{\lambda}^* \in \arg\min_{\boldsymbol{\lambda} \in \Lambda} c(\boldsymbol{\lambda})$. Throughout the paper, we assume that a surrogate model $\hat{c} : \Lambda \to \mathbb{R}$ is given as an approximation to c. If the surrogate is assumed to be a GP, $\hat{c}(\boldsymbol{\lambda})$ is a random variable following a Gaussian posterior distribution. In particular, for any finite indexed family of hyperparameter configurations $\left(\boldsymbol{\lambda}^{(1)}, ..., \boldsymbol{\lambda}^{(k)}\right) \in \Lambda^k$, the vector of estimated performance values is Gaussian with a posterior mean $\hat{\boldsymbol{m}} = \left(\hat{m}\left(\boldsymbol{\lambda}^{(i)}\right)\right)_{i=1,...,k}$ and covariance $\hat{\boldsymbol{K}} = \left(\hat{k}\left(\boldsymbol{\lambda}^{(i)}, \boldsymbol{\lambda}^{(j)}\right)\right)_{i,j=1,...,k}$.

Hyperparameter Importance. Understanding which hyperparameters influence model performance can provide valuable insights into the tuning strategy [Probst et al., 2019]. To quantify relevance of hyperparameters, models that inherently quantify feature relevance – e.g., GPs with ARD kernel [Neil, 1996] – can be used as surrogate models. Hutter et al. [2014a] quantified the importance of hyperparameters based on a random forest fitted on data generated by BO, for which the importance of both the main and the interaction effects of hyperparameters was calculated by a functional ANOVA approach. Similarly, Sharma et al. [2019] quantified the hyperparameter importance of

[2]There exist various implementations [Greenwell, 2017, Pedregosa et al., 2011]), extensions [Greenwell et al., 2018, Goldstein et al., 2015] and applications [Friedman and Meulman, 2003, Cutler et al., 2007].

[3]`https://github.com/slds-lmu/paper_2021_xautoml`

[4]Typically, the model error is estimated via cross-validation or hold-out testing.

residual neural networks. These works highlight how useful it is to quantify the importance of hyperparameters. However, importance scores do not show *how* a specific hyperparameter affects the model performance according to the surrogate model. Therefore, we propose to visualize the assumed marginal effect of a hyperparameter. A model-agnostic interpretation method that can be used for this purpose is the PDP.

PDPs for Hyperparameters. Let $S \subset \{1, 2, ..., d\}$ denote an index set of features, and let $C = \{1, 2, ..., d\} \setminus S$ be its complement. The partial dependence (PD) function [Friedman, 2001] of $c : \Lambda \to \mathbb{R}$ for hyperparameter(s) S is defined as[5]

$$c_S(\boldsymbol{\lambda}_S) := \mathbb{E}_{\boldsymbol{\lambda}_C}\left[c(\boldsymbol{\lambda})\right] = \int_{\Lambda_C} c(\boldsymbol{\lambda}_S, \boldsymbol{\lambda}_C)\, \mathrm{d}\mathbb{P}(\boldsymbol{\lambda}_C). \tag{1}$$

When analyzing the PDP of hyperparameters, we are usually interested in how their values $\boldsymbol{\lambda}_S$ impact model performance uniformly across the hyperparameter space. In line with prior work [Hutter et al., 2014a], we therefore assume $\mathbb{P}$ to be the uniform distribution over Λ_C. Computing $c_S(\boldsymbol{\lambda}_S)$ exactly is usually not possible because c is unknown and expensive to evaluate in the context of HPO. Thus, the posterior mean $\hat{m}$ of the probabilistic surrogate model $\hat{c}(\boldsymbol{\lambda})$ is commonly used as a proxy for c. Furthermore, the integral may not be analytically tractable for arbitrary surrogate models $\hat{c}$. Hence, the integral is approximated by Monte Carlo integration, i.e.,

$$\hat{c}_S\left(\boldsymbol{\lambda}_S\right) \quad = \quad \frac{1}{n}\sum\nolimits_{i=1}^{n} \hat{m}\left(\boldsymbol{\lambda}_S, \boldsymbol{\lambda}_C^{(i)}\right) \tag{2}$$

for a sample $\left(\boldsymbol{\lambda}_C^{(i)}\right)_{i=1,...,n} \sim \mathbb{P}(\boldsymbol{\lambda}_C)$. $\hat{m}\left(\boldsymbol{\lambda}_S, \boldsymbol{\lambda}_C^{(i)}\right)$ represents the marginal effect of $\boldsymbol{\lambda}_S$ for one specific instance i. Individual conditional expectation (ICE) curves [Goldstein et al., 2015] visualize the marginal effect of the i-th observation by plotting the value of $\hat{m}\left(\boldsymbol{\lambda}_S, \boldsymbol{\lambda}_C^{(i)}\right)$ against $\boldsymbol{\lambda}_S$ for a set of grid points[6] $\boldsymbol{\lambda}_S^{(g)} \in \Lambda_S$, $g \in \{1, ..., G\}$. Analogously, the PDP visualizes $\hat{c}_S(\boldsymbol{\lambda}_S)$ against the grid points. Following from Eq. 2, the PDP visualizes the average over all ICE curves. In HPO, the marginal predicted performance is a related concept. Instead of approximating the integral via Monte Carlo, the integral over $\hat{c}$ is computed exactly. Hutter et al. [2014a] propose an efficient approach to compute this integral for random forest surrogate models.

Uncertainty Quantification in PDPs. Quantifying the uncertainty of PDPs provides additional information about the reliability of the mean estimator. Hutter et al. [2014a] quantified the model uncertainty specifically for random forests as surrogates in BO by calculating the standard deviation of the marginal predictions of the individual trees. However, this procedure is not applicable to general probabilistic surrogate models, such as the commonly used GP. There are approaches that quantify the uncertainty for ML models that do not provide uncertainty estimates out-of-the-box. Cafri and Bailey [2016] suggested a bootstrap approach for tree ensembles to quantify the uncertainties of effects based on PDPs. Another approach to quantify the uncertainty of PDPs is to leverage the ICE curves. For example, Greenwell [2017] implemented a method that marginalizes over the mean $\pm$ standard deviation of the ICE curves for each grid point. However, this approach quantifies the underlying uncertainty of the data at hand rather than the model uncertainty, as explained in Appendix A.1. A model-agnostic estimate based on uncertainty estimates for probabilistic models is missing so far.

Subgroup PDPs. Recently, a new research direction concentrates on finding more reliable PDP estimates within subgroups of observations. Molnar et al. [2020] focused on problems in PDP estimation with correlated features. To that end, they apply transformation trees to find homogeneous subgroups and then visualize a PDP for each subgroup. Grömping [2020] looked at the same problem and also uses subgroup PDPs, where ICE curves are grouped regarding a correlated feature. Britton [2019] applied a clustering approach to group ICE curves to find interactions between features. However, none of these approaches aim at finding subgroups where reliable PDP estimates have low uncertainty. Additionally, to the best of our knowledge, nothing similar exists for analyzing experimental data created by HPO.

[5]To keep notation simple, we denote $c(\boldsymbol{\lambda})$ as a function of two arguments $(\boldsymbol{\lambda}_S, \boldsymbol{\lambda}_C)$ to differentiate components in the index set S from those in the complement. The integral shall be understood as a multiple integral of c where $\boldsymbol{\lambda}_j$, $j \in C$, are integrated out.

[6]Grid points are typically chosen as an equidistant grid or sampled from $\mathbb{P}(\boldsymbol{\lambda}_S)$. The granularity G is chosen by the user. For categorical features, the granularity typically corresponds to the number of categories.

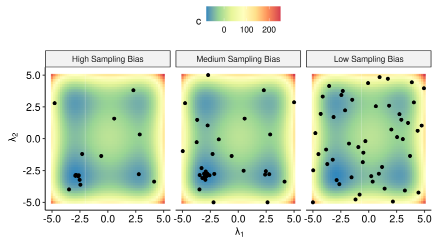


Figure 1: Illustration of the sampling bias when optimizing the $2D$ Styblinski Tang function with BO and the Lower Confidence Bound (LCB) acquisition function $a(\boldsymbol{\lambda}) = \hat{m}(\boldsymbol{\lambda}) + \tau \cdot \hat{s}(\boldsymbol{\lambda})$ for $\tau = 0.1$ (left) and $\tau = 2$ (middle) vs. data sampled uniformly at random (right).

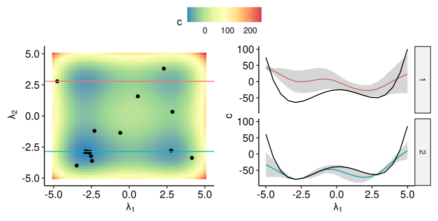


Figure 2: The two horizontal cuts (left) yield two ICE curves (right) showing the mean prediction and uncertainty band against λ_1 for $\hat{c}$ with $\tau = 0.1$ on the $2D$ Styblinski-Tang function. The upper ICE curve deviates more from the true effect (black) and shows a higher uncertainty.

3 Biased Sampling in HPO

Visualizing the marginal effect of hyperparameters of surrogate models via PDPs can be misleading. We show that this problem is due to the sequential nature of BO, which generates dependent instances (i.e., hyperparameter configurations) and thereby introduces a sampling and a resulting model bias. To save computational resources in contrast to grid search or random search, efficient optimizers like BO tend to exploit promising regions of the hyperparameter space while other regions are less explored (see Figure 1). Consequently, predictions of surrogate models are usually more accurate with less uncertainty in well-explored regions and less accurate with high uncertainty in under-explored regions. This model bias also affects the PD estimate (see Figure 2). ICE curves may be biased and less confident if they are computed in poorly-learned regions where the model has not seen much data before. Under the assumption of uniformly distributed hyperparameters, poorly-learned regions are incorporated in the PD estimate with the same weight as well-learned regions. ICE curves belonging to regions with high uncertainty may obfuscate well-learned effects of ICE curves belonging to other regions when they are aggregated to a PDP. Hence, the model bias may also lead to a less reliable PD estimate. PDPs visualizing only the mean estimator of Eq. (2) do not provide insights into the reliability of the PD estimate and how it is affected by the described model bias.

4 Quantifying Uncertainty in PDPs

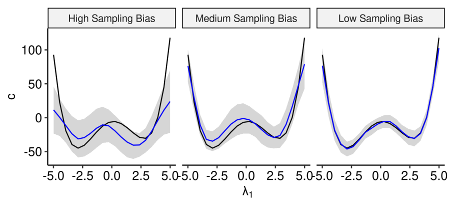


Figure 3: PDPs (blue) with confidence bands for surrogates trained on data created by BO and LCB with $\tau = 0.1$ (left), $\tau = 1$ (middle) and uniform i.i.d. dataset (right) vs. the true PD (black).

Pointwise uncertainty estimates of a probabilistic model provide insights into the reliability of the prediction $\hat{c}(\boldsymbol{\lambda})$ for a specific configuration $\boldsymbol{\lambda}$. This uncertainty directly correlates with how explored the region around $\boldsymbol{\lambda}$ is. Hence, including the model's uncertainty structure into the PD estimate enables users to understand in which regions the PDP is more reliable and which parts of the PDP must be cautiously interpreted.[7] We now extend the PDP of Eq. (2) to probabilistic surrogate models $\hat{c}$ (e.g., a GP). Let $\boldsymbol{\lambda}_S$ be a fixed grid point and $\left(\boldsymbol{\lambda}_C^{(i)}\right)_{i=1,\dots,n} \sim \mathbb{P}\left(\boldsymbol{\lambda}_C\right)$ a sample that is used to compute the Monte Carlo estimate of Eq. (2). The vector of predicted performances at the grid point $\boldsymbol{\lambda}_S$ is $\hat{\boldsymbol{c}}\left(\boldsymbol{\lambda}_S\right) = \left(\hat{c}\left(\boldsymbol{\lambda}_S, \boldsymbol{\lambda}_C^{(i)}\right)\right)_{i=1,\dots,n}$ with (posterior) mean $\hat{\boldsymbol{m}}\left(\boldsymbol{\lambda}_S\right) := \left(\hat{m}\left(\boldsymbol{\lambda}_S, \boldsymbol{\lambda}_C^{(i)}\right)\right)_{i=1,\dots,n}$ and

[7]Note that we aim at representing model uncertainty in a PD estimate, and not the variability of the mean prediction (see Appendix A.1 for a more detailed justification).

4

a (posterior) covariance $\hat{\boldsymbol{K}}(\boldsymbol{\lambda}_S) := \left(\hat{k}\left(\left(\boldsymbol{\lambda}_S, \boldsymbol{\lambda}_C^{(i)}\right), \left(\boldsymbol{\lambda}_S, \boldsymbol{\lambda}_C^{(j)}\right)\right)\right)_{i,j=1,\dots,n}$. Thus, $\hat{c}_S(\boldsymbol{\lambda}_S) = \frac{1}{n}\sum_{i=1}^n \hat{c}\left(\boldsymbol{\lambda}_S, \boldsymbol{\lambda}_C^{(i)}\right)$ is a random variable itself. The expected value of $\hat{c}_S(\boldsymbol{\lambda}_S)$ corresponds to the PD of the posterior mean function $\hat{m}$ at $\boldsymbol{\lambda}_S$, i.e.:

$$\hat{m}_S(\boldsymbol{\lambda}_S) = \mathbb{E}_{\hat{c}}\left[\hat{c}_S(\boldsymbol{\lambda}_S)\right] = \mathbb{E}_{\hat{c}}\left[\frac{1}{n}\sum_{i=1}^n \hat{c}\left(\boldsymbol{\lambda}_S, \boldsymbol{\lambda}_C^{(i)}\right)\right] = \frac{1}{n}\sum_{i=1}^n \hat{m}\left(\boldsymbol{\lambda}_S, \boldsymbol{\lambda}_C^{(i)}\right). \quad (3)$$

The variance of $\hat{c}_S(\boldsymbol{\lambda}_S)$ is

$$\hat{s}_S^2(\boldsymbol{\lambda}_S) = \mathbb{V}_{\hat{c}}\left[\hat{c}_S(\boldsymbol{\lambda}_S)\right] = \mathbb{V}_{\hat{c}}\left[\frac{1}{n}\sum_{i=1}^n \hat{c}\left(\boldsymbol{\lambda}_S, \boldsymbol{\lambda}_C^{(i)}\right)\right] = \frac{1}{n^2}\mathbf{1}^\top \hat{\boldsymbol{K}}(\boldsymbol{\lambda}_S)\,\mathbf{1}. \quad (4)$$

For the above estimate, it is important that the kernel is correctly specified such that the covariance structure is modeled properly by the surrogate model. Eq. (4) can be approximated empirically by treating the pairwise covariances as unknown, i.e.:

$$\hat{s}_S^2(\boldsymbol{\lambda}_S) \approx \frac{1}{n}\sum_{i=1}^n \hat{\boldsymbol{K}}(\boldsymbol{\lambda}_S)_{i,i}. \quad (5)$$

In Appendix A.2, we show empirically that this approximation is less sensitive to kernel misspecifications. Please note that the variance estimate and the mean estimate can also be applied to other probabilistic models, such as GAMLSS[8], transformation trees, or a random forest. An example for PDPs with uncertainty estimates is shown in Figure 3 for different degrees of a sampling bias.

5 Regional PDPs via Confidence Splitting

As discussed in Section 3, (efficient) optimization may imply that the sampling is biased, which in turn can produce misleading interpretations when IML is naively applied. We now aim to identify sub-regions $\Lambda' \subset \Lambda$ of the hyperparameter space in which the PD can be estimated with high confidence, and separate those from sub-regions in which it cannot be estimated reliably. In particular, we identify sub-regions in which poorly-learned effects do not obfuscate the well-learned effects along each grid point, thereby allowing the user to draw conclusions with higher confidence. By partitioning the entire hyperparameter space through a tree-based approach into disjoint and interpretable sub-regions, a more detailed understanding of the sampling process and hyperparameter effects is achieved. Users can either study the hyperparameter effect of a (confident) sub-region individually or understand the exploration-exploitation sampling of HPO by considering the complete tree structure. The result of this procedure for a single split is shown in Figure 5.

The PD estimate on the *entire* hyperparameter space Λ is computed by sampling the Monte Carlo estimate $(\boldsymbol{\lambda}_C^{(i)})_{i\in\mathcal{N}} \sim \mathbb{P}(\boldsymbol{\lambda}_C)$, $\mathcal{N} := \{1, 2, ..., n\}$. We now introduce the PD estimate on a *sub-region* $\Lambda' \subset \Lambda$ simply as $(\boldsymbol{\lambda}_C^{(i)})_{i\in\mathcal{N}'}$ only using $\mathcal{N}' = \{i \in \mathcal{N}\}_{\boldsymbol{\lambda}^{(i)}\in\Lambda'}$. Since we are interested in the marginal effect of the hyperparameter(s) S at each $\boldsymbol{\lambda}_S \in \Lambda_S$, we will usually visualize the PD for the whole range Λ_S. Thus, all obtained sub-regions should be of the form $\Lambda' = \Lambda_S \times \Lambda_C'$ with $\Lambda_C' \subset \Lambda_C$. This corresponds to an average of ICE curves in the set $i \in \mathcal{N}'$. The pseudo-code to partition a hyperparameter (sub-)space Λ and corresponding sample $(\boldsymbol{\lambda}_C^{(i)})_{i\in\mathcal{N}} \in \Lambda_C$, $\mathcal{N} \subseteq \{1, ..., n\}$, into two child regions is shown in Algorithm 1. This splitting is recursively applied in a CART[9]-like procedure [Breiman et al., 1984b] to expand a full tree structure, with the usual stopping criteria (e.g., a maximum number of splits, a minimum size of a region, or a minimum improvement in each node). In each leaf node, the sub-regional PDP and its corresponding uncertainty estimate are computed by aggregating over all contained ICE curves.

The criterion to evaluate a specific partitioning is based on the idea of grouping ICE curves with similar uncertainty structure. To be more exact, we evaluate the impurity of a PD estimate on a sub-region Λ' with the help of the associated set of observations $\mathcal{N}' = \{i \in \mathcal{N}\}_{\boldsymbol{\lambda}_C^{(i)}\in\Lambda_C'}$, also referred to as nodes, as follows: For each grid point $\boldsymbol{\lambda}_S$, we use the L2 loss in $L(\boldsymbol{\lambda}_S, \mathcal{N}')$ to evaluate how the

[8]Generalized additive models for location, scale and shape
[9]Classification and regression trees

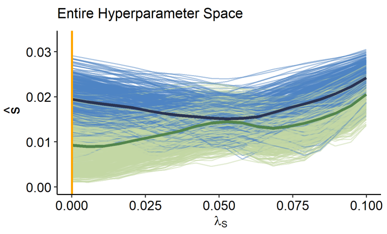


Figure 4: ICE curves of $\hat{s}$ of $\boldsymbol{\lambda}_S$ for the left (green) and right (blue) sub-region after the first split. The darker lines represent the respective PDPs. The orange vertical line marks the value λ_S of the optimal configuration.

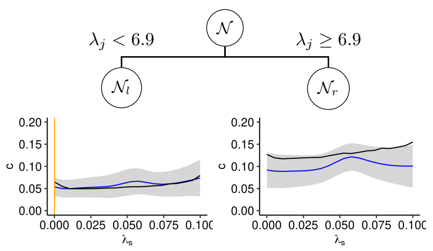


Figure 5: Example of two estimated PDPs (blue line) and 95% confidence bands after one partitioning step. The orange vertical line is the value of $\boldsymbol{\lambda}_S$ from the optimal configuration, the black curve is the true PD estimate $c_S(\boldsymbol{\lambda}_S)$.

uncertainty varies across all ICE estimates $i \in \mathcal{N}'$ using $\hat{s}^2_{S|\mathcal{N}'}(\boldsymbol{\lambda}_S) := \frac{1}{|\mathcal{N}'|}\sum_{i\in\mathcal{N}'} \hat{s}^2\left(\boldsymbol{\lambda}_S, \boldsymbol{\lambda}_C^{(i)}\right)$ and aggregate the loss $\mathcal{L}(\boldsymbol{\lambda}_S, \mathcal{N}')$ over all grid points in $\mathcal{R}_{L2}(\mathcal{N}')$:

$$\mathcal{L}(\boldsymbol{\lambda}_S, \mathcal{N}') = \sum\nolimits_{i\in\mathcal{N}'} \left(\hat{s}^2\left(\boldsymbol{\lambda}_S, \boldsymbol{\lambda}_C^{(i)}\right) - \hat{s}^2_{S|\mathcal{N}'}(\boldsymbol{\lambda}_S)\right)^2 \text{ and } \mathcal{R}_{L2}(\mathcal{N}') = \sum\nolimits_{g=1}^{G} \mathcal{L}(\boldsymbol{\lambda}_S^{(g)}, \mathcal{N}'). \quad (6)$$

Algorithm 1: Tree-based Partitioning

```
input: $\mathcal{N}$
for $j \in C$ do
    for Every split $t$ on hyperparameter $\lambda_j$ do
        $\mathcal{N}_l^{j,t} = \{i \in \mathcal{N}\}_{\lambda_j^{(i)} \leq t}$
        $\mathcal{N}_r^{j,t} = \{i \in \mathcal{N}\}_{\lambda_j^{(i)} > t}$
        $\mathcal{I}(j,t) = \mathcal{R}_{L_2}(\mathcal{N}_l^{j,t}) + \mathcal{R}_{L_2}(\mathcal{N}_r^{j,t})$
    end for
end for
Choose $\left(j^*, t^*_{\lambda_j^*}\right) \in \arg\min_{j,t} \mathcal{I}(j,t)$
Return $\mathcal{N}_l^{j,t}$ and $\mathcal{N}_r^{j,t}$ for $(j,t) = \left(j^*, t^*_{\lambda_j^*}\right)$
```

Hence, we measure the pointwise L_2-distance between ICE curves of the variance function $\hat{s}^2(\boldsymbol{\lambda}_S, \boldsymbol{\lambda}_C^{(i)})$ and its PD estimate $\hat{s}^2_{S|\mathcal{N}'}(\boldsymbol{\lambda}_S)$ within a sub-region $\mathcal{N}'$. This seems reasonable, as ICE curves in well-explored regions of the search space should, on average, have a lower uncertainty than those in less-explored regions. However, since we only split according to hyperparameters in C but not in S, the partitioning does not cut off less explored regions w.r.t. $\boldsymbol{\lambda}_S$. Thus, the chosen split criterion groups ICE curves of the uncertainty estimate such that we receive sub-regions associated with low costs c (and thus high relevance for a user) to be more confident in well-explored regions of $\boldsymbol{\lambda}_S$ and less confident in under-explored regions. Figure 4 shows that ICE curves of the uncertainty measure with high uncertainty over the entire range of $\boldsymbol{\lambda}_S$ are grouped together (right sub-region). Those with low uncertainty close to the optimal configuration of $\boldsymbol{\lambda}_S$ and increasing uncertainties for less suitable configurations are grouped together by curve similarities in the left sub-region. The respective PDPs are illustrated in Figure 5, where the confidence band in the left sub-region decreased compared to the confidence band of the global PDP especially for grid points close to the optimal value of $\boldsymbol{\lambda}_S$. Hence, by grouping observations with similar ICE curves of the variance function, resulting sub-regional PDPs with confidence bands provide the user with the information of which sub-regions of Λ_C are well-explored and lead to more reliable PDP estimates. Furthermore, the user will know which ranges of $\boldsymbol{\lambda}_S$ can be interpreted reliably and which ones need to be regarded with caution.

To sum up, the splitting procedure provides interpretable, disjoint sub-regions of the hyperparameter space. Based on the defined impurity measure, PDPs with high reliability can be identified and analyzed. In particular, the method provides more confident and reliable estimates in the sub-region containing the optimal configuration. Which PDPs are most interesting to explore depends on the question the user would like to answer. If the main interest lies in understanding the optimization and exploring the sampling process, a user might want to keep the number of sub-regions relatively low by performing only a few partitioning steps. Subsequently, one would investigate the overall structure of the sub-regions and the individual sub-regional PDPs. If users are more interested in interpreting hyperparameter effects only in the most relevant sub-region(s), they may want to split deeper and only look at sub-regions that are more confident than the global PDP.

Due to the nature of the splitting procedure, the PDP estimate on the entire hyperparameter space is a weighted average of the respective sub-regional PDPs. Hence, the global PDP estimate is decomposed into several sub-regional PDP estimates. Furthermore, note that the proposed method does not assume a numeric hyperparameter space, since the uncertainty estimates as well as ICE and PDP estimates can also be calculated for categorical features. Thus, it is applicable to problems with mixed spaces as long as a probabilistic surrogate model – and particularly its uncertainty estimates – are available. In Appendix B we describe how our method is applied to hierarchical hyperparameter spaces.

Since the proposed method is an instance of the CART algorithm, finding the optimal split for a categorical variable with q levels generally involves checking 2^q subsets. This becomes computationally infeasible for high values of q. It remains an open question for future work if this can be sped by an optimal procedure as in regression with L2 loss [Fisher, 1958] and binary classification [Breiman et al., 1984a] or by a clever heuristic as for multiclass classification Wright and König [2019].

