Machine Learning Model Selection with Multi-Objective Bayesian Optimization and Reinforcement Learning

Case studies on functional data analysis, pipeline tuning and shifted distribution

Xudong Sun

11. April 2021

Machine Learning Model Selection with Multi-Objective Bayesian Optimization and Reinforcement Learning

Case studies on functional data analysis, pipeline tuning and shifted distribution

Xudong Sun

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

vorgelegt von
Xudong Sun
aus Qingdao, Shandong, China

München, den 11. April 2021

Erstgutachter: Bernd Bischl
Zweitgutachter: Asja Fischer
Drittergutachter: Fabian Scheipl
Tag der mündlichen Prüfung: 30. September 2021

Eidesstattliche Versicherung

(Siehe Promotionsordnung vom 12.07.11, § 8, Abs. 2 Pkt. .5.)

Hiermit erkläre ich an Eides statt, dass die Dissertation von mir selbstständig, ohne unerlaubte Beihilfe angefertigt ist.

München, 05. Oct 2021

Ort, Datum

Unterschrift

Xudong Sun

Acknowledgement

This dissertation would not have been possible without the support of many people, In particular, my sincere gratitude goes to ...

... *My doctor supervisor Prof. Dr. Bernd Bischl for offering me the position, guiding me into the field of multi-objective bayesian optimization for algorithm configuration and machine learning, training me on software engineering skills, his support throughout the years, many advices on projects and granting me freedom to delve into various research areas at later stage.*

... *Prof. Dr. Asja Fischer and PD Dr. Fabian Scheipl for their willingness to act as the second and third reviewer for my Ph.D. dissertation.*

... *Prof. Dr. Christian Heumann and Prof. Dr. Volker Schmid for their availability to be part of the examination panel at my Ph.D. defense.*

... *Anna and her colleagues for their support, trust and patience during the collaboration project.*

... *Dr. Michel Lang, Prof. Dr. Joerg Rahenfuerher for kindly admitting me into the collaboration project, for their kindness, support and insightful advices. To Andrea Bommert during the project collaboration, where I learned from her high quality work attitude and skill.*

... *Janek for helping me to get started with benchmarking on the computation cluster with R and offering research advices and many chances to give talks in datageeks, and Giuseppe for offering me support in the work, giving advices to research, offering this latex template. To Laura who generously shared her research experience, for her efforts, collaboration and patience on the joint project with me. To the* `mlr` *development team with Jakob, Julia and many others.*

... *Quay for his help on my work, teaching me to cook, for his life advices, his help on my German language and generously sharing his funding with me.*

... *Andreas, Micha, Alex, Shuai, Alex, Philipp for accompanying me for late afternoon lunches, going to sport together, eating out during the weekends, cooking together.*

... *Briggite, Pia, Elke for their support for various administrative matters.*

... *David, Sarah, Fabian, Martin, Eva, Almond for generously sharing their research experience. Special thanks to David for his encouragement on finishing my dissertation.*

... *Florian, Christoph, Susanne for the nice office atmosphere, for their help in the everyday work and life. Special thanks to Florian for the active and helpful project discussions in the office.*

... *Jiali, Sebastian, Alexej, for the great project collaboration, inspiring discussions, as well as the nice experience with Michael, Markus. Special thanks to Jiali for her dedication, creativity, admirable work attitude for the nice collaboration. To Martin who helped us with using mlrCPO. Nice discussion with Emilio, Matthias on variational inference and topic modeling, with Alex and Gunnar on Causality, with Jann on Neural Process, special thanks to Alex on his help on my knowledge about causality. To Yu for doing the research assistant job, Tieu-Binh for checking the language of the ReinBo paper.*

... *For Nutan on his generous experience sharing, his encouragement all the time.*

... *Florian, Ulli, Denis for their trust and empowerment, time investment on me, George for his encouragement, Ingo, Ralf, Fabian and Matthias for the time investment, nice discussion with Phillip, Daniel, Steffen, nice time spent together with the residents Doris and Dominik, nice time together with Sebastian, Ahmed, Mark, Abraham, Yi, Ye, Christoph, Rui, Yushan, Yinchong, Zhiliang, Michael, Mark, Thorsten, Sigurd, Aberie, Vera, Agnes, Alex, Lari, Case, Christine, Guliana, Robert, Pamela, William, ..., so many other nices colleagues from Siemens that I can not enumerate fully.*

... *The great CompStat football team with Daniel, Steffen, Christian