6 Experimental Analysis

In this section, we validate the effectiveness of the introduced methods. We formulate two main hypotheses: First, experimental data affected by the sampling bias lead to biased surrogate models and thus to unreliable and misleading PDPs. Second, the proposed partitioning allows us to identify an interpretable sub-region (around the optimal configuration) that yields a more reliable and confident PDP estimate. In a first experiment, we apply our methods to BO runs on a synthetic function. In this controlled setup, we investigate the validity of our hypotheses with regards to problems of different dimensionality and different degrees of sampling bias. In a second experiment, we evaluate our PDP partitioning in the context of HPO for neural networks on a variety of tabular datasets.

We assess the sampling bias of the optimization design points by comparing their empirical distribution to a uniform distribution via Maximum Mean Discrepancy (MMD) [Gretton et al., 2012, Molnar et al., 2020], which is covered in more detail in the Appendix C.1. We measure the reliability of a PDP, i.e., the degree to which a user can rely on the estimate of the PD estimate, by comparing it to the true PD $c_S(\boldsymbol{\lambda}_S)$ as defined in Eq. (1). More specifically, for every grid point $\boldsymbol{\lambda}_S^{(g)}$, we compute the negative log-likelihood (NLL) of $c_S(\boldsymbol{\lambda}_S)$ under the distribution of $\hat{c}_S\left(\boldsymbol{\lambda}_S\right)$ pointwise for every grid point $\boldsymbol{\lambda}_S^{(g)}$. The confidence of a PDP is illustrated by the width of its confidence bands $\hat{m}_S\left(\boldsymbol{\lambda}_S\right) \pm q_{1-\alpha/2} \cdot \hat{s}_S\left(\boldsymbol{\lambda}_S\right)$, with $q_{1-\alpha/2}$ denoting the $(1-\alpha/2)$-quantile of a standard normal distribution. We measure the confidence by assessing $\hat{s}_S(\boldsymbol{\lambda}_S)$ pointwise for every grid point. In particular, we consider the mean confidence (MC) across all grid points $\frac{1}{G}\sum_{g=1}^{G} \hat{s}\left(\boldsymbol{\lambda}_S^{(g)}\right)$ as well as the confidence at the grid point closest to $\hat{\boldsymbol{\lambda}}_S$ abbreviated by OC, with $\hat{\boldsymbol{\lambda}}$ being the best configuration evaluated by the optimizer. To evaluate the performance of the confidence splitting, we report the above metrics on the sub-region that contains the best configuration evaluated by the optimizer, assuming that this region is of particular interest for a user of HPO. PDPs are computed with regards to single features for $G = 20$ equidistant grid points and $n = 1000$ Monte Carlo samples.

6.1 BO on a Synthetic Function

We consider the d-dimensional Styblinski-Tang function $c : [-5,5]^d \rightarrow \mathbb{R}$, $\boldsymbol{\lambda} \mapsto \frac{1}{2}\sum_{i=1}^{d}\left(\boldsymbol{\lambda}_i^4 + 16\boldsymbol{\lambda}_i^2 + 5\boldsymbol{\lambda}_i\right)$ for $d \in \{3,5,8\}$. Since the PD is the same for each dimension i, we only present the effects of $\boldsymbol{\lambda}_1$. We performed BO with a GP surrogate model with a Matérn-3/2 kernel and the LCB acquisition function $a(\boldsymbol{\lambda}) = \hat{m}(\boldsymbol{\lambda}) + \tau \cdot \hat{s}(\boldsymbol{\lambda})$ with different values $\tau \in \{0.1, 1, 5\}$ to control the sampling bias. We compute the global PDP with confidence bands estimated according to Eq. (5) for the GP surrogate model $\hat{c}$ that was fitted in the *last* iteration of BO. We ran Algorithm 1, and computed the PDP in the sub-region containing the optimal configuration. All computations were repeated 30 times. Further details on the setup are given in Appendix C.2.1.

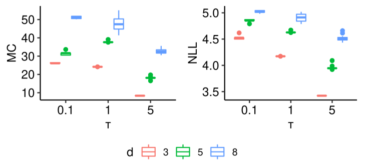


Figure 6: The figure presents the MC (left) and the NLL (right) for $d \in \{3, 5, 8\}$ for a high ($\tau = 0.1$), medium ($\tau = 1$), and low ($\tau = 5$) sampling bias across 30 replications. With a lower sampling bias, we obtain narrower confidence bands and a lower NLL.

Table 1: The table shows the relative improvement of the MC and the NLL via Algorithm 1 with 1 and 3 splits, compared to the global PDP along with the sampling bias for a $\tau = 0.1$ (high), $\tau = 2$ (medium), and $\tau = 5$ (low). Results are averaged across 30 replications.

		δ MC (%)		δ NLL (%)	
d	MMD	$n_{\text{sp}} = 1$	$n_{\text{sp}} = 3$	$n_{\text{sp}} = 1$	$n_{\text{sp}} = 3$
3	low (0.18)	7.65	13.64	5.89	10.92
3	medium (0.51)	12.86	36.92	4.78	7.70
3	high (0.56)	16.52	34.84	2.77	-1.62
5	low (0.15)	6.63	15.45	2.82	6.05
5	medium (0.45)	19.67	37.28	4.05	7.80
5	high (0.53)	11.99	33.06	-3.86	-1.93
8	low (0.11)	3.58	9.67	0.84	2.40
8	medium (0.42)	8.86	23.03	1.51	3.30
8	high (0.56)	6.59	19.84	1.53	4.29

As presented in Figure 6, the PDPs for surrogate models trained on *less biased* data (measured by the MMD) yield *lower* values of the NLL, as well as *lower* values for the MC. Table 1 shows that a single tree-based split reduces the MC by up to almost 20%, and up to 37% when performing 3 partitioning steps. Additionally, the NLL improves with an increasing number of partitioning steps in most cases. The results on the synthetic functions support our second hypothesis that the tree-based partitioning improves the reliability in terms of the NLL and the confidence of the PD estimates. The improvement of the MC is higher for a medium to high sampling bias, compared to scenarios that are less affected by sampling bias. We observe that (particularly for high sampling bias) there are some outlier cases in which the NLL worsens. More detailed results are shown in Appendix C.3.1.

6.2 HPO on Deep Learning

In a second experiment, we investigate HPO in the context of a surrogate benchmark [Eggensperger et al., 2015] based on the LCBench data [Zimmer et al., 2021]. For each of the 35 different OpenML [Vanschoren et al., 2013] classification tasks, LCBench provides access to evaluations of a deep neural network on 2000 configurations randomly drawn from the configuration space defined by Auto-PyTorch Tabular (see Table 5 in Appendix C.2). For each task, we trained a random forest as an empirical performance model that predicts the balanced validation error of the neural network for a given configuration. These empirical performance models serve as cheap to evaluate objective functions, which efficiently approximate the result of the real-world experiment of running a deep learning configuration on an LCBench instance. BO then acts on this empirical performance model as its objective[10].

For each task, we ran BO to obtain the optimal architecture and hyperparameter configuration. Again, we used a GP with a Matérn-3/2 kernel and LCB with $\tau = 1$. Each BO run was allotted a budget of 200 objective function evaluations. We computed the PDPs and their confidences, which are estimated according to Eq. (5), based on the surrogate model $\hat{c}$ after the final iteration. We performed tree-based partitioning with up to 6 splits based on a uniformly distributed dataset of size $n = 1000$. All computations were statistically repeated 30 times. Further details are provided in Appendix C.2.2.

For the real-world data example, we focus on answering the second hypothesis, i.e., whether the tree-based Algorithm 1 improves the reliability of the PD estimates. We compare the PDP in sub-regions after 6 splits with the global PDP. We computed the relative improvement of the confidence (MC and OC) and the NLL of the sub-regional PDP compared to the respective estimates for the global PDP. As shown in Table 2, the MC of the PDPs is on average reduced by 30% to 52%, depending on the hyperparameter. At the optimal configuration $\hat{\boldsymbol{\lambda}}_S$, the improvement even increases to $50\% - 62\%$. Thus, PDP estimates for all hyperparameters are on average – independent of the underlying dataset – clearly more confident in the relevant sub-regions when compared to the global PD estimates, especially around the optimal configuration $\hat{\boldsymbol{\lambda}}_S$. In addition to the MC, the NLL simultaneously improves. In Appendix C.3.2, we provide details regarding the evaluated metrics on the level of the dataset and demonstrate that our split criterion outperforms other impurity measures regarding MC

[10]Please note that the random forest is only used as a surrogate in order to construct an efficient benchmark objective, and not as a surrogate in the BO algorithm, where we use a GP.

and OC. Furthermore, we emphasize in Appendix C.3.2 the significance of our results by providing a comparison to a naive baseline method.

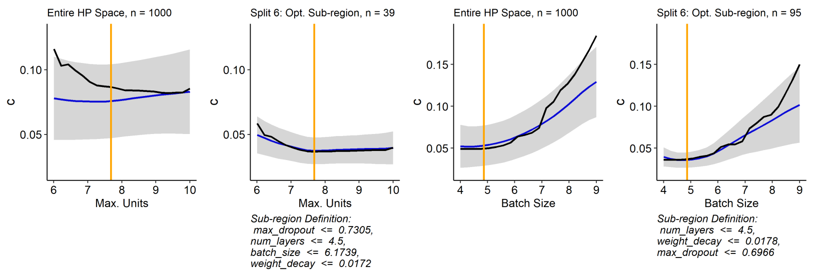


Figure 7: PDP (blue) and confidence band (grey) of the GP for hyperparameter *max. number of units* (*batch size*) on the left (right) side. The black line shows the PDP of the meta surrogate model representing the true PDP estimate. The orange vertical line marks the optimal configuration $\hat{\boldsymbol{\lambda}}_S$. The relative improvements from the global PDP to the sub-regional PDP after 6 splits are for *max. number of units* (*batch size*): δ MC = 61.6% (28.4%), δ OC = 63.5% (62.2%), δ NLL = 48.6% (30.1%).

To further study our suggested method, we now highlight a few individual experiments. We chose one iteration of the *shuttle* dataset. On the two left plots of Figure 7, we see that the true PDP estimate for *max. number of units* is decreasing, while the globally estimated PDP trend is increasing and thus misleading. Although the confidence band already indicates that the PDP cannot be reliably interpreted on the entire hyperparameter space, it remains challenging to draw any conclusions from it. After performing 6 splits, we receive a confident and reliable PD estimate on an interpretable sub-region. The same plots are depicted for the hyperparameter *batch size* on the right part of Figure 7. This example illustrates that the confidence band might not always shrink uniformly over the entire range of $\boldsymbol{\lambda}_S$ during the partitioning, but often particularly around the optimal configuration $\hat{\boldsymbol{\lambda}}_S$.

Table 2: Relative improvement of MC, OC, and NLL on hyperparameter level. The table shows the respective mean (standard deviation) of the average relative improvement over 30 replications for each dataset and 6 splits.

Hyperparameter	δ MC (%)	δ OC (%)	δ NLL (%)
Batch size	40.8 (14.9)	61.9 (13.5)	19.8 (19.5)
Learning rate	50.2 (13.7)	57.6 (14.4)	17.9 (20.5)
Max. dropout	49.7 (15.4)	62.4 (11.9)	17.4 (18.2)
Max. units	51.1 (15.2)	58.6 (12.7)	24.6 (22.0)
Momentum	51.7 (14.5)	58.3 (12.7)	19.7 (21.7)
Number of layers	30.6 (16.4)	50.9 (16.6)	13.8 (32.5)
Weight decay	36.3 (22.6)	61.0 (13.1)	11.9 (19.7)

7 Discussion and Conclusion

In this paper, we showed that partial dependence estimates for surrogate models fitted on experimental data generated by efficient hyperparameter optimization can be unreliable due to an underlying sampling bias. We extended PDPs by an uncertainty estimate to provide users with more information regarding the reliability of the mean estimator. Furthermore, we introduced a tree-based partitioning approach for PDPs, where we leverage the uncertainty estimator to decompose the hyperparameter space into interpretable, disjoint sub-regions. We showed with two experimental studies that we generate, on average, more confident and more reliable regional PDP estimates in the sub-region containing the optimal configuration compared to the global PDP.

One of the main limitations of PDPs is that they bear the risk of providing misleading results if applied to correlated data in the presence of interactions, especially for nonparametric models [Grömping, 2020]. However, existing alternatives that visualize the global marginal effect of a feature such as accumulated local effect (ALE) plots [Apley and Zhu, 2020] do also not provide a fully satisfying solution to this problem [Grömping, 2020]. As a solution to this problem, Grömping [2020] suggests stratified PDPs by conditioning on a correlated and potentially interacting feature to group ICE curves. This idea is in the spirit of our introduced tree-based partitioning algorithm. However, in the context of BO we might assume the distribution in Eq. (1) to be uniform and therefore no correlations are present. Instead of correlated features, we are faced with a sampling bias (see Section 3) where we observe regions of varying uncertainty. Hence, instead of stratifying with respect to correlated features and aggregating ICE curves in regions with less correlated features, we stratify with respect to uncertainty and aggregate ICE curves in regions with low uncertainty variation. Nonetheless, it might be interesting to compare our approach with approaches based on the considerations made by Grömping [2020] – or potentially improved ALE curves.

Another limitation when using single-feature PDPs as in our examples is that hyperparameter interactions are not visible. However, two-way interactions can be visualized by plotting two-dimensional PDPs within sub-regions. Another possibility to detect interactions is to look at ICE curves within the sub-regions. If the shape of ICE curves within a sub-region is very heterogeneous, it indicates that the hyperparameter under consideration interacts with one of the other hyperparameters. Hence, having the additional possibility to look at ICE curves of individual observations within a sub-region is an advantage compared to other global feature effect plots such as ALE plots [Apley and Zhu, 2020], as they are not defined on an observational level. While we mainly discussed GP surrogate models on a numerical hyperparameter space in our examples, our methods are applicable to a wide variety of distributional regression models and also for mixed and hierarchical hyperparameter spaces. We also considered in Appendix C.3.2 different impurity measures. While the one introduced in this paper performed best in our experimental settings, this impurity measure as well as other components are exchangeable within the proposed algorithm. In the future, we will study our method on more complex, hierarchical configuration spaces for neural architecture search.

The proposed interpretation method is based on a surrogate and consequently does provide insights about what the AutoML system has *learned*, which in turn allows plausibility checks and may increase trust in the system. To what extent this allows conclusions on the *true* underlying hyperparameter effects depends on the quality of the surrogate. How to efficiently perform model diagnostics to ensure a high surrogate quality before applying interpretability techniques is subject to future research.

While we focused on providing better explanations without generating any additional experimental data, it might be interesting to investigate in future work how confidence and reliability of IML methods can be increased most efficiently when a user is allowed to conduct additional experiments.

Overall, we believe that increasing interpretability of AutoML will pave the way for human-centered AutoML. Our vision is that users will be able to better understand the reasoning and the sampling process of AutoML systems and thus can either trust and accept the results of the AutoML system or interact with it in a feedback loop based on the gained insights and their preferences. How users can then best interact with AutoML (beyond simple changes of the configuration space) will be left open for future research.

Acknowledgments and Disclosure of Funding

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

D. W. Apley and J. Zhu. Visualizing the effects of predictor variables in black box supervised learning models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 82(4):1059–1086, 2020.

B. Bischl, J. Richter, J. Bossek, D. Horn, J. Thomas, and M. Lang. mlrmbo: A modular framework for model-based optimization of expensive black-box functions, 2018.

L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth, 1984a. ISBN 0-534-98053-8.

L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth, 1984b.

M. Britton. VINE: Visualizing statistical interactions in black box models. *CoRR*, abs/1904.00561, 2019.

G. Cafri and B. A. Bailey. Understanding variable effects from black box prediction: Quantifying effects in tree ensembles using partial dependence. *Journal of Data Science*, 14(1):67–95, 2016.

D. R. Cutler, T. C. Edwards Jr, K. H. Beard, A. Cutler, K. T. Hess, J. Gibson, and J. J. Lawler. Random forests for classification in ecology. *Ecology*, 88(11):2783–2792, 2007.

J. Drozdal, J. Weisz, D. Wang, G. Dass, B. Yao, C. Zhao, M. Muller, L. Ju, and H. Su. Trust in AutoML: Exploring information needs for establishing trust in automated machine learning systems. In *Proceedings of the 25th International Conference on Intelligent User Interfaces*, pages 297–307, 2020.

K. Eggensperger, F. Hutter, H. Hoos, and K. Leyton-Brown. Efficient benchmarking of hyperparameter optimizers via surrogates. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, pages 1114–1120, 2015.

W. D. Fisher. On grouping for maximum homogeneity. *Journal of the American statistical Association*, 53(284): 789–798, 1958.

A. A. Freitas. Automated machine learning for studying the trade-off between predictive accuracy and interpretability. In *Third IFIP International Cross-Domain Conference for Machine Learning and Knowledge Extraction (CD-MAKE 2019)*, volume 11713, pages 48–66. Springer, August 2019.

J. H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, pages 1189–1232, 2001.

J. H. Friedman and J. J. Meulman. Multiple additive regression trees with application in epidemiology. *Statistics in medicine*, 22(9):1365–1381, 2003.

A. Goldstein, A. Kapelner, J. Bleich, and E. Pitkin. Peeking inside the black box: Visualizing statistical learning with plots of individual conditional expectation. *Journal of Computational and Graphical Statistics*, 24(1): 44–65, 2015.

B. M. Greenwell. pdp: An R Package for Constructing Partial Dependence Plots. *The R Journal*, 9(1):421–436, 2017.

B. M. Greenwell, B. C. Boehmke, and A. J. McCarthy. A simple and effective model-based variable importance measure. *arXiv preprint arXiv:1805.04755*, 2018.

A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. Smola. A kernel two-sample test. *Journal of Machine Learning Research*, 13(25):723–773, 2012.

U. Grömping. Model-agnostic effects plots for interpreting machine learning models. Report 1/2020, Reports in Mathematics, Physics and Chemistry. Department II, Beuth University of Applied Sciences Berlin, 2020.

F. Hutter, H. H. Hoos, and K. Leyton-Brown. An efficient approach for assessing hyperparameter importance. In *Proceedings of the 31th International Conference on Machine Learning, ICML*, volume 32, pages 754–762. JMLR.org, 2014a.

F. Hutter, L. Xu, H. H. Hoos, and K. Leyton-Brown. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence*, 206:79–111, 2014b.

J. Levesque, A. Durand, C. Gagné, and R. Sabourin. Bayesian optimization for conditional hyperparameter spaces. In *2017 International Joint Conference on Neural Networks, IJCNN 2017, Anchorage, AK, USA, May 14-19, 2017*, pages 286–293. IEEE, 2017. doi: 10.1109/IJCNN.2017.7965867. URL `https://doi.org/10.1109/IJCNN.2017.7965867`.

C. Molnar, G. König, B. Bischl, and G. Casalicchio. Model-agnostic feature importance and effects with dependent features - A conditional subgroup approach. *CoRR*, abs/2006.04628, 2020.

R. M. Neil. *Bayesian Learning for Neural Networks*. Springer-Verlag, Berlin, Heidelberg, 1996. ISBN 0387947248.

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

F. Pfisterer, J. Thomas, and B. Bischl. Towards human centered AutoML. *CoRR*, abs/1911.02391, 2019.

P. Probst, A. Boulesteix, and B. Bischl. Tunability: Importance of hyperparameters of machine learning algorithms. *Journal of Machine Learning Research*, 20:53:1–53:32, 2019.

A. Sharma, J. N. van Rijn, F. Hutter, and A. Müller. Hyperparameter importance for image classification by residual neural networks. In *Discovery Science - 22nd International Conference, DS*, volume 11828 of *Lecture Notes in Computer Science*, pages 112–126. Springer, 2019.

J. Snoek, H. Larochelle, and R. P. Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems 25*, pages 2960–2968, 2012.

K. Swersky, D. Duvenaud, J. Snoek, F. Hutter, and M. A. Osborne. Raiders of the lost architecture: Kernels for bayesian optimization in conditional parameter spaces. *arXiv: Machine Learning*, 2014.

J. Vanschoren, J. N. van Rijn, B. Bischl, and L. Torgo. Openml: networked science in machine learning. *SIGKDD Explor.*, 15(2):49–60, 2013.

M. N. Wright and I. R. König. Splitting on categorical predictors in random forests. *PeerJ*, 7:e6339, 2019.

I. Xanthopoulos, I. Tsamardinos, V. Christophides, E. Simon, and A. Salinger. Putting the human back in the AutoML loop. In *Proceedings of the Workshops of the EDBT/ICDT 2020 Joint Conference*, volume 2578 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2020.

L. Zimmer, M. Lindauer, and F. Hutter. Auto-PyTorch Tabular: Multi-fidelity metalearning for efficient and robust AutoDL. *IEEE TPAMI*, 2021. Preprint via Early Access.

5.2. Improving Accuracy of Interpretability Measures in Hyperparameter Optimization via Bayesian Algorithm Execution

The paper presents an improved model-based sampling strategy for HPO which increases the accuracy of interpretability methods applied on a respective surrogate model without a considerable drop in optimization performance. While Chapter 5.1 proposes an improved interpretability method to account for a sampling bias post-hoc without changing the sampling, this paper improves the sampling directly. It can thus be also be classified into Section 3.2 – *Interpretability methods (B)*.

Contributing article:

Moosbauer, J., Casalicchio, G., Lindauer, M., Bischl, B. (2023). Improving Accuracy of Interpretability Measures in Hyperparameter Optimization via Bayesian Algorithm Execution. *arXiv preprint arXiv:2206.05447*.

Author contributions:

The idea for the project and for the method was developed by Julia Moosbauer. The whole paper was drafted and, in most parts, written by Julia Moosbauer. All experiments were conducted by Julia Moosbauer. All authors contributed to revisions of the paper. Giuseppe Casalichhio, Marius Lindauer, and Bernd Bischl gave valuable input throughout the project and suggested several notable modifications.

Supplementary material available at:

- Supplementary material: `https://arxiv.org/abs/2206.05447` (full paper)

Improving Accuracy of Interpretability Measures in Hyperparameter Optimization via Bayesian Algorithm Execution

Julia Moosbauer *Julia.Moosbauer@stat.uni-muenchen.de*
Institute of Statistics, Munich Center for Machine Learning (MCML)
Ludwig-Maximilians-Universität München

Giuseppe Casalicchio *Giuseppe.Casalicchio@stat.uni-muenchen.de*
Institute of Statistics, Munich Center for Machine Learning (MCML)
Ludwig-Maximilians-Universität München

Marius Lindauer *lindauer@tnt.uni-hannover.de*
Institute of Artificial Intelligence
Leibniz University Hannover

Bernd Bischl *Bernd.Bischl@stat.uni-muenchen.de*
Institute of Statistics, Munich Center for Machine Learning (MCML)
Ludwig-Maximilians-Universität München

Abstract

Despite all the benefits of automated hyperparameter optimization (HPO), most modern HPO algorithms are black-boxes themselves. This makes it difficult to understand the decision process which leads to the selected configuration, reduces trust in HPO, and thus hinders its broad adoption. Here, we study the combination of HPO with interpretable machine learning (IML) methods such as partial dependence plots. These techniques are more and more used to explain the marginal effect of hyperparameters on the black-box cost function or to quantify the importance of hyperparameters. However, if such methods are naively applied to the experimental data of the HPO process in a post-hoc manner, the underlying sampling bias of the optimizer can distort interpretations. We propose a modified HPO method which efficiently balances the search for the global optimum w.r.t. predictive performance *and* the reliable estimation of IML explanations of an underlying black-box function by coupling Bayesian optimization and Bayesian Algorithm Execution. On benchmark cases of both synthetic objectives and HPO of a neural network, we demonstrate that our method returns more reliable explanations of the underlying black-box without a loss of optimization performance.

1 Introduction

The performance of machine learning (ML) models usually depends on many decisions, such as the choice of a learning algorithm and its hyperparameter configurations. Manually reaching these decisions is usually a tedious trial-and-error process. Automated machine learning (AutoML), e.g., hyperparameter optimization (HPO), can support developers and researchers in this regard. By framing these decisions as an optimization problem and solving them using efficient black-box optimizers such as Bayesian Optimization (BO), HPO is demonstrably more efficient than manual tuning, and grid or random search (Bergstra et al., 2011; Snoek et al., 2012; Turner et al., 2020; Bischl et al., 2021). However, there is still a lack of confidence in AutoML systems and a reluctance to trust the returned best configuration (Drozdal et al., 2020). One reason for why some practitioners still today prefer manual tuning over automated HPO is that existing systems lack the ability to convey an understanding of hyperparameter importance and how certain hyperparameters affect model performance (Hasebrook et al., 2022), helping them to understand why a final configuration was chosen.

Desirable insights into hyperparameter effects or importance could in principle be generated by applying methods of interpretable machine learning (IML) to experimental data from the HPO process, specifically the final surrogate model generated by BO based on this HPO-data. However, these methods – even though possible from a technical perspective and used before (Hutter et al., 2014; Van Rijn & Hutter, 2018; Young et al., 2018; Head et al., 2022) – should be used with caution in this context. The main reason is a sampling bias caused by the desire for efficient optimization during HPO (Moosbauer et al., 2021): Efficient optimizers typically sample more configurations in promising regions with potentially well-performing hyperparameter configurations, while other regions are underrepresented. This sampling bias introduces a surrogate-model bias in under-explored regions as the surrogate model is subject to high uncertainty in these regions. Consequently, explanations of HPO runs, such as partial dependence plots (PDPs) (Friedman, 2001), can be misleading as they also rely on artificially created evaluations in under-explored regions. Moosbauer et al. (2021) had been the first to address this issue, and had proposed an approach to identify well-explored, rather small subregions in which PDPs can be estimated accurately. While this is valuable, it still does not allow to accurately estimate hyperparameter effects globally.

To anticipate these unintended effects of this sampling bias as effectively as possible already during the HPO process, we propose a modified BO algorithm that efficiently searches for the global optimum *and* accurate IML estimates of the underlying black-box function at the same time. We build on the concept of Bayesian Algorithm Execution (BAX) (Neiswanger et al., 2021) to estimate the expected information gain (EIG) (Lindley, 1956) of configurations w.r.t. the output of an interpretation method. We ultimately couple BO with BAX and propose BOBAX as an efficient method that searches for accurate interpretations without a relevant loss of optimization performance. Our proposed method is generic as it is applicable to any BO variant (e.g., different acquisition functions or probabilistic surrogate models). As IML technique we focus on PDPs (Friedman, 2001), which estimate the marginal effect(s) of features (in our case: hyperparameters) on the output by visualizing a marginal 1D or 2D function. PDPs constitute an established IML technique (Lemmens & Croux, 2006; Cutler et al., 2007; Wenger & Olden, 2012; Zhang et al., 2018), have been in use for more than 20 years to analyze ML models, and have recently gained further interest in IML and XAI, and are also increasingly used to analyze hyperparameter effects in HPO and AutoML (Young et al., 2018; Zela et al., 2018; Head et al., 2022). We point out that our technique is in principle not limited to PDPs, but can be combined with any IML technique which can be quantitatively estimated from a surrogate model.

In a benchmark study, we demonstrate how BOBAX consistently yields more reliable estimates for marginal effects estimated via the partial dependence method while maintaining the same level of optimization efficiency as commonly used methods. Finally, we demonstrate how BOBAX can give reliable insights into hyperparameter effects of a neural network during tuning yielding state-of-the-art performance. We believe that through our generic method, the potential of IML methods can be unlocked in the context of HPO, thus paving the way for more interpretability of and trust into human-centered HPO. Our contributions include:

1. The direct optimization for an accurate estimation of IML statistics, e.g., marginal effects for single or multiple hyperparameters, as part of BO for HPO, making HPO interpretable and more trustworthy;
2. The combination of BO and Bayesian Algorithm Execution (BAX), dubbed BOBAX, where BAX is used to guide the search towards more accurate estimation of IML statistics;
3. Thorough study of different variants of BOBAX and baselines on synthetic functions; and
4. Empirical evidence that budget allocation regarding IML estimates does not come at the expense of significantly reduced optimization performance on a deep learning HPO benchmark.

2 Background

In this section, we formalize HPO and BO as the context of our work. We also give an overview of Bayesian Algorithm Execution (BAX) as it serves as basis for our work.

Hyperparameter Optimization The aim of HPO is to efficiently find a well-performing configuration of a learning algorithm. HPO is therefore commonly formalized as finding the minimizer $\boldsymbol{\lambda}^* \in \arg\min_{\boldsymbol{\lambda}\in\Lambda} c(\boldsymbol{\lambda})$

of a *black-box* cost function $c : \Lambda \to \mathbb{R}$ which maps a hyperparameter configuration $\boldsymbol{\lambda} = (\lambda_1, ..., \lambda_d) \in \Lambda$ to the validation error of the model trained by a learning algorithm run using $\boldsymbol{\lambda}$. The hyperparameter space $\Lambda = \Lambda_1 \times ... \times \Lambda_d$ can be mixed, containing categorical and continuous hyperparameters. Particularly in the context of AutoML, where whole machine learning pipeline configurations are optimized over, Λ may even contain hierarchical dependencies between hyperparameters (Thornton et al., 2013; Olson & Moore, 2016).

Bayesian Optimization BO is a black-box optimization algorithm which has become increasingly popular in the context of HPO (Jones et al., 1998; Snoek et al., 2012). BO sequentially chooses configurations $\boldsymbol{\lambda}^{(1)}, ..., \boldsymbol{\lambda}^{(T)}$ that are evaluated $c_{\boldsymbol{\lambda}^{(1)}}, ..., c_{\boldsymbol{\lambda}^{(T)}}$ to obtain an archive $A_T = \left\{\left(\boldsymbol{\lambda}^{(i)}, c_{\boldsymbol{\lambda}^{(i)}}\right)\right\}_{i=1,...,T}$. To choose the next configuration $\boldsymbol{\lambda}^{(T+1)}$ as efficiently as possible, a surrogate model $\hat{c}$ is estimated on the archive A_T, and a new point is proposed based on an acquisition function that leverages information from the surrogate model $\hat{c}$. Typically, we chose a probabilistic model and estimate a distribution over c, denoted by $p(c \mid A_T)$. A common choice are Gaussian processes $c \sim \mathcal{GP}(\mu, k)$, characterized by a mean function $\mu : \Lambda \to \mathbb{R}$ and a covariance function $k : \Lambda \times \Lambda \to \mathbb{R}$. Acquisition functions usually trade off exploration (i.e., sampling in regions with few data points and high posterior uncertainty) and exploitation (i.e., sampling in regions with low mean). Common examples are the expected improvement (EI) (Jones et al., 1998), the lower confidence bound (LCB) (Jones, 2001; Srinivas et al., 2010), entropy search (Hennig & Schuler, 2012; Hernández-Lobato et al., 2014) and knowledge gradient (Wu et al., 2017).

Marginal Effects of Hyperparameters Practitioners of HPO are often interested in whether and how individual hyperparameters affect model performance. Not only is there a desire to gain model comprehension Hasebrook et al. (2022), also such insights can influence decisions, for example whether to tune a hyperparameter or not (Probst et al., 2019), or modify hyperparameter ranges. One interpretation measure that the community is looking at (Hutter et al., 2014; Zela et al., 2018; Young et al., 2018; Van Rijn & Hutter, 2018; Zöller et al., 2022) is the marginal effect of one or multiple hyperparameters $\boldsymbol{\lambda}_S$, $S \subset \{1, 2, ..., d\}$ on model performance, which is defined as[1]

$$c_S(\boldsymbol{\lambda}_S) := \mathbb{E}_{\boldsymbol{\lambda}_R}\left[c(\boldsymbol{\lambda})\right] = \int_{\Lambda_R} c(\boldsymbol{\lambda}_S, \boldsymbol{\lambda}_R)\ \mathrm{d}\mathbb{P}(\boldsymbol{\lambda}_R). \tag{1}$$

In the context of HPO, $\mathbb{P}$ is typically assumed to be the uniform distribution over Λ_R since we are interested in how hyperparameter values $\boldsymbol{\lambda}_S$ impact model performance uniformly across the hyperparameter space (Hutter et al., 2014; Moosbauer et al., 2021). Since computing Eq. (1) analytically is usually possible, the PDP method (Friedman, 2001) approximates the integral, as in Eq. (1), by Monte Carlo approximation.

Information-based Bayesian Algorithm Execution Information-based Bayesian Algorithm Execution (BAX) extends the idea of using entropy search for estimating global optima to estimating other properties of a function $f : \mathcal{X} \to \mathbb{R}$ (Neiswanger et al., 2021). Similar to BO, BAX tries to sequentially choose points $\mathbf{x}^{(i)} \in \mathcal{X}$ in order to estimate the quantity of interest accurately with as few evaluations as possible. It is assumed that the quantity of interest can be computed as the output $\mathcal{O}_\mathcal{A} := \mathcal{O}_\mathcal{A}(f)$ of running an algorithm $\mathcal{A}$ on f, e.g. top-k estimation on a finite set, computing level sets or finding shortest paths.

Similarly to BO, BAX sequentially builds a probabilistic model $p(f \mid A_T)$, e.g., a GP, over an archive of evaluated points A_T. Based on $p(f \mid A_T)$, they derive the posterior distribution over the algorithm output $p(\mathcal{O}_\mathcal{A} \mid A_T)$. To build the archive A_T as efficiently as possible, they choose to evaluate the point $\mathbf{x}^{(T+1)}$ which maximizes the expected information gain about the algorithm output $\mathcal{O}_\mathcal{A}$

$$\mathrm{EIG}_T(\mathbf{x}) \quad := \mathbb{H}\left[\mathcal{O}_\mathcal{A} | A_T\right] - \mathbb{E}_{f_\mathbf{x}|A_T}\left[\mathbb{H}\left[\mathcal{O}_\mathcal{A} | A_{T+1}\right]\right], \tag{2}$$

where $\mathbb{H}$ denotes the entropy, and $A_{T+1} := A_T \cup \{(\mathbf{x}, f_\mathbf{x})\}$ with $f_\mathbf{x}$ the (unrevealed) value of f at $\mathbf{x}$.

[1]To keep notation simple, we denote $c(\boldsymbol{\lambda})$ as a function of two arguments $(\boldsymbol{\lambda}_S, \boldsymbol{\lambda}_R)$ to differentiate components in the index set S from those in the complement $R = \{1, 2, ..., d\} \setminus S$. The integral shall be understood as a multiple integral of c where $\boldsymbol{\lambda}_j$, $j \in R$, are integrated out.

Neiswanger et al. (2021) propose an acquisition function to approximate the EIG in Eq. (2). In its simplest form, the algorithm output $\mathcal{O}_\mathcal{A}$ in the EIG is replaced by the algorithm's execution path $e_\mathcal{A}$, i.e., the sequence of all evaluations the algorithm $\mathcal{A}$ traverses, which thus gives full information about the output. The expected information gain estimated based on the execution path $e_\mathcal{A}$ is given by

$$\begin{aligned}\mathrm{EIG}^e_T(\mathbf{x}) &= \mathbb{H}\left[e_\mathcal{A}|A_T\right] - \mathbb{E}_{f_\mathbf{x}|A_T}\left[\mathbb{H}\left[e_\mathcal{A}|A_{T+1}\right]\right] \\ &= \mathbb{H}\left[f_\mathbf{x}|A_T\right] - \mathbb{E}_{e_\mathcal{A}|A_T}\left[\mathbb{H}\left[f_\mathbf{x}|A_T, e_\mathcal{A}\right]\right].\end{aligned} \tag{3}$$

where they used the symmetry of the mutual information to come up with the latter expression. The first term $\mathbb{H}\left[f_\mathbf{x}|A_T\right]$ is the entropy of the posterior predictive distribution at an input $\mathbf{x}$ and can be computed in closed form. The second term can be estimated as follows: A number of n_path samples $\tilde{f} \sim p(f \mid A_T)$ is drawn from the posterior process. The algorithm $\mathcal{A}$ is run on each of the samples $\tilde{f}$ to produce sample execution paths $\tilde{e}_\mathcal{A}$, yielding samples $\tilde{e}_\mathcal{A} \sim p(e_\mathcal{A} \mid A_T)$, used to estimate the second term as described by Neiswanger et al. (2021).

3 Related Work

Interpretability in AutoML refers either to (1) the interpretation of the resulting model returned by an AutoML system (Xanthopoulos et al., 2020; Binder et al., 2020; Carmichael et al., 2021; Coors et al., 2021), or (2) the interpretation of hyperparameter effects and importance (Moosbauer et al., 2021). We focus on the latter, specifically the construction of accurate and unbiased estimators for, e.g., hyperparameter effects in HPO.

There are HPO and AutoML frameworks that provide visualisations and interpretability statistics as additional outputs, e.g., *Google Vizier* (Golovin et al., 2017) and *xAutoML* (Zöller et al., 2022) provide an interactive dashboard visualizing the progress of the optimization and insights via parallel coordinate plots and multi-dimensional scaling on the optimizer footprint. Similarly, the HPO frameworks *optuna* (Akiba et al., 2019) or *scikit-optimize* (Head et al., 2022) allow for quick and simple visualization of optimization progress and results. However, such relatively simple visualizations do not give a deeper understanding of which hyperparameter influence model performance in what way.

In the context of HPO, practitioners are commonly interested on the marginal effects of hyperparameters on model performance Hutter et al. (2014); Young et al. (2018); Zela et al. (2018) or the importance of hyperparameters on model performance (Hutter et al., 2014; Biedenkapp et al., 2017; Van Rijn & Hutter, 2018; Probst et al., 2019). The latter is often directly derived from marginal effects of hyperparameters (Hutter et al., 2014). Established HPO frameworks (Head et al., 2022; Akiba et al., 2019) as well as visualization toolboxes Zöller et al. (2022) already make implementations of these methods accessible to users, however they neither discuss nor address a distortion of those arising due to a sampling bias. While all of these approaches have their merits, none of them address the imprecision in the estimates of these interpretive measures caused by sample bias that is present in the archive sampled by BO, since BO tends to exploit promising regions while leaving other regions unexplored. So far, only Moosbauer et al. (2021) explicitly proposed a post-hoc method that is able to identify subspaces of the configuration space in which accurate and unbiased PDPs can be computed. However, the method does not provide more accurate global IML estimates. To our knowledge, we are the first to propose a method that improves the sampling process of HPO to provide more accurate global estimates of such IML methods.

4 BOBAX: Enhanced Estimation of Interpretability Measures for HPO

We present our main contribution: BOBAX that efficiently searches for accurate marginal effect estimates of hyperparameters while maintaining competitive HPO performance.

4.1 Expected Information Gain for Partial Dependence

We first derive the information gained with regards to the estimate of a marginal effect of a hyperparameter $\boldsymbol{\lambda}_S$ *if* we observe performance $c_{\boldsymbol{\lambda}^{(T+1)}}$ for a hyperparameter configuration $\boldsymbol{\lambda}^{(T+1)}$. To this end, we quantify and analyze how a marginal effect is estimated in the context of HPO. Two types of approximations are performed: First, instead of estimating the marginal effect with regards to the *true*, but unknown and expensive objective c, we estimate the marginal effect of the surrogate model $\hat{c}$ [2], with $\hat{c}$ denoting the posterior mean of a probabilistic model $p(c \mid A_T)$. Secondly, we use the partial dependence method (Friedman, 2001) for efficient estimation of marginal effects of $\hat{c} : \Lambda \to \mathbb{R}$, which estimates Eq. (1) by Monte-Carlo sampling:

$$\varphi_{\boldsymbol{\lambda}_S} = \frac{1}{n} \sum_{i=1}^{n} \hat{c}\left(\boldsymbol{\lambda}_S, \boldsymbol{\lambda}_R^{(i)}\right), \tag{4}$$

with $\boldsymbol{\lambda}_S$ fixed and $\boldsymbol{\lambda}_R^{(i)} \overset{i.i.d.}{\sim} \mathbb{P}(\boldsymbol{\lambda}_R)$ a Monte-Carlo sample drawn from a uniform distribution $\mathbb{P}$. To bound the computational effort to compute the PDP, Eq. (4) is evaluated for a (typically equidistant) set of grid points $\{\boldsymbol{\lambda}_S^{(j)}\}_{j=1,...,G}$. The PDP visualizes $\varphi_{\boldsymbol{\lambda}_S}$ against $\boldsymbol{\lambda}_S$.

To define the expected information gain for partial dependence EIG_{PDP}, we have the partial dependence method in terms of a formal execution path (see also Algorithm 1): We iterate over all grid points, and compute the mean prediction $\hat{c}^{(g,i)}$. The execution path $e_{\mathcal{A}}$ thus corresponds to the Cartesian product $\left(\boldsymbol{\lambda}_S^{(g)}, \boldsymbol{\lambda}_R^{(i)}\right)$ for $g \in \{1, ..., G\}$ and $i \in \{1, ..., n\}$ of all grid points $\boldsymbol{\lambda}_S^{(g)}$ and the Monte-Carlo samples $\boldsymbol{\lambda}_R^{(i)}$.

As proposed by Neiswanger et al. (2021) as one variant, we estimate the information gained with regards to the execution path of $e_{\mathcal{A}}$ instead of estimating the execution path with regards to the algorithm output $O_{\mathcal{A}}$. Note that Neiswanger et al. (2021) argued that the criterion in Eq. (3) is in general suboptimal, if for example large parts of the execution path $e_{\mathcal{A}}$ do not have an influence on the algorithm output. We argue, however, that it is not applicable to our use-case since every element in the execution path of the PD method contributes with equal weight to the computation of the partial dependence. Figure 1 illustrates the computation of the PD based on the execution path, as well as the computation of the EIG_{PDP}.

[2] Constructed by BO, usually this will be the final surrogate model of the BO run, but this can also be applied interactively to intermediate models

Algorithm 1 PD algorithm with explicit execution path e_A

```
Input  $G$, $\hat{c}$, $\left(\boldsymbol{\lambda}_R^{(i)}\right) \overset{i.i.d.}{\sim} \mathbb{P}(\boldsymbol{\lambda}_R)$

$\left(\boldsymbol{\lambda}_S^{(1)}, ..., \boldsymbol{\lambda}_S^{(G)}\right) \leftarrow$ equidist. grid on $\Lambda_S$
for $g \in \{1, 2, ..., G\}$ do
    for $i \in \{1, 2, ..., n\}$ do
        $\boldsymbol{\lambda}^{(g,i)} \leftarrow \left(\boldsymbol{\lambda}_S^{(g)}, \boldsymbol{\lambda}_R^{(i)}\right)$
        $\hat{c}^{(g,i)} \leftarrow \hat{c}\left(\boldsymbol{\lambda}^{(g,i)}\right)$
        $e_\mathcal{A} \leftarrow e_\mathcal{A} \cup \left(\boldsymbol{\lambda}^{(g,i)}, \hat{c}^{(g,i)}\right)$
    end for
    $\varphi_{\boldsymbol{\lambda}_S^{(g)}} \leftarrow \frac{1}{n}\sum_{i=1}^{n} \hat{c}^{(g,i)}$
end for
Return  $\left(\boldsymbol{\lambda}_S^{(g)}, \varphi_{\boldsymbol{\lambda}_S^{(g)}}\right)$, $g = 1, ..., G$
```

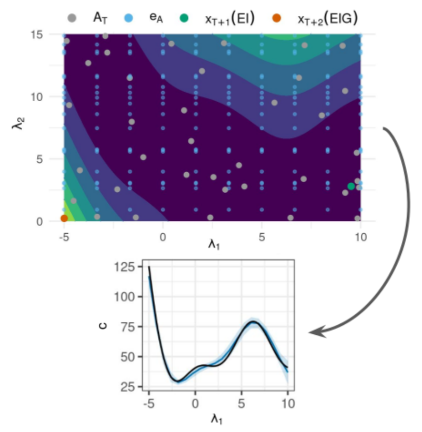


Figure 1: Shown are the elements of $e_\mathcal{A}$ (blue) PD method. The grey points show the configurations in the archive which are used by BO to construct the surrogate model. The green configuration is sampled by EI (showing more exploitation) while the orange point is the point maximizing the information gained about the PD estimate.

4.2 BOBAX: Efficient Optimization and Search for More Accurate Interpretability Measures

Given the EIG_PDP for PD, the optimization for interpretability of hyperparameter effects as part of BO is possible by using the EIG_PDP as acquisition function. However, interpretability alone is rarely of primary interest in practice; rather, the goal is to identify well-performing configurations and obtaining reasonable interpretations at the same time. We propose a method, dubbed BOBAX, that allows to efficiently search for explanations without a relevant loss of optimization efficiency.

BOBAX is an interleaving strategy which performs BO, and iterates between using the EI (or any other suited acquisition function) and the EIG_PDP as acquisition function. Although we have investigated also more complex variants (see Appendix B.2), interleaving EIG_PDP in every k-th iteration is simple yet efficient. The smaller k is, the higher is the weight of optimizing for accurate interpretations in a BO run. We note that this strategy can replace other interleaving exploration strategies, such as random samples (Hutter et al., 2011), since optimizing for interpretability can be seen as another strategy to cover the entire space in an efficient manner.[3]

From a practitioner's point of view, it may be reasonable to consider accuracy of interpretations rather as a constraint than an objective function to optimize for. As soon as this constraint is fulfilled, a user may want to invest all remaining budget into optimization only. Therefore, we also propose an adaptive variant of BOBAX, dubbed a-BOBAX, which performs the interleaving strategy in BOBAX as described above in a first phase, and transitions into optimization only in a second phase as soon as the constraint is fulfilled. To allow a user to input a meaningful constraint, the constraint must itself be interpretable by a user. Therefore, we define this constraint by a desired average width of confidence intervals around PD estimates, using the definition[4] of Moosbauer et al. (2021). As an example, a user may want to specify a tolerance $\pm 1\%$ in validation accuracy in estimation of PDs (see green tolerance bands in Figure 3 for illustration).

[3]One might have also considered addressing this as a multi-objective problem since we have two objectives: (i) finding the optimum and (ii) obtaining good PDPs. However, usually post-hoc multi-objective optimizers construct a Pareto front of a set of multiple candidate solution, which we are not interested in here. Instead, in each iteration of BO, the optimizer has to choose a concrete trade-off between both objectives. For dynamically balancing out this trade-off, please also refer to the next section.

[4]Confidence intervals are defined as $\varphi_{\boldsymbol{\lambda}_S^{(g)}} \pm q_{1-\alpha/2} \cdot \hat{s}_{\boldsymbol{\lambda}_S^{(g)}}$ around the PD estimate. $\hat{s}_{\boldsymbol{\lambda}_S^{(g)}}$ denotes the uncertainty of a PD estimate for a grid point g. As default, we look at $\alpha = 0.05$.

4.3 Theoretical and Practical Considerations

Runtime Complexity Since BOBAX comes with additional overhead, we discuss this here in more detail. The computation of the expectation requires posterior samples of the execution path $e_{\mathcal{A}} \sim p(e_{\mathcal{A}} \mid A_T)$. This is achieved by sampling from the posterior GP $\tilde{c} \sim p(c \mid A_T)$ and execution of $O_{\mathcal{A}}$ on those samples, which may produce a computational overhead depending on the costs of running $O_{\mathcal{A}}$. We assume that executing $O_{\mathcal{A}}$ is neglectable in terms of runtime. However, to compute the entropy $\mathbb{H}\left[c_{\boldsymbol{\lambda}}|A_T, e_{\mathcal{A}}\right]$, the posterior process needs to be trained based on $A_T \cup e_{\mathcal{A}}$ (which has size $T + n \cdot G$). Thus, the overall runtime complexity is dominated by $\mathcal{O}\left(n_{\text{path}} \cdot (T+n)^3\right)$, as we compute the entropy n_{path} times to approximate the expectation and since training a GP is cubic in the number of data points. Therefore, we recommend to keep an eye on the runtime overhead of the calculation of EIG_{PDP} in relation to evaluating c (e.g., training and evaluating an ML algorithm). Especially in the context of deep learning, the evaluation of a single configuration is usually by orders of magnitude higher than that of computing the EIG_{PDP}[5]. Also, we would like to emphasize that the implementation of our method is based on GPflow (Matthews et al., 2017), which allows fast execution of GPs on GPUs. Since GPUs are typically in use for training in the context of DL anyway, they can easily be leveraged in between iterations to speed up the computation of the EIG_{PDP}.

Marginal Effects for Multiple Hyperparameters Until now we have assumed that a user specifies a single hyperparameter of interest $\boldsymbol{\lambda}_S$ for which we will compute the PD. However, it is difficult to prioritize the hyperparameter of interest a-priori. Fortunately, it is possible to extend the execution path to compute EIG_{PDP} by the respective execution paths of the PDs with regards to all variables $e_{\mathcal{A}} = e_{\mathcal{A},\boldsymbol{\lambda}_1} \cup e_{\mathcal{A},\boldsymbol{\lambda}_2} \cup \ldots \cup e_{\mathcal{A},\boldsymbol{\lambda}_d}$. We investigate the differences between EIG_{PDP} for a single hyperparameter vs. for multiple hyperparameters in more detail in Appendix C; in the practical use-case (see Section 6), we compute the EIG_{PDP} for multiple hyperparameters.

5 Benchmark

In this section, we present experiments to demonstrate the validity of our method. In particular, we look at:

Hypothesis H1 Performing BO with EIG_{PDP} as acquisition function is more efficient than random search in optimizing for accurate interpretations

Hypothesis H2 Through BOBAX the accuracy of marginal effect estimates is clearly improved without a significant loss of optimization performance.

5.1 Experimental Setup

Objective Functions We apply our method to synthetic functions which are treated as black-box function during optimization: Branin ($d = 2$), Camelback ($d = 2$), Stylinski-Tang ($d = 3$), Hartmann3 ($d = 3$) and Hartmann6 ($d = 6$).

Algorithms To investigate H1, we consider BO with EIG_{PDP} as acquisition function (**BAX**). For H2, we consider BOBAX as described in Algorithm 2, where we iterate evenly ($k = 2$) between EI and EIG_{PDP} as acquisition function. Following Neiswanger et al. (2021) we set the number of execution path samples to 20 to approximate the expectation in Eq. (3) in both variants. As strong baseline for accurate PDs we consider random search (**RS**) and BO with posterior variance as acquistion function (**PVAR**) as a pure exploration case of LCB. As strong baseline for optimization we consider BO with EI (**BO-EI**). Further variants of our methods (e.g., different frequencies of interleaving) and additional baselines (such as BO with LCB with different exploration factors, or BO with EI and random interleaving) are described in Appendix C.

Evaluation We evaluate the accuracy of PD estimates by comparing the PD $\varphi_S^{(g)}$ (estimated based on $\hat{c}$) against the PD $\tilde{\varphi}_S^{(g)}$ computed on the ground-truth objective function c, approximated with the same sample

[5]In our case, the computation of the EIG_{PDP} was ranging from the order of a few seconds to a few minutes.

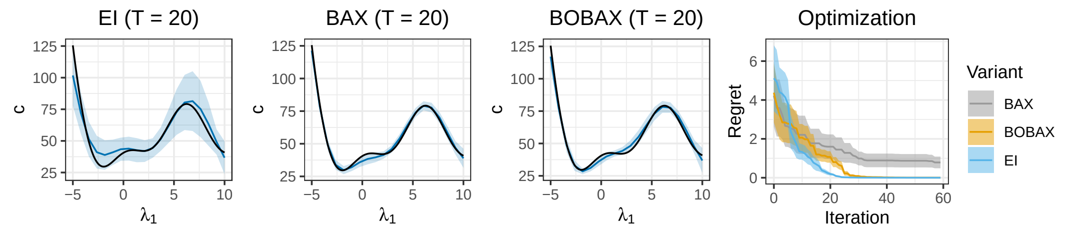


Figure 2: The first three plots show the estimated PD with 95% confidence interval (blue) based on the surrogate model $\hat{c}$ after $T = 30$ iterations vs. the true marginal effect (black). BAX and BOBAX yield more accurate estimates for the PD as compared the BO with EI. The right plot shows the cumulative regret for the three methods. BAX, which is not performing optimization at all, is also clearly outperformed in optimization performance. BOBAX reaches the optimization result of BO with EI only after a few more iterations.

$\boldsymbol{\lambda}_R^{(i)}$ and the same grid size G. As measure we use the L_1 distance $\mathrm{d}_{\mathrm{L}_1} := \frac{1}{G}\sum_{g=1}^{G}\left|\varphi_S^{(g)} - \tilde{\varphi}_S^{(g)}\right|$ averaged over all grid points. To assess optimization performance, we report the simple regret $c(\hat{\boldsymbol{\lambda}}) - c(\boldsymbol{\lambda}^*)$, where $\boldsymbol{\lambda}^*$ denotes the theoretical optimum of a function, and $\hat{\boldsymbol{\lambda}} \in \operatorname{argmin}\{c_{\boldsymbol{\lambda}} \mid (\boldsymbol{\lambda}, c_{\boldsymbol{\lambda}}) \in A_T\}$ is the best found configuration during optimization.

Further Configurations A Gaussian process with a squared exponential kernel is used as surrogate model for all BO variants, and PDs are estimated on the respective surrogate models. For RS, a GP (with same configuration) is fitted on A_T and the PD is computed thereon. Acquisition function optimization is performed by randomly sampling 1500 configurations, evaluating the respective acquisition function and returning the best. Each (BO) run is given a maximum number of $30 \cdot d$ function evaluations.

Reproducibility and Open Science The implementation of methods as well as reproducible scripts for all experiments are publicly made available. Each experiment is replicated 20 times based 20 different seeds fixed across all variants. More details on the code and on computational can be found in Appendix E.

5.2 H1: More accurate interpretations

Our experiments support hypothesis H1, i.e., we can achieve more accurate PD estimates more efficiently through targeted sampling via the $\mathrm{EIG}_{\mathrm{PDP}}$. An example run on the Branin function shown in Figure 2 illustrates the behavior of the methods that is observable across all experiments: BAX is yielding clearly more accurate PDPs than BO with EI already after few iterations. Figure 4 in Appendix C.2 supports that PDs estimated on data produced by BO with EI might provide not only quantitatively, but also qualitatively wrong information in terms of ranking the values $\varphi_{\boldsymbol{\lambda}_S^{(g)}}$ differently than the ground-truth. As expected, increased accuracy of interpretations through BAX comes to the cost of optimization efficiency. Results aggregated across all problems and replications confirm this behavior on a broader scale, see Table 1[6]. BAX is producing more accurate PDPs than RS (which can be assumed to converge against the true marginal effect) already at early stages, and is strongly significantly ($\alpha = 1\%$) outperforming RS with less iterations. We conclude that both BAX and PVAR can contribute to approximating the true marginal effect well, but BAX is converging faster. In addition, BO with EI is significantly outperformed in terms of accuracy of PDPs, which supports our assumption of lowered quality caused through a heavy sampling bias.

[6]We note that the different functions live on different scales s.t. we normalized it by showing relative metrics wrt baselines, such RS for PDP estimates and EI for optimization regret.

Table 1: **Left**: $L1$ error of the estimated PDP w.r.t. the ground truth PDP, relative to RS as baseline. Negative values mean a relative reduction of the $L1$ error compared to random search. **Right:** Optimization relative to BO-EI as baseline. Results are averaged across all 20 replications. Best values are bold, and values are underlined if not significantly worse than the best based on a Post-Hoc Friedman test ($\alpha = 1\%$), see also Demsar (2006); García et al. (2010) and Appendix C.1 for more details.

	Relative d_{L_1}(PDP) after					Relative optimization regret after			
	25%	50%	75%	100%		25%	50%	75%	100%
	Max. iterations spent					Max. iterations spent			
RS	0.00	0.00	0.00	<u>0.00</u>	RS	2.42	160.99	530.70	951.47
BO-EI	0.18	0.39	0.47	0.67	BO-EI	**<u>0.00</u>**	**<u>0.00</u>**	**<u>0.00</u>**	**<u>0.00</u>**
PVAR	0.13	<u>-0.08</u>	<u>0.08</u>	<u>0.14</u>	PVAR	3.38	232.14	741.69	1887.22
BAX	**<u>-0.17</u>**	**<u>-0.20</u>**	**<u>-0.07</u>**	**<u>0.00</u>**	BAX	2.27	242.062	602.15	1408.62
BOBAX	<u>-0.14</u>	<u>-0.16</u>	<u>-0.04</u>	<u>0.03</u>	BOBAX	1.68	<u>5.04</u>	<u>4.73</u>	<u>3.26</u>

5.3 H2: More accurate interpretations at no relevant loss of optimization efficiency

Our experiments also support hypothesis H2, i.e., with BOBAX we can achieve clearly more accurate PD estimates while maintaining a competitive level of optimization efficiency. Table 1 compares the accuracy of PD estimates (measured via d_{L_1}) and optimization regret as compared to baselines RS and BO-EI, respectively, aggregated over all five objective functions. (BO)BAX allows for more accurate PDPs than the other methods, with diminishing relative distance to RS, while BO with EI is clearly outperformed. On the other hand, it can be observed that BOBAX is giving optimization performance comparable to BO with EI throughout the course of optimization, whereas RS is clearly outperformed. So, BOBAX combines the best of both worlds: good interpretability (even better than RS) and efficient optimization (on par with BO-EI). Figure 5 in Appendix C.2 shows that this effect is visible for all objective function, but the strength of the effect depends on the objective functions.

We conclude that our experiments support that BOBAX makes no (or only little) compromises in optimization performance, but yields clearly better estimates of marginal effects at the same time.

6 Practical HPO Application

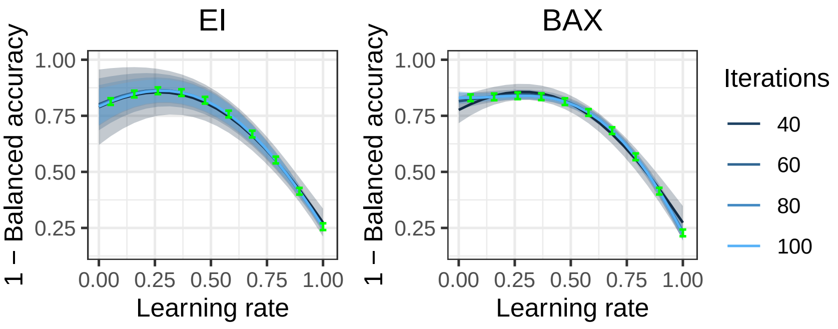


Figure 3: Comparing PDP evolution for number of iterations for EI and BAX. BAX returns fairly certain PDPs early on, whereas BO with EI requires much more time.

We demonstrate a-BOBAX on a concrete HPO scenario, following the setup of Moosbauer et al. (2021). We tune common hyperparameters of a neural network with regards to balanced validation accuracy on 15 different datasets respresenting different tasks from different domains (see Tables 5, 6 in Appendix D)

Table 2: Iterations needed to reach the desired precision of PD estimate of ±1.5 balanced accuracy points, accuracy of the final PD based on the L1 error to the ground truth, as well as the final model performance reached. Results are averaged across all 30 replications and all 15 datasets. Best values are bold, and values are underlined if not significantly worse than the best based on a Post-Hoc Friedman test ($\alpha = 1\%$), see also Demsar (2006); García et al. (2010) and Appendix C.1 for more details.

	Iterations to desired precision	Rel. d_{L_1} (PDP)	1 - Balanced Accuracy
RS	14.91	<u>**0.49**</u>	22.56
BO-EI	22.59	0.57	<u>**19.38**</u>
BAX	9.85	0.51	23.97
a-BOBAX	**11.56**	<u>0.52</u>	<u>19.94</u>

using the interface provided by YAHPO gym (Pfisterer et al., 2021). We compare RS, EI, BAX, and adaptive BOBAX (a-BOBAX). In a-BOBAX, we set the desired width of confidence intervals to ±1.5% balanced accuracy points; we emphasize thought, that this value can be set by the user. For a-BOBAX, we compute the EIG_{PDP} jointly for the PDPs of *learning rate*, *dropout*, *max. number of units*, *weight decay*, and *momentum*. The respective methods ran under the same conditions as in Section 5, but were replicated 30 times.

Figure 3 shows how accuracy of the PD estimate increases over time for BO with EI vs. BAX. We observe that BAX is clearly more efficient in returning an accurate estimate, which is in line with the results we observed in Section 5. As motivated in Section 4.2, a practitioner might prefer to rather ensure a minimum accuracy of IML measures, and therefore, handle this rather as a constraint than as an objective. Table 2 is showing the time to reach the desired precision of ±1.5% for the PDP, as well as final accuracy of PDs and final optimization performance, aggregated over all experiments an replications. We observe that a-BOBAX is (i) significantly faster in reaching the desired precision threshold, allowing a user to interact earlier with confidence, (ii) is comparable to RS in terms of final accurate representation of PDs, and (iii) comparable to BO-EI in terms of optimization performance. Note that the effect again depends strongly on the respective dataset (see Figures 7, 8, 9 in Appendix D.2).

7 Discussion and Conclusion

Findings We proposed (adaptive) BOBAX, modifying Bayesian Optimization (BO) for black-box optimization and HPO to enhance interpretability of the optimization problem at hand. We achieved this by adapting BAX to optimize for accurate marginal effects and then interleaved BO and BAX. We further showed that BOBAX can significantly enhance the accuracy of PD estimates during an optimization procedure, while not losing optimization performance.

Usage If a user has some desired precision of the IML estimates in mind, a-BOBAX allows them to make use of BAX only until this level is not reached yet and will focus on the optimization quality afterwards. This simple, yet efficient strategy allows to get the most out of the overall budget.

Critical View and Limitations Even though the usage of EIG is beneficial to the quality of a PD estimate, there are also examples where no significant improvement is observed. We assume that this particularly holds for hyperparameters that have a simple (and therefore easy-to-learn) effect on performance. Consequently, the marginal effect is easily learned for any of the methods. In addition to using the adaptive version of BOBAX, we recommend dropping these simple-to-learn hyperparameters from the joint computation of the EIG (4.3) as soon as the PDPs are sufficiently certain. Furthermore, our method comes at a computational overhead, being slightly larger than traditional BO since computing EIG with BAX costs a bit more compute time. In terms of application to HPO, we expect that the cost for training and validating hyperparameter configurations or architectures of neural networks will be much larger than BOBAX's overhead in most relevant cases.

Outlook We believe that BOBAX will contribute in particular towards more human-centered HPO, where developers can start inspecting intermediate results as soon as desired confidence was reached and then adapt the configuration space if necessary. Although we focused on PDPs as an interpretability method, extending our BOBAX idea to other IML approaches would be straightforward and opens up new follow up directions. As one next step, we envision extending BOBAX to the multi-fidelity setting (Li et al., 2017; Falkner et al., 2018) which is required for more expensive HPO and AutoML problems. Last but not least, we emphasize that we developed BOBAX primarily for HPO problems, but it can also be applied to any black-box optimization problem, e.g., in engineering or chemistry.

References

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.

James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, 24, 2011.

Andre Biedenkapp, Marius Lindauer, Katharina Eggensperger, Frank Hutter, Chris Fawcett, and Holger H. Hoos. Efficient parameter importance analysis via ablation with surrogates. In Satinder Singh and Shaul Markovitch (eds.), *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pp. 773–779. AAAI Press, 2017.

Martin Binder, Julia Moosbauer, Janek Thomas, and Bernd Bischl. Multi-objective hyperparameter tuning and feature selection using filter ensembles. In Carlos Artemio Coello Coello (ed.), *GECCO '20: Genetic and Evolutionary Computation Conference, Cancún Mexico, July 8-12, 2020*, pp. 471–479. ACM, 2020.

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. *CoRR*, abs/2107.05847, 2021.

Zachariah Carmichael, Tim Moon, and Sam Ade Jacobs. Learning interpretable models through multi-objective neural architecture search. *CoRR*, abs/2112.08645, 2021.

Stefan Coors, Daniel Schalk, Bernd Bischl, and David Rügamer. Automatic componentwise boosting: An interpretable automl system. *CoRR*, abs/2109.05583, 2021.

D Richard Cutler, Thomas C Edwards Jr, Karen H Beard, Adele Cutler, Kyle T Hess, Jacob Gibson, and Joshua J Lawler. Random forests for classification in ecology. *Ecology*, 88(11):2783–2792, 2007.

Janez Demsar. Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.*, 7:1–30, 2006.

Jaimie Drozdal, Justin Weisz, Dakuo Wang, Gaurav Dass, Bingsheng Yao, Changruo Zhao, Michael Muller, Lin Ju, and Hui Su. Trust in AutoML. *Proceedings of the 25th International Conference on Intelligent User Interfaces*, Mar 2020.

Stefan Falkner, Aaron Klein, and Frank Hutter. BOHB: robust and efficient hyperparameter optimization at scale. In Jennifer G. Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1436–1445. PMLR, 2018.

Aaron Fisher, Cynthia Rudin, and Francesca Dominici. All models are wrong, but many are useful: Learning a variable's importance by studying an entire class of prediction models simultaneously. *J. Mach. Learn. Res.*, 20:177:1–177:81, 2019.

Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pp. 1189–1232, 2001.

Salvador García, Alberto Fernández, Julián Luengo, and Francisco Herrera. Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power. *Inf. Sci.*, 180(10):2044–2064, 2010.

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, Halifax, NS, Canada, August 13 - 17, 2017*, pp. 1487–1495. ACM, 2017.

Niklas Hasebrook, Felix Morsbach, Niclas Kannengießer, Jörg K. H. Franke, Frank Hutter, and Ali Sunyaev. Why do machine learning practitioners still use manual tuning? A qualitative study. *CoRR*, abs/2203.01717, 2022.

Tim Head, Manoj Kumar, Holger Nahrstaedt, Gilles Louppe, and Iaroslav Shcherbatyi. scikit-optimize/scikit-optimize, April 2022. URL `https://doi.org/10.5281/zenodo.6451894`.

Philipp Hennig and Christian J. Schuler. Entropy search for information-efficient global optimization. *J. Mach. Learn. Res.*, 13:1809–1837, 2012.

José Miguel Hernández-Lobato, Matthew W. Hoffman, and Zoubin Ghahramani. Predictive entropy search for efficient global optimization of black-box functions. In Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger (eds.), *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pp. 918–926, 2014.

Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In Carlos A. Coello Coello (ed.), *Learning and Intelligent Optimization - 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers*, volume 6683 of *Lecture Notes in Computer Science*, pp. 507–523. Springer, 2011.

Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. An efficient approach for assessing hyperparameter importance. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, volume 32 of *JMLR Workshop and Conference Proceedings*, pp. 754–762. JMLR.org, 2014.

Carl Hvarfner, Danny Stoll, Artur L. F. Souza, Marius Lindauer, Frank Hutter, and Luigi Nardi. πbo: Augmenting acquisition functions with user beliefs for bayesian optimization. *CoRR*, abs/2204.11051, 2022. doi: 10.48550/arXiv.2204.11051. URL `https://doi.org/10.48550/arXiv.2204.11051`.

Donald R. Jones. A taxonomy of global optimization methods based on response surfaces. *Journal of Global Optimization*, 21(4):345–383, 2001.

Donald R. Jones, Matthias Schonlau, and William J. Welch. Efficient global optimization of expensive black-box functions. *J. Glob. Optim.*, 13(4):455–492, 1998.

Aurélie Lemmens and Christophe Croux. Bagging and boosting classification trees to predict churn. *Journal of Marketing Research*, 43(2):276–286, 2006.

Lisha Li, Kevin G. Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *J. Mach. Learn. Res.*, 18:185:1–185:52, 2017.

D. V. Lindley. On a measure of the information provided by an experiment. *The Annals of Mathematical Statistics*, 27(4):986–1005, 1956. ISSN 00034851.

Alexander G. de G. Matthews, Mark van der Wilk, Tom Nickson, Keisuke. Fujii, Alexis Boukouvalas, Pablo León-Villagrá, Zoubin Ghahramani, and James Hensman. GPflow: A Gaussian process library using TensorFlow. *Journal of Machine Learning Research*, 18(40):1–6, apr 2017.

Julia Moosbauer, Julia Herbinger, Giuseppe Casalicchio, Marius Lindauer, and Bernd Bischl. Explaining hyperparameter optimization via partial dependence plots. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 2280–2291. Curran Associates, Inc., 2021.

Willie Neiswanger, Ke Alexander Wang, and Stefano Ermon. Bayesian algorithm execution: Estimating computable properties of black-box functions using mutual information. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pp. 8005–8015. PMLR, 2021.

Randal S. Olson and Jason H. Moore. TPOT: A tree-based pipeline optimization tool for automating machine learning. In *Proceedings of the 2016 Workshop on Automatic Machine Learning*, volume 64 of *JMLR Workshop and Conference Proceedings*, pp. 66–74. JMLR.org, 2016.

Florian Pfisterer, Lennart Schneider, Julia Moosbauer, Martin Binder, and Bernd Bischl. YAHPO gym - design criteria and a new multifidelity benchmark for hyperparameter optimization. *CoRR*, abs/2109.03670, 2021.

Philipp Probst, Anne-Laure Boulesteix, and Bernd Bischl. Tunability: Importance of hyperparameters of machine learning algorithms. *Journal of Machine Learning Research*, 20:53:1–53:32, 2019.

Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems 25*, pp. 2960–2968, 2012.

Niranjan Srinivas, Andreas Krause, Sham M. Kakade, and Matthias W. Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. In Johannes Fürnkranz and Thorsten Joachims (eds.), *Proceedings of the 27th International Conference on Machine Learning*, pp. 1015–1022. Omnipress, 2010.

Chris Thornton, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Auto-weka: combined selection and hyperparameter optimization of classification algorithms. In *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 847–855. ACM, 2013.

Ryan Turner, David Eriksson, Michael McCourt, Juha Kiili, Eero Laaksonen, Zhen Xu, and Isabelle 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*, volume 133 of *Proceedings of Machine Learning Research*, pp. 3–26. PMLR, 2020.

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

Seth J Wenger and Julian D Olden. Assessing transferability of ecological models: an underappreciated aspect of statistical validation. *Methods in Ecology and Evolution*, 3(2):260–267, 2012.

Jian Wu, Matthias Poloczek, Andrew Gordon Wilson, and Peter I. Frazier. Bayesian optimization with gradients. In *Advances in Neural Information Processing Systems 30*, pp. 5267–5278, 2017.

Iordanis Xanthopoulos, Ioannis Tsamardinos, Vassilis Christophides, Eric Simon, and Alejandro Salinger. Putting the human back in the AutoML loop. In *Proceedings of the Workshops of the EDBT/ICDT 2020 Joint Conference*, volume 2578 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2020.

M. Todd Young, Jacob D. Hinkle, Arvind Ramanathan, and Ramakrishnan Kannan. Hyperspace: Distributed bayesian hyperparameter optimization. In *30th International Symposium on Computer Architecture and High Performance Computing, SBAC-PAD 2018, Lyon, France, September 24-27, 2018*, pp. 339–347. IEEE, 2018. doi: 10.1109/CAHPC.2018.8645954. URL `https://doi.org/10.1109/CAHPC.2018.8645954`.

Arber Zela, Aaron Klein, Stefan Falkner, and Frank Hutter. Towards automated deep learning: Efficient joint neural architecture and hyperparameter search. *CoRR*, abs/1807.06906, 2018. URL `http://arxiv.org/abs/1807.06906`.

Zhongheng Zhang, Marcus W Beck, David A Winkler, Bin Huang, Wilbert Sibanda, Hemant Goyal, et al. Opening the black box of neural networks: methods for interpreting neural network models in clinical applications. *Annals of translational medicine*, 6(11), 2018.

Lucas Zimmer, Marius Lindauer, and Frank Hutter. Auto-PyTorch Tabular: Multi-fidelity metalearning for efficient and robust AutoDL. *IEEE TPAMI*, 2021. Preprint via Early Access.

Marc-André Zöller, Waldemar Titov, Thomas Schlegel, and Marco F. Huber. Xautoml: A visual analytics tool for establishing trust in automated machine learning. *CoRR*, abs/2202.11954, 2022.

6. Contributions to explainability of the AutoML system

6.1. Automated Benchmark-Driven Design and Explanation of Hyperparameter Optimizers

The work unifies the most common multi-fidelity HPO optimizers in a configurable framework. Following the programming-by-optimization optimization principle, a search space comprising a multitude of HPO optimizers is searched over efficiently by BO, and an ablation analysis is performed to challenge whether design choices are critical to performance or could be replaced by simpler design choices. This work presents a principled approach to the explanation of hyperparameter optimizers and contributes to gaining a more holistic understanding of which algorithmic components are driving performance. It can thus be classified methodologically as a contribution to Section 3.3 – *Sensitivity analysis of AutoML tools (B).*

Contributing article:

Moosbauer, J., Binder, M., Schneider, L., Pfisterer, F., Becker, M., Lang, M., Kotthoff, L., Bischl., B. (2022). Automated Benchmark-Driven Design and Explanation of Hyperparameter Optimizers. *IEEE Transactions on Evolutionary Computation, vol. 26, no. 6, pp. 1336–1350.*

Author contributions:

The idea for the algorithmic framework (SMASHY) considered in the project originated from Martin Binder and Bernd Bischl, with input from Julia Moosbauer. The idea for putting it into context as automatic configuration and analysis of algorithm components for hyperparameter optimizers originated from Julia Moosbauer and Martin Binder. Code for the algorithmic framework was developed by Martin Binder based on a previous project[1]. Code for the experiments for algorithm configuration was written by Marc Becker and Martin Binder. Benchmarks of competing state-of-the-art algorithm implementations were conducted by Julia Moosbauer. Code for the experiments for algorithm analysis was written by Lennart Schneider (ablation studies), Martin Binder, and Julia Moosbauer. The manuscript was written by Julia Moosbauer and Martin Binder, with refinement from Lennart Schneider, Lars Kotthoff, Marc Becker, Florian Pfisterer, and Bernd Bischl. Lennart Schneider, Martin Binder, Florian Pfisterer, Lennart Schneider, Michel Lag, and Lars Kotthoff advised throughout the whole project.

[1]`https://github.com/mlr-org/miesmuschel`

Supplementary material available at:

- Supplementary material:
  `https://ieeexplore.ieee.org/document/9913342/media#media`

1336 IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, VOL. 26, NO. 6, DECEMBER 2022

Automated Benchmark-Driven Design and Explanation of Hyperparameter Optimizers

Julia Moosbauer, Martin Binder, Lennart Schneider, Florian Pfisterer, Marc Becker, Michel Lang, Lars Kotthoff, and 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 multifidelity 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 analysis, algorithm design, automated machine learning (AutoML), hyperparameter optimization (HPO), multifidelity.**

Manuscript received 21 September 2021; revised 15 March 2022 and 28 July 2022; accepted 15 September 2022. Date of publication 6 October 2022; date of current version 1 December 2022. The work of Lennart Schneider was 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 under Grant 20-3410-2-9-8. The work of Michel Lang was supported in part by the Research Center Trustworthy Data Science and Security (https://rc-trust.ai). The work of Lars Kotthoff was supported by NSF under Grant 1813537. *(Julia Moosbauer and Martin Binder contributed equally to this work.) (Corresponding author: Julia Moosbauer.)*

Julia Moosbauer, Martin Binder, Lennart Schneider, Florian Pfisterer, Marc Becker, and Bernd Bischl are with the Department of Statistics, Ludwig-Maximilians-Universität München, 80539 Munich, Germany (e-mail: julia.moosbauer@stat.uni-muenchen.de; martin.binder@stat.uni-muenchen.de; lennart.schneider@stat.uni-muenchen.de; florian.pfisterer@stat.uni-muenchen.de; marc.becker@stat.uni-muenchen.de; bernd.bischl@stat.uni-muenchen.de).

Michel Lang is with the Research Center Trustworthy Data Science and Security, 44227 Dortmund, Germany (e-mail: michel.lang@tu-dortmund.de).

Lars Kotthoff is with the Department of Computer Science, University of Wyoming, Laramie, WY 82071 USA (e-mail: larsko@uwyo.edu).

This article has supplementary downloadable material available at https://doi.org/10.1109/TEVC.2022.3211336, provided by the authors.

Digital Object Identifier 10.1109/TEVC.2022.3211336

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 HPO (CASH) problem [2]. Successfully addressing this problem can lead to large performance gains compared to simply using defaults, and in the context of automated ML (AutoML), the use of HPO can make ML more accessible to nonexperts. 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.

1) *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.
2) *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].
3) *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].
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].
5) *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 the 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 nonlinear 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], [11], [12], and [13]).

Multifidelity 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 tradeoff 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 HB (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.

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 multifidelity algorithms, for example, are extensions and further developments of HB that take the fixed successive halving (SH) 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 as follows.

1) *Formalization:* We formalize the design space of MF-HPO algorithms and demonstrate that established MF-HPO algorithms represent instances within this space.
2) *Framework:* Based on this formalization, we present a rich, configurable framework for MF-HPO algorithms, whose software implementation we call surrogate model-assisted HB (SMASHY).
3) *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.
4) *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.
5) *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 toward such systematic development have often relied on a *high-level* language or template that allows expressing solution algorithms for a given problem class, e.g., to solve constraint satisfaction problems [21], [22], [23], satisfiability problems [24], scheduling problems [25], or general multiobjective combinatorial problems [26], [27].

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, iterated F-RACE [28] has been used for the automatic design of multiobjective ant colony optimization algorithms [26]. Similarly, IRACE [29] has been used for the automatic design multiobjective evolutionary algorithms [27] 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. While such intensification mechanisms have been used in other work before [31], [32], SMAC also uses 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. Besides racing and sequential model-based approaches, genetic algorithms have also been used to evolve optimal solvers [33].

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 [34]), we are not aware of many cases where PBO is applied to designing HPO systems themselves. One exception is [35], who use SMACv3 [36] to automatically configure 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 [35], 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 multifidelity handling (e.g., SH) 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 [37]. 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 [38]. This class of methods has also become popular in the field of ML to analyze the importance of hyperparameters [39].

Popular ways of analyzing effects of algorithmic effects in ML and algorithm configuration are *ablation studies* [40]. 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 [41]. In the context of algorithm configuration, Fawcett and Hoos [40] proposed 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, Minton [21] compared the designs and their approach finds automatically to the designs expert humans generated. López-Ibáñez and Stützle [42] performed ANOVA and nonparametric Friedman tests to investigate in detail the effects that algorithmic choices, found through automatic configuration [26], have on the performance of multiobjective ant colony optimization algorithms. de Nobel et al. [43] presented a modular framework for CMA-ES variants on which they perform optimization; in particular, they investigate how the optimized configuration changes when the search space is enlarged by introducing new components.

III. Methodology

A. Supervised Machine Learning

Supervised ML typically deals with a dataset (which is, mathematically speaking, a tuple) $\mathcal{D} = ((\mathbf{x}^{(i)}, y^{(i)})) \in (\mathcal{X} \times \mathcal{Y})^n$ 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 optimization problems that we will define in Sections III-B and III-C, we term this the "first-level" optimization problem.

The expectation of the loss value of predictions made for data samples drawn from $\mathbb{P}_{xy}$ is the *generalization error*

$$\mathrm{GE} := \mathbb{E}_{(\mathbf{x},y)\sim\mathbb{P}_{xy}}\left[L\left(y, \hat{f}(x)\right)\right] \quad (1)$$

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

[1] where g allows handling of multioutput regression, as well as multiclass classification with g classes by returning decision scores.

subsets to obtain an estimate of the generalization error

$$\widehat{\mathrm{GE}}(\mathcal{I}, \boldsymbol{\lambda}, \mathbf{J}) = \frac{1}{N_{\mathrm{iter}}} \sum_{j=1}^{N_{\mathrm{iter}}} L\big(y[-J_j], \mathcal{I}\big(\mathcal{D}[J_j], \boldsymbol{\lambda}\big)\big(x[-J_j]\big)\big). \quad (2)$$

Depending on the resampling method, the inducer $\mathcal{I}$, and the quantity of data in $\mathcal{D}$, estimating the generalization error $\widehat{\mathrm{GE}}(\mathcal{I}, \boldsymbol{\lambda}, \mathbf{J})$ 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 (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

$$\boldsymbol{\lambda}_S^* \in \underset{\boldsymbol{\lambda}_S \in \Lambda_S}{\operatorname{argmin}}\ c(\boldsymbol{\lambda}_S) = \underset{\boldsymbol{\lambda}_S \in \Lambda_S}{\operatorname{argmin}} \widehat{\mathrm{GE}}(\mathcal{I}, (\boldsymbol{\lambda}_S, \boldsymbol{\lambda}_C), \mathbf{J}). \quad (3)$$

Here, $\boldsymbol{\lambda}_S^*$ denotes a theoretical optimum, and $c(\boldsymbol{\lambda}_S)$ is a shorthand for the estimated generalization error in (2). We refer to Problem 3 as the "second-level" optimization 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 tradeoffs: 1) the exploration versus exploitation tradeoff 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., RS) and 2) the inference versus search tradeoff 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 versus 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; *multifidelity*, i.e., if the optimizer uses cheaper evaluations to infer performance on the full data.

Multifidelity methods make use of the fact that the resampling procedure in (2) can be modified in multiple ways to make evaluation cheaper: one can 1) reduce the training sizes $|J_j|$ via subsampling, as model evaluation complexity is often at least linear in training set size or 2) change some components in $\boldsymbol{\lambda}$ in a way that makes model fits cheaper. Examples of 2) 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 $\widehat{\mathrm{GE}}$ 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 $\widehat{\mathrm{GE}}$ and define

$$c(\boldsymbol{\lambda}_S; r) := \widehat{\mathrm{GE}}(\mathcal{I}, (\boldsymbol{\lambda}_S, \boldsymbol{\lambda}_C(r)), \mathbf{J}(r)). \quad (4)$$

With this definition, we make the choice that r should influence the evaluation cost of $\widehat{\mathrm{GE}}$ 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 [44], and the optimization problem remains as in (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 $(\boldsymbol{\lambda}_S^*, r^*) \in \operatorname{argmin}_{\boldsymbol{\lambda}_S \in \Lambda_S, r \in (0,1]} c(\boldsymbol{\lambda}_S; r)$.

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 predetermined budget is exhausted, such an algorithm decides in every iteration 1) which configuration(s) $\boldsymbol{\lambda}_S$ to evaluate next and 2) which fidelity r to use for evaluation; nonmultifidelity 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.

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 are proposed in line 2 and evaluated

Algorithm 1 Generic HPO Algorithm

```
1: while budget is not exhausted do
2:     Propose $(\boldsymbol{\lambda}_S^{(i)}, r^{(i)})$, $i = 1, ..., k$, based on archive $\mathcal{A}$
3:     Write proposals into a shared archive $\mathcal{A}$
4:     Estimate generalization error(s) $c(\boldsymbol{\lambda}_S^{(i)}; r^{(i)})$
5:     Write results into shared archive $\mathcal{A}$
6: end while
7: Wait for workers to synchronize
8: Return best configuration in archive $\mathcal{A}$
```

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], [45]. 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 $A : \Omega \times \Gamma \to \Lambda$ parameterized 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 $\mathbb{P}_\Omega$

$$\boldsymbol{\gamma}^* \in \operatorname*{argmin}_{\boldsymbol{\gamma}\in\Gamma} \mathbb{E}_{\omega\sim\mathbb{P}_\Omega}\left[\zeta(A(\omega, \boldsymbol{\gamma}))\right]. \quad (5)$$

In our example, Γ corresponds to the space of possible components of our HPO method and Ω corresponds 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.

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( $\mathcal{A}, \mu(b), r; \mathcal{I}_{f_{\text{sur}}}, \mathbb{P}_{\boldsymbol{\lambda}}(\mathcal{A}),$
                   $\rho(t), \left(N_s^0(t), N_s^1(t)\right), n_{\text{trn}}$)
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($\mathcal{A}, \tilde{\mu}, r; \mathcal{I}_{f_{\text{sur}}}, \mathbb{P}_{\boldsymbol{\lambda}}(\mathcal{A}),$
                       $\rho(t), \left(N_s^0(t), N_s^1(t)\right), n_{\text{trn}}$)
14:        end if
15:    end if
16:    Evaluate configuration(s) $c(\boldsymbol{\lambda}_S; r)$ 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
```

IV. Formalizing Broad Class of MF-HPO Algorithms

We aim to find an HPO algorithm that performs particularly well in the multifidelity setting. To design an algorithm by optimization, we propose a framework and search space of HPO algorithm candidates that cover a large class of possible algorithms and focus on a subclass of algorithms similar to HB because of their favorable properties. This subclass focuses on multifidelity algorithms that use a predefined schedule of geometrically increasing fidelity evaluations containing algorithms like HB [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: 1) sampling new configurations at low fidelities (lines 2–7) and 2) 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.

TABLE I
RS, BO, SH, HB, AND BOHB AS INSTANCES OF ALGORITHM 2. η, ρ, AND 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 NONSTANDARD 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	$\lfloor -\log_\eta(r_{\text{min}}) \rfloor + 1$	η	η	—	1	—	SH	uniform
HB	$\lceil s \cdot \frac{\eta^{s-b}}{s-b+1} \rceil$	$\lfloor -\log_\eta(r_{\text{min}}) \rfloor + 1$	η	η	—	1	—	HB	uniform
BOHB	$\lceil s \cdot \frac{\eta^{s-b}}{s-b+1} \rceil$	$\lfloor -\log_\eta(r_{\text{min}}) \rfloor + 1$	η	η	TPE*	ρ	N_{s}	HB	KDE†

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 nondiscarded 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 HB, by setting $\rho = 1$. The influence of the surrogate model on sampled candidates is larger when 1) the number of sampled configurations N_s is large or 2) 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 in the supplementary material). 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 SH [19] and HB [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 SH. The `HB` mode evaluates brackets, as performed by HB. 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 HB, 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 Fig. 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 $(N_s^0(t), N_s^1(t))$ 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 $N_s^{\cdot}(t)$ perform better when t is small. Conversely, it is likely that small $\rho(t)$/large $N_s^{\cdot}(t)$ 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: 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 [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

(a)

	$b=1$		$b=2$		$b=3$		$b=4$	
i	$\lvert C\rvert$	r	$\lvert C\rvert$	r	$\lvert C\rvert$	r	$\lvert C\rvert$	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	$\lvert C\rvert$	r
1	8	1/8
2	8	1/4
3	8	1/2
4	8	1

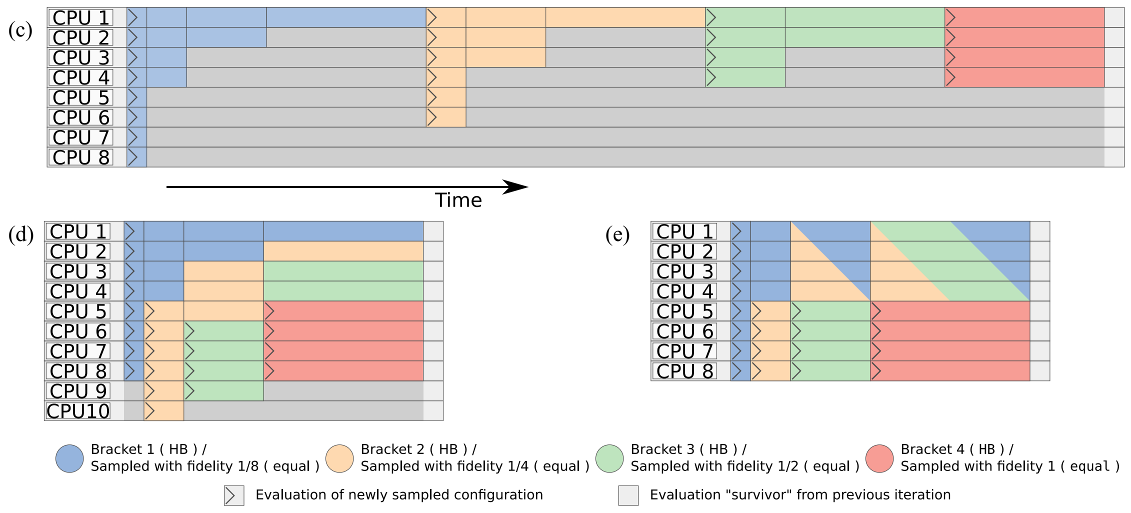

(c)
CPU 1
CPU 2
CPU 3
CPU 4
CPU 5
CPU 6
CPU 7
CPU 8
Time
(d)
CPU 1
CPU 2
CPU 3
CPU 4
CPU 5
CPU 6
CPU 7
CPU 8
CPU 9
CPU10
(e)
CPU 1
CPU 2
CPU 3
CPU 4
CPU 5
CPU 6
CPU 7
CPU 8
Bracket 1 (HB) / Sampled with fidelity 1/8 (equal)
Bracket 2 (HB) / Sampled with fidelity 1/4 (equal)
Bracket 3 (HB) / Sampled with 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 the values of $\eta_{\text{fid}} = \eta_{\text{surv}} = 2$, $s = 4$, and $\mu = 8$. The tables show the (a) `HB` method and (b) `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) Simpler `equal` batch scheduling method always evaluates the same number of configurations within each batch and, therefore, makes optimal use of available parallel resources.

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 [19]: SH, also called sequential halving [46], is a simple multifidelity optimization algorithm that combines the 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 $r_{\min} < 1$. This is 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 [14]: Similar to SH, HB uses a fixed schedule for the fidelity parameter r, but it augments SH by using multiple *brackets* b of SH runs starting at different $r_{\min}(b)$ and with different $\mu(b)$. The number of brackets is set to

$$s = \lfloor \log_\eta(1/r_{\min}) \rfloor + 1 \tag{6}$$

which coincides with the number of fidelity steps that can be performed on a geometric scale on the interval $[r_{\min}, 1]$. 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: $\mu(b) = \lceil s \cdot (\eta^{s-b}/s - b + 1) \rceil$.

5) Bayesian Optimization Hyperband [16]: Model-based methods outperform HB 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 SH brackets like HB, 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 KDEs of the top and bottom configurations in $\mathcal{A}$, similar to the process in [47]. To implement BOHB in Algorithm 2, one, therefore, needs to use an inducer $\mathcal{I}_{f_{\text{surr}}}$ that

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	—

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 [48]: Fabolas is a continuous multifidelity BO method, where the conditional validation error is modeled 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 [15] and Asynchronous Hyperband: HB, 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 HB works similarly.

3) Asynchronous 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 [49] 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 (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 multifidelity methods in general?
RQ5a: Does changing SAMPLE configuration parameters throughout the optimization process offer an advantage?
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 [50], 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: HPO 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, and continuous); and 3) whether there are hierarchical dependencies between hyperparameters. More details on the characteristics of the problem classes are given in Appendix B in the supplementary material. 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 D in the supplementary material). This means that for analyzing 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 $\boldsymbol{\gamma}^{*\text{lcbench}}$ and $\boldsymbol{\gamma}^{*\text{rbv2\_super}}$, 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 $\mathbb{E}_{\omega \sim \mathbb{P}_\Omega}[\zeta(A(\omega, \boldsymbol{\gamma}))]$ for a

TABLE III
SUMMARY OF EXPERIMENT. SHOWN ARE THE VARIOUS OPTIMIZER CONFIGURATIONS $\boldsymbol{\gamma}$ 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 $\boldsymbol{\gamma}^*$

Name	RQ	Optimize	Design Modification
γ^*	1, 2, 3	✓	none (global optimization)
γ_1	4	✗	$\eta_{\text{fid}} \to \infty$
γ_2	5a	✓	$n_{\text{trn}}(0) = n_{\text{trn}}(1),\ N_s^0(0) = N_s^0(1),\ N_s^1(0) = N_s^1(1),\ \rho(0) = \rho(1)$
γ_3	5b	✓	*filter_method*$\to$ `tournament`, $n_{\text{trn}} \to 1,\ N_s^0(0) = N_s^0(1) = N_s^1(0) = N_s^1(1),\ \rho(0) = \rho(1)$
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

configuration γ in (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 has not been held out for subsequent analysis are used. As a configuration for our framework, we use BO with the lower confidence bound acquisition function [51] with interleaved random configurations every three evaluations.[2] Configuration is repeated three times for each scenario, each running for 60 h, 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* [52] 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 Table V in Appendix C in the supplementary material. 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 [53] (`RF`), K-nearest-neighbors with k set to 1 (`KNN1`), kernelized K-nearest-neighbors with "optimal" weighting [54] (`KKNN7`), and the ratio of density predictions of good and bad points, similar to tree parzen estimators [47] without a hierarchical structure as in BOHB [16] (`TPE`). For the prefiltering 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 in the supplementary material, this influences the behavior of $\mathcal{I}_{f_{\text{surr}}}$.

[2]Note that this optimizer used for third-level optimization is not an instance of Algorithm 2.

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 HB, it is set to the largest number of stages that is possible given η_{fid} and the lower bound on r according to (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 SA 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*.

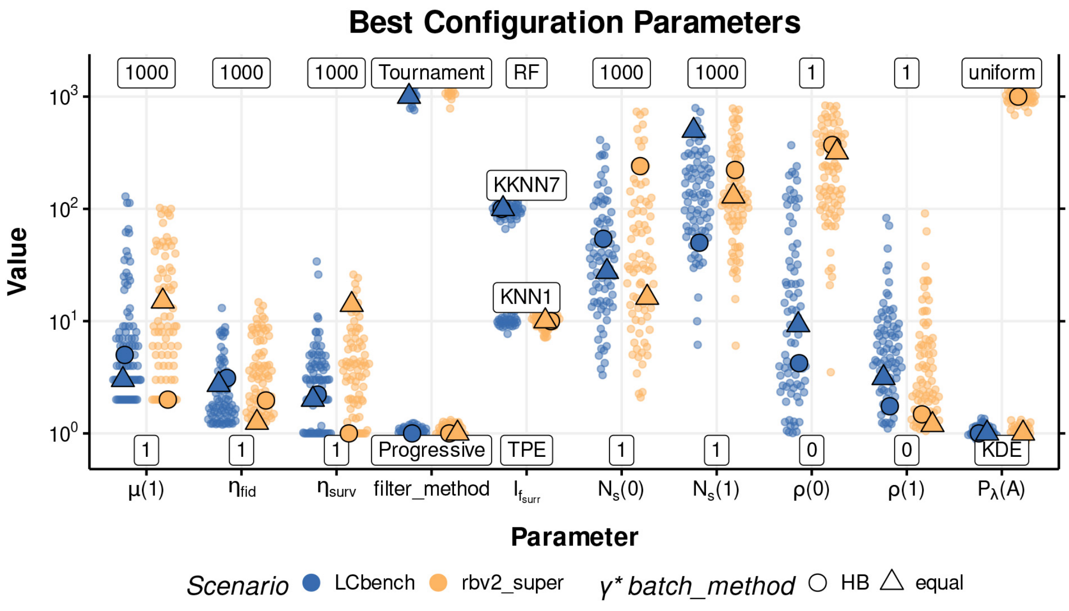


Fig. 2. Beeswarm plot of the best configurations according to the surrogate model over the meta-optimization archive of γ^*. Shown are the top 80 configuration points (according to the surrogate-model-predicted performance) that were evaluated during optimization. Levels of discrete parameters are shown. Most numeric parameters are on a log-scale (left axis), except for $\rho(0)$ and $\rho(1)$, 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.

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 in the supplementary material 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]. Fig. 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. Fig. 4 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 in the supplementary material and in Fig. 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. $\mathbb{P}_\lambda(\mathcal{A})$ 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 Fig. 4. The optimized results generalize well to test instances from the same scenario as they were configured on. Fig. 3 shows the optimization progress (on unseen test instances) of configurations if configured on the same scenario versus 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. Fig. 2), and that specific settings for these two algorithmic components are needed for *lcbench* to reach optimal performance.

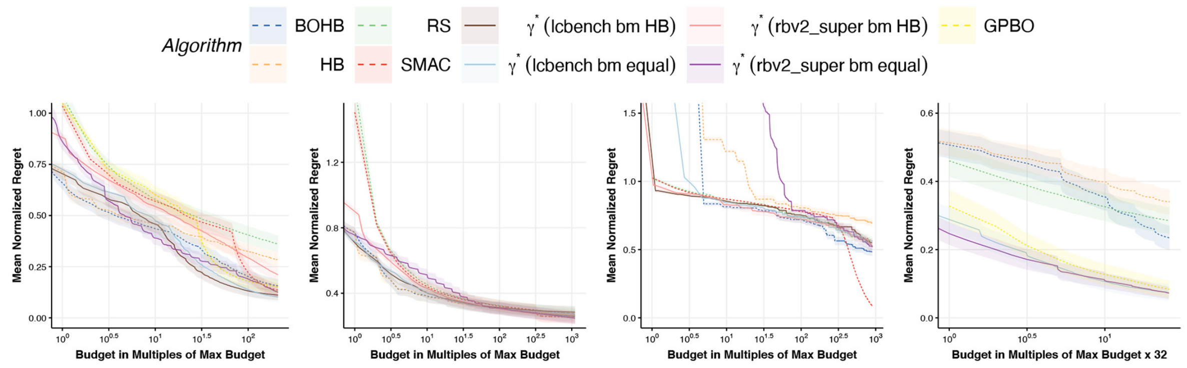


Fig. 3. Optimization progress (mean normalized regret) of serial evaluation on each benchmark scenario as well as 32× parallel evaluation on *lcbench*. Different configurations of Algorithm 2 are executed on benchmark functions that have not been used for the meta-optimization itself, and the progress 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 constructed 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 makes 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 Fig. 6 in Appendix E in the supplementary material for a larger version.

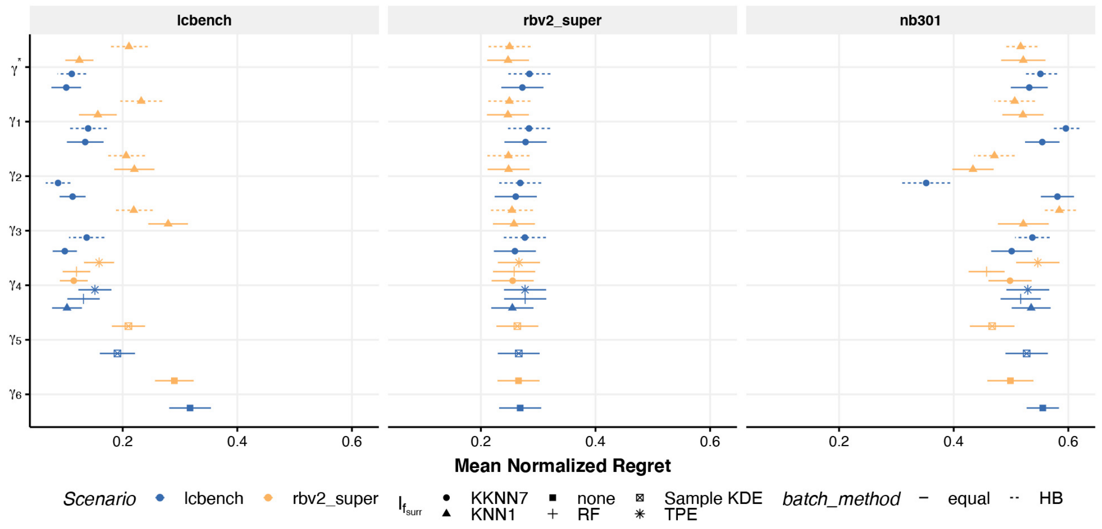


Fig. 4. Mean normalized regret of final performance on "test" benchmark instances for the configuration, shown in Table III. Shown is the mean over 30 evaluations, averaged over all available test benchmark instances for each of the three scenarios. The uncertainty bands show the standard error over instance means. 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.

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?

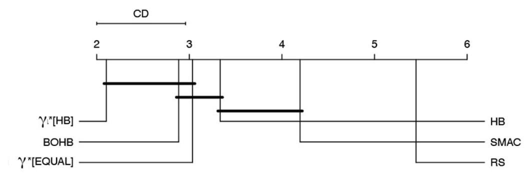
Setup: We evaluate several well-known HPO algorithms in their default configuration on the same benchmark instances: for BOHB [16], we use the implementation found in `HpBandSter`[3] (version 0.7.4); for HB [14], we use `mlr3hyperband`[4] (version 0.1.2); and for SMAC [5], we use the SMACv3 package[5] (version 1.0.1). We also construct a traditional Gaussian process-based BO (GPBO) [4] with mlrMBO[6] (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 multifidelity algorithms and therefore always evaluate points with maximum fidelity 1.

[3]https://github.com/automl/HpBandSter

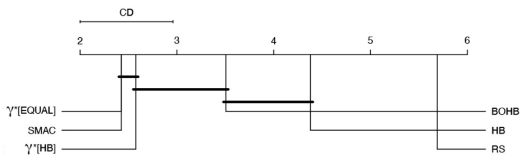
[4]https://cran.r-project.org/package=mlr3hyperband

[5]https://github.com/automl/SMAC3

[6]https://cran.r-project.org/package=mlrMBO



(a)



(b)

Fig. 5. Critical difference plot [55] 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. (a) Intermediate optimization budget of 100 full evaluations. (b) Full evaluation budget (final performance).

Results: The performance curves for the mean normalized regret are shown in Fig. 3, and the final performance values at $30 \cdot d$ full-fidelity evaluations are shown in Fig. 4. A critical difference plot and test can be seen in Fig. 5(b). 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 multifidelity 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 multifidelity 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 Fig. 5(b).

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 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 Fig. 5(a) and (b) shows 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 Fig. 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 multifidelity methods in general?

Setup: We evaluate the performance of a modified γ^* where the number of fidelity stages s is set to 1, thus ensuring that configurations are only evaluated with maximum fidelity 1.[7]

Results: Our results show the superiority of MF-HPO methods compared to HPO methods that do not make use of lower-fidelity approximations. Fig. 5(a) suggests that multifidelity methods are significantly better than their nonmultifidelity 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, Fig. 5(b) provides evidence that multifidelity 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 multifidelity 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 multifidelity 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.

[7]Because s is not part of the search space Γ and is instead given by 6, this is achieved by setting η_{fid} to ∞.

Results: The observations made for γ_2 (forbidding change over 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 introduce 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 in the supplementary material). Furthermore, $\boldsymbol{\gamma}^*$[equal] is evaluated with ρ set to 1 (i.e., all points are sampled randomly from a distribution that may be nonuniform), 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 Fig. 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?

Setup: The 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 [50]. 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: Fig. 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.[8] 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 MF-HPO. We formalized the search space of multifidelity 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 multifidelity evaluations provides substantial benefits. However, based on our experiments, we argue that design choices made by established multifidelity 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.

A 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 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 benefits. Optimization on *rbv2_super* did choose time-varying random interleaving, and overall, more aggressive filtering late during an optimization run ($N_s(1) > N_s(0)$) was slightly favored, but the results did not consistently outperform a configuration obtained from a restricted optimization that excluded time-varying configuration parameters.

[8]https://github.com/mlr-org/smashy, https://github.com/compstat-lmu/paper_2021_benchmarking_special_issue

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.

References

[1] B. Bischl et al., "Hyperparameter optimization: Foundations, algorithms, best practices and open challenges," 2021, *arXiv:2107.05847*.

[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, Switzerland: Springer Int., 2019, pp. 81–95.

[3] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 67–82, Apr. 1997.

[4] D. R. Jones, M. Schonlau, and W. J. Welch, "Efficient global optimization of expensive black-box functions," *J. Global 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, Germany: Springer, 2011, pp. 507–523.

[6] K. Swersky, J. Snoek, and R. P. Adams, "Freeze-thaw Bayesian optimization," 2014, *arXiv:1406.3896*.

[7] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, pp. 281–305, Feb. 2012.

[8] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian optimization of machine learning algorithms," in *Proc. 25th Int. Conf. Neural Inf. Process. Syst. Vol. 2*, 2012, pp. 2951–2959.

[9] R. Turner et al., "Bayesian optimization is superior to random search for machine learning hyperparameter tuning: Analysis of the black-box optimization challenge 2020," in *Proc. NeurIPS Competition Demonstration Track*, vol. 133. Vancouver, BC, Canada, Dec. 2020, pp. 3–26.

[10] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Parallel algorithm configuration," in *Learning and Intelligent Optimization*, Y. Hamadi and M. Schoenauer, Eds. Berlin, Germany: Springer, 2012, 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 *Proc. 8th Int. Conf. Learn. Intell. Optim.*, Gainesville, FL, USA, Feb. 2014, pp. 173–186.

[12] J. González, Z. Dai, P. Hennig, and N. D. Lawrence, "Batch Bayesian optimization via local penalization," in *Proc. 19th Int. Conf. Artif. Intell. Stat. (AISTATS)*, vol. 51. Cadiz, Spain, May 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*. Berlin, Germany: Springer, 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. 1–52, Jan. 2017.

[15] L. Li et al., "A system for massively parallel hyperparameter tuning," in *Proc. Int. Conf. Mach. Learn. Syst. (MLSys)*, Austin, TX, USA, Mar. 2020, pp. 230–246.

[16] S. Falkner, A. Klein, and F. Hutter, "BOHB: Robust and efficient hyperparameter optimization at scale," in *Proc. 35th Int. Conf. Mach. Learn. (ICML)*, vol. 80. Stockholm, Sweden, Jul. 2018, pp. 1436–1445.

[17] L. C. Tiao, A. Klein, C. Archambeau, and M. W. Seeger, "Model-based asynchronous hyperparameter optimization," 2020, *arXiv:2003.10865*.

[18] D. Sculley et al., "Hidden technical debt in machine learning systems," in *Proc. Annu. Conf. Neural Inf. Process. Syst.*, Montreal, QC, Canada, Dec. 2015, pp. 2503–2511.

[19] K. G. Jamieson and A. Talwalkar, "Non-stochastic best arm identification and hyperparameter optimization," in *Proc. 19th Int. Conf. Artif. Intell. Stat. (AISTATS)*, vol. 51. Cadiz, Spain, May 2016, pp. 240–248.

[20] H. H. Hoos, "Programming by optimization," *Commun. Assoc. Comput. Mach.*, 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, Sep. 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 *Proc. 9th Workshop IEEE/IFIP Conf. Softw. Archit.*, 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 *Proc. 21st Int. Joint Conf. Artif. Intell.*. San Francisco, CA, USA, 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*. Boston, MA, USA: Springer, 2009, pp. 43–59.

[26] M. López-Ibáñez and T. Stützle, "The automatic design of multiobjective ant colony optimization algorithms," *IEEE Trans. Evol. Comput.*, vol. 16, no. 6, pp. 861–875, Dec. 2012.

[27] L. C. T. Bezerra, M. López-Ibáñez, and T. Stützle, "Automatic component-wise design of multiobjective evolutionary algorithms," *IEEE Trans. Evol. Comput.*, vol. 20, no. 3, pp. 403–417, Jun. 2016.

[28] M. Birattari, Z. Yuan, P. Balaprakash, and T. Stützle, "F-race and iterated F-race: An overview," in *Experimental Methods for the Analysis of Optimization Algorithms*. Berlin, Germany: Springer, 2010, pp. 311–336.

[29] M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, and T. Stützle, "The irace package: Iterated racing for automatic algorithm configuration," *Oper. Res. Perspect.*, vol. 3, pp. 43–58, Jan. 2016.

[30] N. Dang, L. P. Cáceres, P. D. Causmaecker, and T. Stützle, "Configuring irace using surrogate configuration benchmarks," in *Proc. Genet. Evol. Comput. Conf. (GECCO)*, Berlin, Germany, Jul. 2017, pp. 243–250.

[31] F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stützle, "ParamILS: An automatic algorithm configuration framework," *J. Artif. Intell. Res.*, vol. 36, pp. 267–306, Sep. 2009.

[32] O. Maron and A. W. Moore, "The racing algorithm: Model selection for lazy learners," *Artif. Intell. Rev.*, vol. 11, nos. 1–5, pp. 193–225, 1997. [Online]. Available: https://doi.org/10.1023/A:1006556606079

[33] S. van Rijn, H. Wang, M. van Leeuwen, and T. Bäck, "Evolving the structure of evolution strategies," in *Proc. IEEE Symp. Ser. Comput. Intell. (SSCI)*, 2016, pp. 1–8.

[34] G. Malkomes and R. Garnett, "Automating Bayesian optimization with Bayesian optimization," in *Proc. Int. Conf. Adv. Neural Inf. Process. Syst.*, vol. 31, 2018, pp. 5984–5994.

[35] M. Lindauer, M. Feurer, K. Eggensperger, A. Biedenkapp, and F. Hutter, "Towards assessing the impact of Bayesian optimization's own hyperparameters," 2019, *arXiv:1908.06674*.

[36] M. Lindauer et al., "SMAC3: A versatile Bayesian optimization package for hyperparameter optimization," 2021, *arXiv:2109.09831*.

[37] A. Saltelli, "Sensitivity analysis for importance assessment," *Risk Anal.*, vol. 22, no. 3, pp. 579–590, 2002.

[38] W. Hoeffding, "A class of statistics with asymptotically normal distribution," *Ann. Math. Stat.*, vol. 19, no. 3, pp. 293–325, 1948.

[39] F. Hutter, H. Hoos, and K. Leyton-Brown, "An efficient approach for assessing hyperparameter importance," in *Proc. 31st Int. Conf. Mach. Learn.*, vol. 32. Bejing, China, Jun. 2014, pp. 754–762.

[40] C. Fawcett and H. H. Hoos, "Analysing differences between algorithm configurations through ablation," *J. Heuristics*, vol. 22, no. 4, pp. 431–458, 2016.

[41] S. Sheikholeslami, M. Meister, T. Wang, A. H. Payberah, V. Vlassov, and J. Dowling, "AutoAblation: Automated parallel ablation studies for deep learning," in *Proc. 1st Workshop Mach. Learn. (EuroMLSys@EuroSys)*, Edinburgh, U.K., Apr. 2021, pp. 55–61.

[42] M. López-Ibáñez and T. Stützle, "An experimental analysis of design choices of multi-objective ant colony optimization algorithms," *Swarm Intell.*, vol. 6, pp. 207–232, Jul. 2012.

[43] 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 *Proc. Genet. Evol. Comput. Conf. Companion*, 2021, pp. 1375–1384.

[44] A. Klein, L. C. Tiao, T. Lienart, C. Archambeau, and M. Seeger, "Model-based asynchronous hyperparameter and neural architecture search," 2020, *arXiv:2003.10865*.

[45] M. Birattari, *Tuning Metaheuristics—A Machine Learning Perspective* (Studies in Computational Intelligence), vol. 197. Berlin, Germany: Springer, 2009.

[46] Z. Karnin, T. Koren, and O. Somekh, "Almost optimal exploration in multi-armed bandits," in *Proc. Int. Conf. Mach. Learn.*, 2013, pp. 1238–1246.

[47] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Proc. Int. Conf. Adv. Neural Inf. Process. Syst.*, vol. 24, 2011, pp. 2546–2554.

[48] A. Klein, S. Falkner, S. Bartels, P. Hennig, and F. Hutter, "Fast Bayesian optimization of machine learning hyperparameters on large datasets," in *Proc. Int. Conf. Artif. Intell. Stat.*, 2017, pp. 528–536.

[49] D. Golovin, B. Solnik, S. Moitra, G. Kochanski, J. Karro, and D. Sculley, "Google vizier: A service for black-box optimization," in *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.*, 2017, pp. 1487–1495.

[50] F. Pfisterer, L. Schneider, J. Moosbauer, M. Binder, and B. Bischl, "YAHPO gym—Design criteria and a new multifidelity benchmark for hyperparameter optimization," 2021, *arXiv:2109.03670*.

[51] D. R. Jones, "A taxonomy of global optimization methods based on response surfaces," *J. Global Optim.*, vol. 21, no. 4, pp. 345–383, 2001.

[52] H. Jalali, I. Van Nieuwenhuyse, and V. Picheny, "Comparison of kriging-based algorithms for simulation optimization with heterogeneous noise," *Eur. J. Oper. Res.*, vol. 261, no. 1, pp. 279–301, 2017.

[53] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.

[54] R. J. Samworth, "Optimal weighted nearest neighbour classifiers," *Ann. Statist.*, vol. 40, no. 5, pp. 2733–2763, Oct. 2012.

[55] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *J. Mach. Learn. Res.*, vol. 7, pp. 1–30, Dec. 2006.

Julia Moosbauer is currently pursuing the Doctoral degree with the Chair for Statistical Learning and Data Science, Ludwig-Maximilians-Universität München, Munich, Germany.

She is a member of the Munich Center for Machine Learning. Her research focus lies in the interface between automated and explainable machine learning with the goal to increase transparency and trust into automated machine learning systems. She is also interested in hyperparameter optimization algorithms (in particular, Bayesian optimization), algorithm configuration, multiobjective optimization, and (sequential) experimental design.

Martin Binder is currently pursuing the Doctoral degree with the Chair for Statistical Learning and Data Science, Ludwig-Maximilians-Universität München, Munich, Germany.

He is a member of the Munich Center for Machine Learning. He is mostly working on black-box optimization methods for automatic machine learning and hyperparameter optimization, with a focus on multifidelity optimization. He also works on deep learning, specifically self-supervised learning, for genome sequence classification and analysis.

Lennart Schneider is currently pursuing the Doctoral degree with the Chair of Statistical Learning and Data Science, Ludwig-Maximilians-Universität München, Munich, Germany.

His research focuses on automated machine learning, hyperparameter optimization, multiobjective optimization, and neural architecture search.

Florian Pfisterer received the master's degree in statistics from the Ludwig Maximilian University of Munich, Munich, Germany, in 2018, where he is currently pursuing the Ph.D. degree with the Statistical Learning and Data Science Group, Department of Statistics.

His research interests are in the field of automated machine learning, multiobjective optimization, and algorithmic fairness.

Marc Becker received the master's degree in geoinformatics from Friedrich-Schiller University Jena, Jena, Germany, in 2020.

Since 2020, he has been a Research Engineer with the Ludwig Maximilian University of Munich, Munich, Germany, and a Main Developer of the mlr3 optimization packages.

Michel Lang received the Ph.D. degree in statistics, Dortmund Technical University, Dortmund, Germany, in 2015.

He is a former member of the Munich Center for Machine Learning. He is currently a Scientific Manager of the Research Center Trustworthy Data Science and Security, Dortmund, Germany. He is the author of many popular R packages for machine learning and parallelization, e.g., mlr3 or batchtools. His research areas include machine learning, optimization, and software development.

Lars Kotthoff received the Doctoral degree in computer science from the University of St Andrews, St Andrews, U.K., in 2012.

He is an Assistant Professor of Computer Science with the University of Wyoming, Laramie, WY, USA. His research focuses on data-driven combinatorial optimization, automated machine learning, and applying machine learning in other disciplines, for example, materials science.

Bernd Bischl studied computer science, artificial intelligence, and data sciences in Hamburg, Germany; Edinburgh, U.K.; and Dortmund, Germany. He received the Ph.D. degree in statistics from Dortmund Technical University, Dortmund, in 2013 with a thesis on "Model and Algorithm Selection in Statistical Learning and Optimization."

He holds the Chair of Statistical Learning and Data Science with the Department of Statistics, Ludwig-Maximilians-Universität München, Munich, Germany, and he is the Co-Director of the Munich Center for Machine Learning, one of Germany's national competence centers for ML. He is a member of ELLISl, and a Faculty Member of ELLIS Munich, an active developer of several R-packages, leads the "mlr" (Machine Learning in R) Engineering Group and is the Co-Founder of the Science Platform "OpenML" for open and reproducible ML. Furthermore, he leads the Munich branch of the Fraunhofer ADA Lovelace Center for Analytics, Data, and Applications, i.e., a new type of research infrastructure to support businesses in Bavaria, especially in the SME sector. His research interests include AutoML, model selection, interpretable ML, as well as the development of statistical software.

6.2. YAHPO Gym - An Efficient Multi-Objective Multi-Fidelity Benchmark for Hyperparameter Optimization

YAHPO Gym is an surrogate-based benchmark which allows to evaluate HPO methods on a multitude of benchmark scenarios and for multiple objectives. The benchmark has been designed on surrogates, which allows conducting benchmark studies considerably more cost-effective as compared to benchmarking on real problem instances. Since computational costs can be a limitation to running expressive benchmark studies and ablation analyses, this work clearly facilitates benchmarking and thus contributes to enhancing explainability of AutoML frameworks, see Section 3.3 – *Benchmarking of AutoML tools (A)* and *Sensitivity analysis of AutoML tools (B)*.

Contributing article:

Pfisterer, F., Schneider, L., Moosbauer, J., and Binder, M., Bischl, B. (2022). YAHPO Gym - An Efficient Multi-Objective Multi-Fidelity Benchmark for Hyperparameter Optimization. *Proceedings of the First International Conference on Automated Machine Learning, Proceedings of Machine Learning Research vol. 188, pp. 3/1–39.*

Author contributions:

Florian Pfisterer and Lennart Schneider contributed equally. The core idea for the system originated from Florian Pfisterer who also developed initial code and a first version of the system. Florian Pfisterer re-implemented the underlying software with the help by Lennart Schneider. Florian Pfisterer, Lennart Schneider, and Martin Binder collected samples from relevant benchmarks and performance datasets. Lennart Schneider contributed several improvements to software, stability, and functionality as well as automated tuning. Martin Binder, Julia Moosbauer, and Bernd Bischl advised throughout this process. Lennart Schneider and Florian Pfisterer jointly developed the experiments, which were executed by Lennart Schneider who also contributed implementations of relevant baseline algorithms. Florian Pfisterer and Lennart Schneider jointly authored the resulting manuscript with input and improvements by Martin Binder, Julia Moosbauer, and Bernd Bischl.

Supplementary material available at:

- R package: `https://github.com/slds-lmu/yahpo_gym`
- Supplementary material:
  `https://proceedings.mlr.press/v188/pfisterer22a.html` (full paper)

YAHPO Gym - An Efficient Multi-Objective Multi-Fidelity Benchmark for Hyperparameter Optimization

Florian Pfisterer[1,2] **Lennart Schneider**[1,2] **Julia Moosbauer**[1] **Martin Binder**[1] **Bernd Bischl**[1]

[1]Department of Statistics, LMU Munich, Germany
[2]Equal contributions

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 [43]. Over the last ten years, a large and still growing set of HPO tuning methods based on different principles has been developed [31, 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 [44, 21, 35], as well as multi-objective methods, which allow for simultaneous optimization of multiple objectives [40]. 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, 74, 59] and *meta-optimization* [49, 53]. 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, 15, 60, 19]. 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. Each category has 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 (NAS). 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* [14], 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 [29] provides a collection of various synthetic black-box benchmark functions, while *kurobako* [56] 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* [15] 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 [6]. Its successor *HPOBench* [19] offers similar capabilities, focussing on reproducible containerized benchmarks. It offers 12 benchmark scenarios and more than 100 test instances. Recently, [60] 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* [37] 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, 72, 22] and NAS (c.f. [50]). Eggensperger et al. [17] compared

[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 $0.13s$.

different instance surrogate models for 9 different HPO problems and 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 [17]). Similar observations have been made in the context of algorithm configuration [18] and NAS [65].

We compare YAHPO Gym with the recently published benchmarks HPOBench [19] and HPO-B [60] 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. [55]). Instead of relying on *containerization* to allow for portability, our library relies on neural network surrogates compressed using *ONNX* [3], 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. See details in Supplement B.2. 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. See details in Supplement E.2 and Supplement E.3.

3 Background

3.1 Hyperparameter Optimization

An ML *learner* or *inducer* $\mathcal{I}$ configured by hyperparameters $\boldsymbol{\lambda} \in \Lambda$ maps a dataset $\mathcal{D} \in \mathbb{D}$ to a model $\hat{f}$, i.e., $\mathcal{I} : \mathbb{D} \times \Lambda \rightarrow \mathcal{H}, (\mathcal{D}, \boldsymbol{\lambda}) \mapsto \hat{f}$. HPO methods for ML aim to identify a well-performing hyperparameter configuration (HPC) $\boldsymbol{\lambda} \in \tilde{\Lambda}$ for $\mathcal{I}_{\boldsymbol{\lambda}}$ [10]. 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:

$$\boldsymbol{\lambda}^* \in \arg\min_{\boldsymbol{\lambda} \in \tilde{\Lambda}} c(\boldsymbol{\lambda}), \quad \text{with} \quad c : \tilde{\Lambda} \rightarrow \mathbb{R}^m, \tag{1}$$

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 [10] 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, 30, 7, 57, 27]. $c(\boldsymbol{\lambda})$ 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 [44, 62, 35]. For example, 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 [19] 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 pre-recorded 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 Supplement B.3. *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, 59, 24]. 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 $\tilde{\Lambda}$ 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: Categorical	1	2: perf(1) + rt(1)	epoch	✓
lcbench	7D: Numeric	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: Numeric	4	12: perf(4) + inp(3) + rt(2) + mem(3)	fraction	
iaml_glmnet	2D: Numeric	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	✓

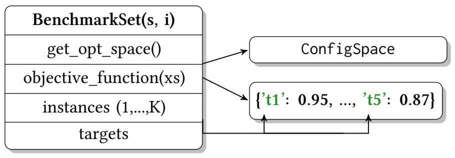
Mixed = numeric and categorical hyperparameters; perf = performance measures; rt = train/predict time; mem = memory consumption; inp = interpretability measures; H = Hierarchical search space. We do not include the fidelity parameter in the search space dimensionality.

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 (mostly 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` [3] as surrogate models. Our neural networks are `ResNets` for tabular data [26] 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



(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.

Figure 1: API overview.

across platforms 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 **v1.0**) 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 Supplement C.2. 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 [35]). 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, [44]) across all settings. BO is configured with algorithm surrogate model either a Gaussian process (BO_GP), ensemble of feed-forward neural networks (BO_NN, [73]) or random forest (BO_RF, [12]) and acquisition function optimizer either Nelder-Mead/exhaustive search[3] (*_DF [54]) or Random

[3]for tabular benchmarks

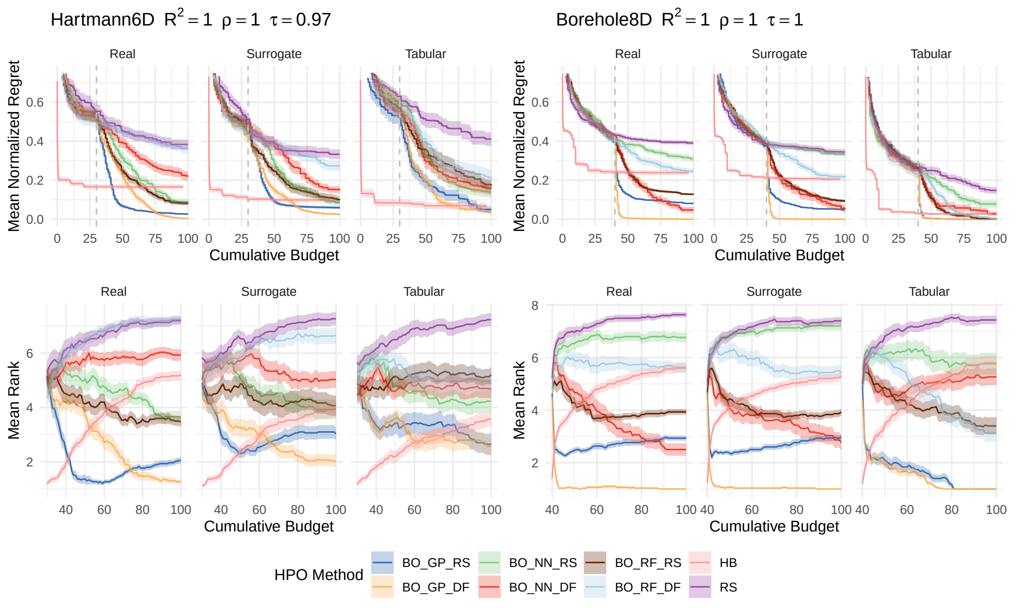


Figure 2: Mean normalized regret (top) and mean ranks (bottom) of different HPO methods on different benchmarks. Ribbons represent standard errors. The gray vertical line indicates the cumulative budget used for the initial design of BO methods. Performance measures of the surrogate benchmarks are stated after the benchmark function. 30 replications.

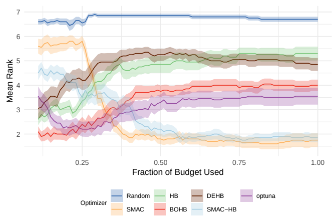
Search (*_RS). We describe additional details regarding the benchmark setup in Supplement E.1 and briefly present results: Figure 2 shows the anytime performance and mean rank of each HPO method split for the real, surrogate, and tabular benchmark on the Hartmann6D and Borehole8D test functions. We observe 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 [51] for a given benchmark type. We use the optimization based symmetric difference (SD) [36] 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

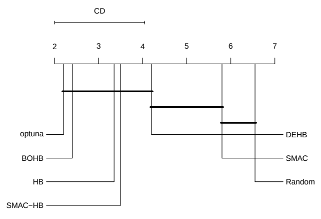
We now demonstrate how YAHPO Gym can be used in practice to benchmark different HPO methods. We benchmark 7 single-objective HPO methods on YAHPO-SO and 7 multi-objective HPO methods on YAHPO-MO and want to answer the following research questions: **(RQ1)** *Do multi-fidelity (single-objective) HPO methods 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?

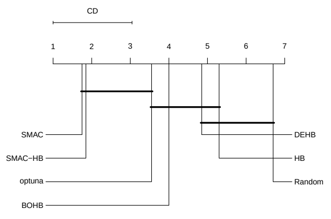
We compare Random Search and SMAC (SMAC4HPO facade; [47]) to the multi-fidelity methods Hyperband [44], BOHB [21], DEHB [4], SMAC-HB (SMAC4MF facade; [47]) and optuna ([2]; TPE sampler and median pruner following successive halving steps). More details on the experimental setup and HPO methods is given 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 [19] 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.

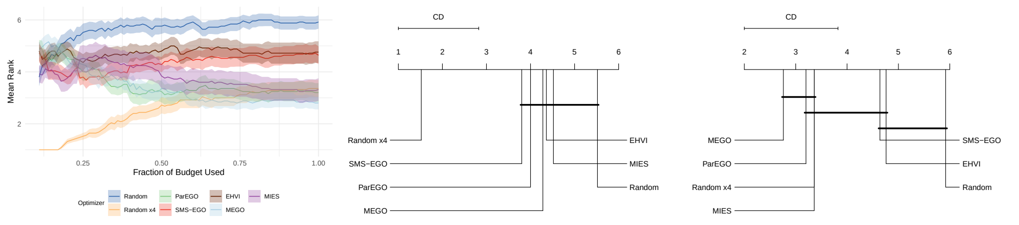


(c) Critical differences plot for mean ranks of of HPO methods after 100% of the optimization budget.

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 [40], SMS-EGO [61], EHVI [20], MEGO [33] and MIES [46] 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%.

(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 139.57 CPU days using YAHPO Gym. We estimate that the YAHPO-SO benchmark, would take 14.75 CPU **years** when running real benchmarks, while our benchmark using YAHPO Gym took only 397.51 CPU **hours**, essentially speeding up evaluation by a factor of ~ 300.

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 [21]. 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.

8 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 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] 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]

(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 run. Random seeds can be obtained from the accompanying 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 139.57 CPU days across all replications (263 kg CO_2 equivalent). The tabular vs. surrogate benchmark required 22 CPU-hours (2 kg 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]

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].

References

[1] Advanced Research Computing Center. Teton computing environment. `https://doi.org/10.15786/M2FY47`, 2018. University of Wyoming.

[2] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. 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] ONNX authors. ONNX. `https://github.com/onnx/onnx`, 2022.

[4] N. Awad, N. Mallik, and F. Hutter. DEHB: Evolutionary hyberband for scalable, robust and efficient hyperparameter optimization. *arXiv:2105.09821 [cs.LG]*, 2021.

[5] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, pages 2546–2554, 2011.

[6] J. Bergstra, B. Komer, C. Eliasmith, and D. Warde-Farley. Preliminary evaluation of hyperopt algorithms on HPOLib. In *ICML Workshop on Automatic Machine Learning*, 2014.

[7] M. Binder, J. Moosbauer, J. Thomas, and B. Bischl. Multi-objective hyperparameter tuning and feature selection using filter ensembles. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pages 471–479, 2020.

[8] 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.

[9] M. Binder, F. Psterer, and B. Bischl. Collecting empirical data about hyperparameters for data driven AutoML. In *ICML Workshop on Automatic Machine Learning*, 2020.

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

[11] B. Bischl, P. Kerschke, L. Kotthoff, M. Lindauer, Y. Malitsky, A. Fréchette, H. Hoos, F. Hutter, K. Leyton-Brown, K. Tierney, and Vanschoren J. Aslib: A benchmark library for algorithm selection. *Artificial Intelligence*, 237:41–58, 2016.

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

[13] H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, R. Anil, Z. Haque, L. Hong, V. Jain, X. Liu, and H. Shah. Wide & deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, pages 7–10, 2016.

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

[15] K. Eggensperger, M. Feurer, F. Hutter, J. Bergstra, J. Snoek, H. 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.

[16] K. Eggensperger, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Surrogate benchmarks for hyperparameter optimization. In *MetaSel@ ECAI*, pages 24–31, 2014.

[17] K. Eggensperger, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Efficient benchmarking of hyperparameter optimizers via surrogates. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, pages 1114–1120, 2015.

[18] K. Eggensperger, M. Lindauer, H. H. Hoos, F. Hutter, and K. Leyton-Brown. Efficient benchmarking of algorithm configurators via model-based surrogates. *Machine Learning*, 107(1):15–41, 2018.

[19] 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 J. Vanschoren and S. Yeung, editors, *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021.

[20] M. T. M. Emmerich. Single- and multi-objective evolutionary design optimization assisted by Gaussian random field metamodels. *PhD Dissertation*, 2005.

[21] S. Falkner, A. Klein, and F. Hutter. BOHB: Robust and efficient hyperparameter optimization at scale. In *International Conference on Machine Learning*, pages 1437–1446, 2018.

[22] M. Feurer, B. Letham, F. Hutter, and E. Bakshy. Practical transfer learning for Bayesian optimization. *arXiv:1802.02219 [stat.ML]*, 2021.

[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, 2015.

[24] P. Gijsbers, F. Pfisterer, J. N. van Rijn, B. Bischl, and J. Vanschoren. Meta-learning for symbolic hyperparameter defaults. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 151–152, 2021.

[25] 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*, pages 1487–1495, 2017.

[26] Y. Gorishniy, I. Rubachev, V. Khrulkov, and A. Babenko. Revisiting deep learning models for tabular data. *Advances in Neural Information Processing Systems*, 34, 2021.

[27] 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.

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

[29] N. Hansen, A. Auger, R. Ros, O. Mersmann, T. Tušar, and D. Brockhoff. COCO: A platform for comparing continuous optimizers in a black-box setting. *Optimization Methods and Software*, 36(1):114–144, 2021.

[30] D. Horn and B. 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.

[31] 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.

[32] K. Jamieson and A. Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2015.

[33] 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.

[34] 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.

[35] K. Kandasamy, G. Dasarathy, J. Schneider, and B. Póczos. Multi-fidelity Bayesian optimisation with continuous approximations. In *International Conference on Machine Learning*, pages 1799–1808, 2017.

[36] J. G. Kemeny and J. L. Snell. *Mathematical Models in the Social Sciences.* MIT Press, Cambridge, MA, USA, 1972.

[37] A. Klein, Z. Dai, F. Hutter, N. Lawrence, and J. 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, 2019.

[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, pages 528–536, 2017.

[39] A. Klein, L. C. Tiao, T. Lienart, C. Archambeau, and M. Seeger. Model-based asynchronous hyperparameter and neural architecture search. *arXiv:2003.10865 [cs.LG]*, 2020.

[40] 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.

[41] 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*, 4(44):1903, 2019.

[42] M. Lang, B. Bischl, and D. Surmann. batchtools: Tools for R to work on batch systems. *The Journal of Open Source Software*, 2017.

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

[44] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18(185):1–52, 2018.

[45] L. Li, K. Jamieson, A. Rostamizadeh, E. Gonina, J. Bentzur, M. Hardt, B. Recht, and A. 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] R. Li, M. T. M. Emmerich, J. Eggermont, T. Bäck, M. Schütz, J. Dijkstra, and J. H. C. Reiber. Mixed integer evolution strategies for parameter optimization. *Evolutionary Computation*, 21(1):29–64, 2013.

[47] 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.

[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] M. Lindauer, M. Feurer, K. Eggensperger, A. Biedenkapp, and F. Hutter. Towards assessing the impact of Bayesian optimization's own hyperparameters. *arXiv:1908.06674 [cs.LG]*, 2019.

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

[51] O. Mersmann, H. Trautmann, B. Naujoks, and C. Weihs. Benchmarking evolutionary multi-objective optimization algorithms. In *IEEE Congress on Evolutionary Computation*, pages 1–8, 2010.

[52] C. Molnar, G. Casalicchio, and B. Bischl. Quantifying model complexity via functional decomposition for better post-hoc interpretability. In P. Cellier and K. Driessens, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 193–204, 2020.

[53] 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. *arXiv:2111.14756 [cs.LG]*, 2021.

[54] J. A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, 1965.

[55] 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*, page e1441, 2021.

[56] T. Ohta and H. V. Yamazaki. Kurobako. `https://github.com/optuna/kurobako`, 2022.

[57] M. Parsa, J. P. Mitchell, C. D. Schuman, R. M. Patton, T. E. Potok, and K. Roy. Bayesian multi-objective hyperparameter optimization for accurate, fast, and efficient neural network accelerator design. *Frontiers in Neuroscience*, 14:667, 2020.

[58] V. Perrone, R. Jenatton, M. W. Seeger, and C. Archambeau. Scalable hyperparameter transfer learning. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

[59] F. Pfisterer, J. N. van Rijn, P. Probst, A. C. Müller, and B. Bischl. Learning multiple defaults for machine learning algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 241–242, 2021.

[60] S. Pineda Arango, H. S. Jomaa, M. Wistuba, and J. Grabocka. HPO-B: A large-scale reproducible benchmark for black-box HPO based on OpenML. In J. Vanschoren and S. Yeung, editors, *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021.

[61] 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 G. Rudolph, T. Jansen, N. Beume, S. Lucas, and C. Poloni, editors, *Parallel Problem Solving from Nature PPSN X*, pages 784–794, 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] R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni. Green AI. *Communications of the ACM*, 63(12):54–63, 2020.

[64] K. Šehić, A. Gramfort, J. Salmon, and L. Nardi. LassoBench: A high-dimensional hyperparameter optimization benchmark suite for lasso. *arXiv:2111.02790 [cs.LG]*, 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] J. Snoek, H. Larochelle, and R. P. Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, volume 25, 2012.

[67] K. Swersky, J. Snoek, and R. P. Adams. Freeze-thaw Bayesian optimization. *arXiv:1406.3896 [stat.ML]*, 2014.

[68] 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.

[69] R. Turner. Uber bayesopt benchmark. `https://github.com/uber/bayesmark`, 2022.

[70] J. Vanschoren. Meta-Learning. In F. Hutter, L. Kotthoff, and J. Vanschoren, editors, *Automated Machine Learning: Methods, Systems, Challenges*, pages 35–61. Springer International Publishing, 2019.

[71] J. Vanschoren, J. N. van Rijn, B. Bischl, and L. Torgo. OpenML: Networked science in machine learning. *SIGKDD Explor.*, 15(2):49–60, 2013.

[72] M. Volpp, L. P. Fröhlich, K. Fischer, A. Doerr, S. Falkner, F. Hutter, and C. Daniel. Meta-learning acquisition functions for transfer learning in Bayesian optimization. *International Conference on Learning Representations*, 2020.

[73] 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.

[74] M. 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.

[75] M. Wistuba, N. Schilling, and L. Schmidt-Thieme. Two-stage transfer surrogate model for automatic hyperparameter optimization. In *European Conference on Machine Learning and Knowledge Discovery in Databases - Volume 9851*, pages 199–214, 2016.

[76] 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.

[77] L. Zimmer. data_2k_lw.zip. figshare. Dataset. `https://doi.org/10.6084/m9.figshare.11662422.v1`, Apache License, Version 2.0, 2020.

[78] L. Zimmer. nasbench301_full_data. figshare. Dataset. `https://doi.org/10.6084/m9.figshare.13286105.v1`, Apache License, Version 2.0, 2020.

[79] 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*, 43(9):3079–3090, 2021.

[80] 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):117–132, 2003.

Part IV.

Conclusion and future work

7. Conclusion and future work

This thesis has highlighted the importance of explainability for ensuring responsible and safe interaction with AutoML systems. By formally and systematically connecting the emerging fields of AutoML and explainable AI, this work aims to provide the community with a comprehensive and unified understanding of explainable AutoML. The key contributions of this thesis lie in the three distinct levels at which explainability can be a requirement – model, algorithm, and AutoML system explainability. However, as we strive towards more explainable AutoML systems, many questions remain unanswered, problems unresolved, and several avenues of research remain unexplored.

Model explainability In recent years, there has been significant growth in the field of explainability in ML, and many interpretability methods have emerged. Although it may appear that the entire range of *post-hoc* interpretability methods can be straightforwardly applied to models generated by AutoML systems, it is important to consider that interpretability methods may come with their pitfalls and may provide inaccurate and wrong information if applied in the wrong context (Molnar et al., 2020). If AutoML developers make an attempt to integrate interpretability methods into AutoML systems for better usability and accessability, AutoML systems should automatically verify assumptions underlying respective interpretability methods (e.g., checking for correlated features), return uncertainty estimates for interpretability methods and issue warnings in case assumptions are violated. Ideally, AutoML systems should also be capable of selecting the most appropriate interpretability method based on the assumptions underlying the interpretation method.

One promising approach to designing AutoML systems that prioritize model explainability are multi-objective systems that incorporate a preference for explainability, because a user does not have to a-priori define the trade-off between multiple objectives. The key to the successful implementation of multi-objective AutoML systems that take explainability into account is that explainability can be quantified in an objective and model-agnostic manner. Although there have been some model-agnostic methods proposed in the past for measuring model explainability, such as using model complexity as a proxy (Molnar et al., 2019) or using explainability metrics based on ordinary language philosophy (Sovrano and Vitali, 2021), it is still open to future work to integrate the most suitable metric into AutoML systems.

In high-risk domains, it may be valuable to develop AutoML systems that enable users to discover causal relationships within models. Beyond discovering and presenting causal relationships, it would be intriguing to investigate whether users presented with causal relationships modeled by a candidate model can provide interactive feedback during an AutoML process on desired and undesired causal relationships, thereby guiding the optimization process. For example, consider a use case where a car insurance company aims to price insurance for car owners by predicting their accident rates, assuming a latent variable of aggressive driving that increases the likelihood of accidents (desired causal relationship), as well as the likelihood that people prefer red cars (undesired causal relationship), which people with certain characteristics also prefer. Suppose an AutoML system could identify suspected causal relationships throughout the optimization process and enable users to investigate and reject undesired ones (e.g., confounding effects or discriminative causal relationships), which could then be used as a constraint by the AutoML system to find a better model that does not account undesired causal relationships.

Staying along the lines of more interactive AutoML that is accessible to domain experts without ML expertise, it would be compelling to investigate the possibility of creating AutoML systems that can incorporate human-understandable input. This input might involve expectations or restrictions on model behavior. In other words, beyond looking into AutoML only returning human-understandable outputs, it would also be interesting to see whether AutoML system can be designed to take human-understandable inputs.

Safe and reliable interaction with AutoML systems in highly regulated domains like insurance, finance, and healthcare can be facilitated by explainability. However, explainability may not be sufficient for safety, and applications in highly regulated domains may require AutoML systems to provide statistical guarantees on the returned models' performance.

Algorithm explainability To achieve successful algorithm explainability, the community needs to put a concerted effort into agreeing on definitions and establishing a shared understanding of concepts related to algorithm explainability, as illustrated by the example of hyperparameter importance in Figure 3.2. Additionally, it is important to identify and prioritize the key questions we aim to answer to generate a more comprehensive understanding of algorithm behavior (beyond simple hyperparameter importance and effects) and then develop appropriate interpretability methods accordingly. To this end, it might be worthwhile exploring the combination of local concepts of interpretability like counterfactuals with AutoML, which may help answer questions such as what is the minimum change required in a learning algorithm to achieve a desired level of reliability or interpretability or to eliminate a specific unintended causal relationship in the final model.

Moreover, the interpretations of algorithm behavior generated by interpretability methods can still be highly technical and difficult to process, even for experts. In many cases, it may still require considerable human effort to examine a multitude of hyperparameter importance scores and effect plots and draw meaningful conclusions. To overcome this challenge, exploring how various interpretability metrics can be transformed into more semantically meaningful explanations would be interesting. Usability studies should be conducted to ensure that the resulting explanations are sufficiently clear and provide answers to the relevant questions.

Most of the current research on algorithm explainability has focused on simple and well-behaved problems, such as interpreting hyperparameter importance or effects when there is a small number of numeric hyperparameters only (Hutter et al., 2014; Biedenkapp et al., 2017; Moosbauer et al., 2021). Many of the existing methods use surrogate models to fit an archive of sampled hyperparameter configurations, which may work well for low-dimensional and mostly numeric hyperparameter spaces but pose challenges for hyperparameter spaces with high complexity, such as those encountered in NAS with a multitude of categorical hyperparameters with dependencies between them. In addition, as the number of dimensions of the hyperparameter space increases, extracting meaningful patterns may become more challenging, and it may be worth exploring interpretability methods that leverage dimensionality reduction techniques. Overall, developing concepts and solutions for interpreting complex and high-dimensional HPO problems is an important avenue for future research.

AutoML system explainability Given the no-free lunch theorem, a potential research direction is investigating which AutoML systems are best suited for which types of tasks. Specifically, it is worth examining which AutoML systems are better suited for specific application domains such as healthcare or finance and for varying dataset sizes. For instance, it could be that an AutoML system performs well in healthcare domains due to the inclusion of a specific learning algorithm that excels at tasks within this field into the pipeline, which could be inferred from data on the performance of different AutoML systems on a variety of datasets. Additionally, it may be the case that an AutoML system is well-suited for larger datasets due to the presence of a specific dimensionality reduction operator within the pipeline, which is consistently selected for larger datasets. Investigating the reasons behind the suitability of different AutoML systems for different tasks and datasets would be a valuable area of research.

Efforts have been made towards a unified way of benchmarking AutoML systems against each other (Gijsbers et al., 2022). However, a standardized approach for conducting ablation analyses is needed to develop a valid, common understanding of the factors driving the superior performance of one AutoML system over another. Ablation analysis is particularly important when new AutoML systems claim to surpass state of the art, as it provides evidence that the changes introduced by the new system significantly contribute to improved performance. A harmonized framework, in which different configurations correspond to different AutoML tools and the newly introduced AutoML systems (as target configuration) could be compared against the previous state of the art (as source configuration), could facilitate effective ablation analyses for AutoML systems. A contribution to this thesis (Moosbauer et al., 2022) (also discussed in Chapter 4.2) represents a first step in this direction, as it provides a standardized framework covering a range of HPO algorithms that are used for ablation analysis. Still, a standardized approach to performing ablation analyses should be adopted across the community.

In addition to elucidating the inner workings of classical AutoML systems, investigating the explainability of meta-learning-based systems could be an exciting direction for further research. Extending upon previous works such as (van Rijn and Hutter, 2018; Moussa et al., 2022), which delve into the importance of hyperparameters across datasets, it would be interesting to explore the specific dataset properties that meta-learning algorithms exploit for better performance and faster results. Such an investigation could yield valuable insights into the underlying mechanisms of these systems and contribute to their further development and refinement.

Lastly, when conducting benchmarks and sensitivity analyses, it is important to consider both performance metrics and secondary metrics, such as an AutoML system's ability to produce robust, fair, and explainable models, among others. Such metrics can provide valuable insights into the broader implications and impact of AutoML systems beyond performance and contribute to developing more comprehensive and responsible approaches to AutoML.

Contributing Publications

Binder*, M., Moosbauer*, J., Thomas, J., Bischl, B. (2020). Multi-objective hyperparameter tuning and feature selection using filter ensembles. *Proceedings of the 2020 Genetic and Evolutionary Computation Conference (GECCO '20), pp. 471–479. ACM.*

Karl*, F., Pielok*, T., Moosbauer, J., Pfisterer, F., Coors, S., Binder, M., Schneider, L., Thomas, J., Richter, J., Lang, M., Garrido-Merchán, E., Branke, J., Bischl, B. (2022). Multi-Objective Hyperparameter Optimization – An Overview. *arXiv preprint arXiv:2206.07438*. Under review (ACM Transactions on Evolutionary Learning and Optimization).

Moosbauer*, J., Herbinger*, J., Casalicchio, G., Lindauer, M., Bischl, B. (2021). Explaining Hyperparameter Optimization via Partial Dependence Plots. *Advances in Neural Information Processing Systems 34 (NeurIPS 2021), pp. 2280–2291.*

Moosbauer, J., Casalicchio, G., Lindauer, M., Bischl, B. (2023). Improving Accuracy of Interpretability Measures in Hyperparameter Optimization via Bayesian Algorithm Execution. *arXiv preprint arXiv:2206.05447.*

Pfisterer, F., Schneider, L., Moosbauer, J., and Binder, M., Bischl, B. (2022). YAHPO Gym - An Efficient Multi-Objective Multi-Fidelity Benchmark for Hyperparameter Optimization. *Proceedings of the First International Conference on Automated Machine Learning, Proceedings of Machine Learning Research vol. 188, pp. 3/1–39.*

Moosbauer, J., Binder, M., Schneider, L., Pfisterer, F., Becker, M., Lang, M., Kotthoff, L., Bischl., B. (2022). Automated Benchmark-Driven Design and Explanation of Hyperparameter Optimizers. *IEEE Transactions on Evolutionary Computation, vol. 26, no. 6, pp. 1336–1350.*

Further References

Akhtar, N. and A. S. Mian (2018). Threat of adversarial attacks on deep learning in computer vision: A survey. *IEEE Access 6*, 14410–14430.

Akiba, T., S. Sano, T. Yanase, T. Ohta, and M. Koyama (2019). Optuna: A next-generation hyperparameter optimization framework. In A. Teredesai, V. Kumar, Y. Li, R. Rosales, E. Terzi, and G. Karypis (Eds.), *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*, pp. 2623–2631. ACM.

Alaa, A. and M. van der Schaar (2018, 10–15 Jul). AutoPrognosis: Automated clinical prognostic modeling via Bayesian optimization with structured kernel learning. In J. Dy and A. Krause (Eds.), *Proceedings of the 35th International Conference on Machine Learning*, Volume 80 of *Proceedings of Machine Learning Research*, pp. 139–148. PMLR.

Alaa, A. M., T. Bolton, E. Di Angelantonio, J. H. Rudd, and M. Van der Schaar (2019). Cardiovascular disease risk prediction using automated machine learning: A prospective study of 423,604 uk biobank participants. *PloS one 14*(5), e0213653.

Ardila, D., A. P. Kiraly, S. Bharadwaj, B. Choi, J. J. Reicher, L. Peng, D. Tse, M. Etemadi, W. Ye, G. Corrado, et al. (2019). End-to-end lung cancer screening with three-dimensional deep learning on low-dose chest computed tomography. *Nature medicine 25*(6), 954–961.

Baloglu, U. B., M. Talo, Ö. Yildirim, R. S. Tan, and U. R. Acharya (2019). Classification of myocardial infarction with multi-lead ECG signals and deep CNN. *Pattern Recognit. Lett. 122*, 23–30.

Baskarada, S. and A. Koronios (2017). Unicorn data scientist: the rarest of breeds. *Program Electronic Library and Information Systems 51*(1), 65–74.

Belakaria, S., A. Deshwal, and J. R. Doppa (2019). Max-value entropy search for multi-objective bayesian optimization. In H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett (Eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 7823–7833.

Bergstra, J., R. Bardenet, Y. Bengio, and B. Kégl (2011). Algorithms for hyper-parameter optimization. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. C. N. Pereira, and K. Q. Weinberger (Eds.), *Advances in Neural Information Processing Systems 24: Annual Conference on Neural Information Processing Systems 2011, NeurIPS 2011, December 12-14, 2011, Granada, Spain*, pp. 2546–2554.

Bergstra, J. and Y. Bengio (2012). Random search for hyper-parameter optimization. *J. Mach. Learn. Res. 13*, 281–305.

Biedenkapp, A., M. Lindauer, K. Eggensperger, F. Hutter, C. Fawcett, and H. H. Hoos (2017). Efficient parameter importance analysis via ablation with surrogates. In S. Singh and S. Markovitch (Eds.), *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pp. 773–779. AAAI Press.

Biedenkapp, A., J. Marben, M. Lindauer, and F. Hutter (2018). CAVE: configuration assessment, visualization and evaluation. In R. Battiti, M. Brunato, I. S. Kotsireas, and P. M. Pardalos (Eds.), *Learning and Intelligent Optimization - 12th International Conference, LION 12, Kalamata, Greece, June 10-15, 2018, Revised Selected Papers*, Volume 11353 of *Lecture Notes in Computer Science*, pp. 115–130. Springer.

Bischl, B., M. Binder, M. Lang, T. Pielok, J. Richter, S. Coors, J. Thomas, T. Ullmann, M. Becker, A. Boulesteix, D. Deng, and M. Lindauer (2021). Hyperparameter optimization: Foundations, algorithms, best practices and open challenges. *CoRR abs/2107.05847*.

Breiman, L. (1996). Bagging predictors. *Mach. Learn. 24*(2), 123–140.

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

Bromberger, S. (1992). *On what we know we don't know: Explanation, theory, linguistics, and how questions shape them.* University of Chicago Press.

Cai, C. J., J. Jongejan, and J. Holbrook (2019). The effects of example-based explanations in a machine learning interface. In W. Fu, S. Pan, O. Brdiczka, P. Chau, and G. Calvary (Eds.), *Proceedings of the 24th International Conference on Intelligent User Interfaces, IUI 2019, Marina del Ray, CA, USA, March 17-20, 2019*, pp. 258–262. ACM.

Carvalho, D. V., E. M. Pereira, and J. S. Cardoso (2019). Machine learning interpretability: A survey on methods and metrics. *Electronics 8*(8).

Chen, T. and C. Guestrin (2016). Xgboost: A scalable tree boosting system. In B. Krishnapuram, M. Shah, A. J. Smola, C. C. Aggarwal, D. Shen, and R. Rastogi (Eds.), *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pp. 785–794. ACM.

Conrad, F., M. Mälzer, M. Schwarzenberger, H. Wiemer, and S. Ihlenfeldt (2022, Nov). Benchmarking automl for regression tasks on small tabular data in materials design. *Scientific Reports 12*.

Coors, S., D. Schalk, B. Bischl, and D. Rügamer (2021). Automatic componentwise boosting: An interpretable automl system. *CoRR abs/2109.05583*.

Corbett-Davies, S. and S. Goel (2018). The measure and mismeasure of fairness: A critical review of fair machine learning. *CoRR abs/1808.00023*.

Czitrom, V. (1999). One-factor-at-a-time versus designed experiments. *The American Statistician 53*(2), 126–131.

Dandl, S., C. Molnar, M. Binder, and B. Bischl (2020). Multi-objective counterfactual explanations. In T. Bäck, M. Preuss, A. H. Deutz, H. Wang, C. Doerr, M. T. M. Emmerich, and H. Trautmann (Eds.), *Parallel Problem Solving from Nature - PPSN XVI - 16th International Conference, PPSN 2020, Leiden, The Netherlands, September 5-9, 2020, Proceedings, Part I*, Volume 12269 of *Lecture Notes in Computer Science*, pp. 448–469. Springer.

Deb, K., A. Pratap, S. Agarwal, and T. Meyarivan (2002). A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation 6*(2), 182–197.

Díez, J., O. Luaces, J. J. del Coz, and A. Bahamonde (2015). Optimizing different loss functions in multilabel classifications. *Prog. Artif. Intell. 3*(2), 107–118.

Doran, D., S. Schulz, and T. R. Besold (2017). What does explainable AI really mean? A new conceptualization of perspectives. In T. R. Besold and O. Kutz (Eds.), *Proceedings of the First International Workshop on Comprehensibility and Explanation in AI and ML 2017 co-located with 16th International Conference of the Italian Association for Artificial Intelligence (AI*IA 2017), Bari, Italy, November 16th and 17th, 2017*, Volume 2071 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Doshi-Velez, F. and B. Kim (2017). Towards a rigorous science of interpretable machine learning. *CoRR abs/1702.08608*.

Doshi-Velez, F. and B. Kim (2020). Auto-sklearn 2.0: Hands-free automl via meta-learning. *CoRR abs/2007.04074*.

Drozdal, J., J. D. Weisz, D. Wang, G. Dass, B. Yao, C. Zhao, M. J. Muller, L. Ju, and H. Su (2020). Trust in automl: exploring information needs for establishing trust in automated machine learning systems. In F. Paternò, N. Oliver, C. Conati, L. D. Spano, and N. Tintarev (Eds.), *IUI '20: 25th International Conference on Intelligent User Interfaces, Cagliari, Italy, March 17-20, 2020*, pp. 297–307. ACM.

Elsken, T., J. H. Metzen, and F. Hutter (2019). Neural architecture search: A survey. *J. Mach. Learn. Res. 20*, 55:1–55:21.

Erickson, N., J. Mueller, A. Shirkov, H. Zhang, P. Larroy, M. Li, and A. J. Smola (2020). Autogluon-tabular: Robust and accurate automl for structured data. *CoRR abs/2003.06505*.

Falkner, S., A. Klein, and F. Hutter (2018). BOHB: robust and efficient hyperparameter optimization at scale. In J. G. Dy and A. Krause (Eds.), *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, Volume 80 of *Proceedings of Machine Learning Research*, pp. 1436–1445. PMLR.

Fawcett, C. and H. H. Hoos (2016). Analysing differences between algorithm configurations through ablation. *J. Heuristics 22*(4), 431–458.

Feurer, M. and F. Hutter (2019). Hyperparameter optimization. In F. Hutter, L. Kotthoff, and J. Vanschoren (Eds.), *Automated Machine Learning - Methods, Systems, Challenges*, The Springer Series on Challenges in Machine Learning, pp. 3–33. Springer.

Feurer, M., A. Klein, K. Eggensperger, J. T. Springenberg, M. Blum, and F. Hutter (2015). Efficient and robust automated machine learning. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (Eds.), *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pp. 2962–2970.

Franceschi, L., P. Frasconi, S. Salzo, R. Grazzi, and M. Pontil (2018). Bilevel programming for hyperparameter optimization and meta-learning. In J. G. Dy and A. Krause (Eds.), *Proceedings* *of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, Volume 80 of *Proceedings of Machine Learning Research*, pp. 1563–1572. PMLR.

Freitas, A. A. (2019). Automated machine learning for studying the trade-off between predictive accuracy and interpretability. In A. Holzinger, P. Kieseberg, A. M. Tjoa, and E. R. Weippl (Eds.), *Machine Learning and Knowledge Extraction - Third IFIP TC 5, TC 12, WG 8.4, WG 8.9, WG 12.9 International Cross-Domain Conference, CD-MAKE 2019, Canterbury, UK, August 26-29, 2019, Proceedings*, Volume 11713 of *Lecture Notes in Computer Science*, pp. 48–66. Springer.

Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 1189–1232.

Garouani, M., A. Ahmad, M. Bouneffa, A. Lewandowski, G. Bourguin, and M. Hamlich (2021). Towards the automation of industrial data science: A meta-learning based approach. In J. Filipe, M. Smialek, A. Brodsky, and S. Hammoudi (Eds.), *Proceedings of the 23rd International Conference on Enterprise Information Systems, ICEIS 2021, Online Streaming, April 26-28, 2021, Volume 1*, pp. 709–716. SCITEPRESS.

Gijsbers, P., M. L. P. Bueno, S. Coors, E. LeDell, S. Poirier, J. Thomas, B. Bischl, and J. Vanschoren (2022). AMLB: an automl benchmark. *CoRR abs/2207.12560.*

Gijsbers, P. and J. Vanschoren (2019). Gama: Genetic automated machine learning assistant. *Journal of Open Source Software 4*(33), 1132.

Gilpin, L. H., D. Bau, B. Z. Yuan, A. Bajwa, M. A. Specter, and L. Kagal (2018). Explaining explanations: An overview of interpretability of machine learning. In F. Bonchi, F. J. Provost, T. Eliassi-Rad, W. Wang, C. Cattuto, and R. Ghani (Eds.), *5th IEEE International Conference on Data Science and Advanced Analytics, DSAA 2018, Turin, Italy, October 1-3, 2018*, pp. 80–89. IEEE.

Goldstein, A., A. Kapelner, J. Bleich, and E. Pitkin (2015). Peeking inside the black box: Visualizing statistical learning with plots of individual conditional expectation. *Journal of Computational and Graphical Statistics 24*(1), 44–65.

Golovin, D., B. Solnik, S. Moitra, G. Kochanski, J. Karro, and D. Sculley (2017). Google vizier: A service for black-box optimization. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, August 13 - 17, 2017*, pp. 1487–1495. ACM.

Guyon, I., A. Saffari, G. Dror, and G. C. Cawley (2010). Model selection: Beyond the bayesian/frequentist divide. *J. Mach. Learn. Res. 11*, 61–87.

Hamon, R., H. Junklewitz, and I. Sanchez (2020). Robustness and explainability of artificial intelligence. (KJ-NA-30040-EN-N (online)).

Hansen, N. and A. Auger (2011). Cma-es: Evolution strategies and covariance matrix adaptation. In *Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation*, GECCO '11, New York, NY, USA, pp. 991–1010. Association for Computing Machinery.

Hasebrook, N., F. Morsbach, N. Kannengießer, J. K. H. Franke, F. Hutter, and A. Sunyaev (2022). Why do machine learning practitioners still use manual tuning? A qualitative study. *CoRR abs/2203.01717*.

Hennig, P. and C. J. Schuler (2012). Entropy search for information-efficient global optimization. *J. Mach. Learn. Res. 13*, 1809–1837.

Hernández-Lobato, J. M., M. W. Hoffman, and Z. Ghahramani (2014). Predictive entropy search for efficient global optimization of black-box functions. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger (Eds.), *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pp. 918–926.

Hutter, F., H. H. Hoos, and K. Leyton-Brown (2011). Sequential model-based optimization for general algorithm configuration. In C. A. C. Coello (Ed.), *Learning and Intelligent Optimization - 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers*, Volume 6683 of *Lecture Notes in Computer Science*, pp. 507–523. Springer.

Hutter, F., H. H. Hoos, and K. Leyton-Brown (2014). An efficient approach for assessing hyperparameter importance. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, Volume 32 of *JMLR Workshop and Conference Proceedings*, pp. 754–762. JMLR.org.

Hutter, F., L. Kotthoff, and J. Vanschoren (Eds.) (2019). *Automated Machine Learning - Methods, Systems, Challenges.* The Springer Series on Challenges in Machine Learning. Springer.

Ikemura, K., E. Bellin, Y. Yagi, H. Billett, M. Saada, K. Simone, L. Stahl, J. Szymanski, D. Goldstein, M. R. Gil, et al. (2021). Using automated machine learning to predict the mortality of patients with covid-19: prediction model development study. *Journal of medical Internet research 23*(2), e23458.

Iooss, B. and P. Lemaître (2015). A review on global sensitivity analysis methods. *Uncertainty management in simulation-optimization of complex systems*, 101–122.

Jamieson, K. G. and A. Talwalkar (2016). Non-stochastic best arm identification and hyperparameter optimization. In A. Gretton and C. C. Robert (Eds.), *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, AISTATS 2016, Cadiz, Spain, May 9-11, 2016*, Volume 51 of *JMLR Workshop and Conference Proceedings*, pp. 240–248. JMLR.org.

Jin, H., Q. Song, and X. Hu (2019). Auto-keras: An efficient neural architecture search system. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1946–1956. ACM.

Jones, D. R. (2001). A taxonomy of global optimization methods based on response surfaces. *Journal of Global Optimization 21*(4), 345–383.

Jones, D. R., M. Schonlau, and W. J. Welch (1998). Efficient global optimization of expensive black-box functions. *J. Glob. Optim. 13*(4), 455–492.

Kapishnikov, A., T. Bolukbasi, F. B. Viégas, and M. Terry (2019). XRAI: better attributions through regions. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pp. 4947–4956. IEEE.

Karaglani, M., K. Gourlia, I. Tsamardinos, and E. Chatzaki (2020). Accurate blood-based diagnostic biosignatures for alzheimer's disease via automated machine learning. *Journal of Clinical Medicine 9*(9).

Knowles, J. D. (2006). Parego: a hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. *IEEE Trans. Evol. Comput. 10*(1), 50–66.

Komer, B., J. Bergstra, and C. Eliasmith (2019). Hyperopt-sklearn. In F. Hutter, L. Kotthoff, and J. Vanschoren (Eds.), *Automated Machine Learning - Methods, Systems, Challenges*, The Springer Series on Challenges in Machine Learning, pp. 97–111. Springer.

Krauß, J., B. M. Pacheco, H. M. Zang, and R. H. Schmitt (2020). Automated machine learning for predictive quality in production. *Procedia CIRP 93*, 443–448.

Le, T. T., W. Fu, and J. H. Moore (2020). Scaling tree-based automated machine learning to biomedical big data with a feature set selector. *Bioinformatics 36*(1), 250–256.

LeDell, E. and S. Poirier (2020, July). H2O AutoML: Scalable automatic machine learning. *7th ICML Workshop on Automated Machine Learning (AutoML).*

Lee, C., C. Lee, T. Ha, and S. Cho (2022). Survey: Strategies for loss-based discrete-time hazard and survival function estimation. In *13th International Conference on Information and Communication Technology Convergence, ICTC 2022, Jeju Island, Korea, Republic of, October 19-21, 2022*, pp. 844–846. IEEE.

Li, L., K. G. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar (2017). Hyperband: Bandit-based configuration evaluation for hyperparameter optimization. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings.* OpenReview.net.

Lindauer, M., K. Eggensperger, M. Feurer, A. Biedenkapp, D. Deng, C. Benjamins, T. Ruhkopf, R. Sass, and F. Hutter (2022). Smac3: A versatile bayesian optimization package for hyperparameter optimization. *Journal of Machine Learning Research 23*(54), 1–9.

Lindauer, M., M. Feurer, K. Eggensperger, A. Biedenkapp, and F. Hutter (2019). Towards assessing the impact of bayesian optimization's own hyperparameters. *CoRR abs/1908.06674.*

Lipton, Z. C. (2018). The mythos of model interpretability. *Commun. ACM 61*(10), 36–43.

Loshchilov, I. and F. Hutter (2016). CMA-ES for hyperparameter optimization of deep neural networks. *CoRR abs/1604.07269.*

Lundberg, S. M. and S. Lee (2017). A unified approach to interpreting model predictions. In I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett (Eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 4765–4774.

Mehrabi, N., F. Morstatter, N. Saxena, K. Lerman, and A. Galstyan (2021). A survey on bias and fairness in machine learning. *ACM Comput. Surv. 54*(6), 115:1–115:35.

Miller, T. (2019). Explanation in artificial intelligence: Insights from the social sciences. *Artif. Intell. 267*, 1–38.

Mockus, J. (1974). On bayesian methods for seeking the extremum. In G. I. Marchuk (Ed.), *Optimization Techniques, IFIP Technical Conference, Novosibirsk, USSR, July 1-7, 1974*, Volume 27 of *Lecture Notes in Computer Science*, pp. 400–404. Springer.

Mohr, F., M. Wever, and E. Hüllermeier (2018). Ml-plan: Automated machine learning via hierarchical planning. *Machine Learning 107*(8-10), 1495–1515.

Molnar, C. (2022). *Interpretable Machine Learning* (2 ed.).

Molnar, C., G. Casalicchio, and B. Bischl (2019). Quantifying model complexity via functional decomposition for better post-hoc interpretability. In P. Cellier and K. Driessens (Eds.), *Machine Learning and Knowledge Discovery in Databases - International Workshops of ECML PKDD 2019, Würzburg, Germany, September 16-20, 2019, Proceedings, Part I*, Volume 1167 of *Communications in Computer and Information Science*, pp. 193–204. Springer.

Molnar, C., G. König, J. Herbinger, T. Freiesleben, S. Dandl, C. A. Scholbeck, G. Casalicchio, M. Grosse-Wentrup, and B. Bischl (2020). General pitfalls of model-agnostic interpretation methods for machine learning models. In A. Holzinger, R. Goebel, R. Fong, T. Moon, K. Müller, and W. Samek (Eds.), *xxAI - Beyond Explainable AI - International Workshop, Held in Conjunction with ICML 2020, July 18, 2020, Vienna, Austria, Revised and Extended Papers*, Volume 13200 of *Lecture Notes in Computer Science*, pp. 39–68. Springer.

Moosbauer, J., M. Binder, L. Schneider, F. Pfisterer, M. Becker, M. Lang, L. Kotthoff, and B. Bischl (2022). Automated benchmark-driven design and explanation of hyperparameter optimizers. *IEEE Trans. Evol. Comput. 26*(6), 1336–1350.

Moosbauer, J., J. Herbinger, G. Casalicchio, M. Lindauer, and B. Bischl (2021). Explaining hyperparameter optimization via partial dependence plots. In M. Ranzato, A. Beygelzimer, Y. N. Dauphin, P. Liang, and J. W. Vaughan (Eds.), *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, Virtual Event*, pp. 2280–2291.

Moussa, C., J. N. van Rijn, T. Bäck, and V. Dunjko (2022). Hyperparameter importance of quantum neural networks across small datasets. In P. Pascal and D. Ienco (Eds.), *Discovery Science - 25th International Conference, DS 2022, Montpellier, France, October 10-12, 2022, Proceedings*, Volume 13601 of *Lecture Notes in Computer Science*, pp. 32–46. Springer.

Mustafa, A. and M. R. Azghadi (2021). Automated machine learning for healthcare and clinical notes analysis. *Comput. 10*(2), 24.

OECD (1998). *21st Century Technologies.*

Ono, J. P., S. Castelo, R. Lopez, E. Bertini, J. Freire, and C. T. Silva (2021). Pipelineprofiler: A visual analytics tool for the exploration of automl pipelines. *IEEE Trans. Vis. Comput. Graph. 27*(2), 390–400.

Park, A., C. Chute, P. Rajpurkar, J. Lou, R. L. Ball, K. Shpanskaya, R. Jabarkheel, L. H. Kim, E. McKenna, J. Tseng, J. Ni, F. Wishah, F. Wittber, D. S. Hong, T. J. Wilson, S. Halabi, S. Basu, B. N. Patel, M. P. Lungren, A. Y. Ng, and K. W. Yeom (2019, 06). Deep Learning–Assisted Diagnosis of Cerebral Aneurysms Using the HeadXNet Model. *JAMA Network Open 2*(6), e195600–e195600.

Park, H., Y. Nam, J. Kim, and J. Choo (2021). Hypertendril: Visual analytics for user-driven hyperparameter optimization of deep neural networks. *IEEE Trans. Vis. Comput. Graph. 27*(2), 1407–1416.

Pfisterer, F., S. Coors, J. Thomas, and B. Bischl (2019). Multi-objective automatic machine learning with autoxgboostmc. *CoRR abs/1908.10796.*

Płońska, A. and P. Płoński (2021). Mljar: State-of-the-art automated machine learning framework for tabular data. version 0.10.3.

Ponweiser, W., T. Wagner, D. Biermann, and M. Vincze (2008). Multiobjective optimization on a limited budget of evaluations using model-assisted -metric selection. In G. Rudolph, T. Jansen, S. M. Lucas, C. Poloni, and N. Beume (Eds.), *Parallel Problem Solving from Nature - PPSN X, 10th International Conference Dortmund, Germany, September 13-17, 2008, Proceedings*, Volume 5199 of *Lecture Notes in Computer Science*, pp. 784–794. Springer.

Portugal, I., P. S. C. Alencar, and D. D. Cowan (2018). The use of machine learning algorithms in recommender systems: A systematic review. *Expert Syst. Appl. 97*, 205–227.

Probst, P., A. Boulesteix, and B. Bischl (2019). Tunability: Importance of hyperparameters of machine learning algorithms. *J. Mach. Learn. Res. 20*, 53:1–53:32.

Rasmussen, C. E. (2003). Gaussian processes in machine learning. In *Summer school on machine learning*, pp. 63–71. Springer.

Real, E., C. Liang, D. R. So, and Q. V. Le (2020). Automl-zero: Evolving machine learning algorithms from scratch. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, Volume 119 of *Proceedings of Machine Learning Research*, pp. 8007–8019. PMLR.

Rudin, C. (2019). Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nat. Mach. Intell. 1*(5), 206–215.

Saltelli, A., M. Ratto, T. Andres, F. Campolongo, J. Cariboni, D. Gatelli, M. Saisana, and S. Tarantola (2008). *Global Sensitivity Analysis: The Primer.* Wiley.

Santurkar, S., D. Tsipras, A. Ilyas, and A. Madry (2018). How does batch normalization help optimization? In S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.), *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pp. 2488–2498.

Sass, R., E. Bergman, A. Biedenkapp, F. Hutter, and M. Lindauer (2022). Deepcave: An interactive analysis tool for automated machine learning. *CoRR abs/2206.03493.*

Scher, S. and A. Trügler (2022). Robustness of machine learning models beyond adversarial attacks. *CoRR abs/2204.10046.*

Schmid, M. and T. Hothorn (2008). Boosting additive models using component-wise p-splines. *Comput. Stat. Data Anal. 53*(2), 298–311.

Serband, A., K. van der Blom, H. Hoos, and J. Visser (2020). The 2020 state of automl adoption.

Shalev-Shwartz, S. and S. Ben-David (2014). *Understanding Machine Learning - From Theory to Algorithms.* Cambridge University Press.

Snoek, J., H. Larochelle, and R. P. Adams (2012). Practical bayesian optimization of machine learning algorithms. In F. Pereira, C. Burges, L. Bottou, and K. Weinberger (Eds.), *Advances in Neural Information Processing Systems*, Volume 25. Curran Associates, Inc.

Song, J., L. Yu, W. Neiswanger, and S. Ermon (2022, 17–23 Jul). A general recipe for likelihood-free Bayesian optimization. In K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato (Eds.), *Proceedings of the 39th International Conference on Machine Learning*, Volume 162 of *Proceedings of Machine Learning Research*, pp. 20384–20404. PMLR.

Sovrano, F. and F. Vitali (2021). An objective metric for explainable AI: how and why to estimate the degree of explainability. *CoRR abs/2109.05327*.

Srinivas, N., A. Krause, S. M. Kakade, and M. W. Seeger (2010). Gaussian process optimization in the bandit setting: No regret and experimental design. In J. Fürnkranz and T. Joachims (Eds.), *Proceedings of the 27th International Conference on Machine Learning*, pp. 1015–1022. Omnipress.

Stacke, K., G. Eilertsen, J. Unger, and C. Lundström (2021). Measuring domain shift for deep learning in histopathology. *IEEE J. Biomed. Health Informatics 25*(2), 325–336.

Strobl, C., A. Boulesteix, T. Kneib, T. Augustin, and A. Zeileis (2008). Conditional variable importance for random forests. *BMC Bioinform. 9*.

Sundararajan, M., A. Taly, and Q. Yan (2017). Axiomatic attribution for deep networks. In D. Precup and Y. W. Teh (Eds.), *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, Volume 70 of *Proceedings of Machine Learning Research*, pp. 3319–3328. PMLR.

Swearingen, T., W. Drevo, B. Cyphers, A. Cuesta-Infante, A. Ross, and K. Veeramachaneni (2017). ATM: A distributed, collaborative, scalable system for automated machine learning. In *2017 IEEE International Conference on Big Data, BigData 2017, Boston, MA, USA, December 11-14, 2017*, pp. 151–162.

Thornton, C., F. Hutter, H. H. Hoos, and K. Leyton-Brown (2013). Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *Proc. of KDD-2013*, pp. 847–855.

Turner, R., D. Eriksson, M. McCourt, J. Kiili, E. Laaksonen, Z. Xu, and I. Guyon (2021, 06–12 Dec). Bayesian optimization is superior to random search for machine learning hyperparameter tuning: Analysis of the black-box optimization challenge 2020. In H. J. Escalante and K. Hofmann (Eds.), *Proceedings of the NeurIPS 2020 Competition and Demonstration Track*, Volume 133 of *Proceedings of Machine Learning Research*, pp. 3–26. PMLR.

Vakhrushev, A., A. Ryzhkov, M. Savchenko, D. Simakov, R. Damdinov, and A. Tuzhilin (2021). Lightautoml: Automl solution for a large financial services ecosystem. *CoRR abs/2109.01528*.

Vamathevan, J., D. Clark, P. Czodrowski, I. Dunham, E. Ferran, G. Lee, B. Li, A. Madabhushi, P. Shah, M. Spitzer, and S. Zhao (2019, Jun). Applications of machine learning in drug discovery and development. *Nature Reviews Drug Discovery 18*(6), 463–477.

van Rijn, J. N. and F. Hutter (2018). Hyperparameter importance across datasets. In Y. Guo and F. Farooq (Eds.), *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018*, pp. 2367–2376. ACM.

Vanschoren, J. (2018). Meta-learning: A survey. *CoRR abs/1810.03548*.

Vanschoren, J., J. N. van Rijn, B. Bischl, and L. Torgo (2013). Openml: networked science in machine learning. *SIGKDD Explor. 15*(2), 49–60.

Wachter, S., B. D. Mittelstadt, and C. Russell (2017). Counterfactual explanations without opening the black box: Automated decisions and the GDPR. *CoRR abs/1711.00399*.

Wang, C., Q. Wu, M. Weimer, and E. Zhu (2021). Flaml: A fast and lightweight automl library. In *MLSys*.

Weidele, D. K. I., J. D. Weisz, E. Oduor, M. J. Muller, J. Andres, A. G. Gray, and D. Wang (2020). Autoaiviz: opening the blackbox of automated artificial intelligence with conditional parallel coordinates. In F. Paternò, N. Oliver, C. Conati, L. D. Spano, and N. Tintarev (Eds.), *IUI '20: 25th International Conference on Intelligent User Interfaces, Cagliari, Italy, March 17-20, 2020*, pp. 308–312. ACM.

Wolpert, D. H. and W. G. Macready (1997). No free lunch theorems for optimization. *IEEE Trans. Evol. Comput. 1*(1), 67–82.

Wu, J., M. Poloczek, A. G. Wilson, and P. I. Frazier (2017). Bayesian optimization with gradients. In *Advances in Neural Information Processing Systems 30*, pp. 5267–5278.

Yang, F., Y. Qiao, Y. Qi, J. Bo, and X. Wang (2022). Bacs: blockchain and automl-based technology for efficient credit scoring classification. *Annals of Operations Research*, 1–21.

Zimmer, L., M. Lindauer, and F. Hutter (2021). Auto-pytorch: Multi-fidelity metalearning for efficient and robust autodl. *IEEE Trans. Pattern Anal. Mach. Intell. 43*(9), 3079–3090.

Zöller, M. and M. F. Huber (2021). Benchmark and survey of automated machine learning frameworks. *J. Artif. Intell. Res. 70*, 409–472.

Zöller, M., W. Titov, T. Schlegel, and M. F. Huber (2022). Xautoml: A visual analytics tool for establishing trust in automated machine learning. *CoRR abs/2202.11954*.

Zonta, T., C. A. da Costa, R. da Rosa Righi, M. J. de Lima, E. S. da Trindade, and G. Li (2020). Predictive maintenance in the industry 4.0: A systematic literature review. *Comput. Ind. Eng. 150*, 106889.

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 22.02.2023

Julia Moosbauer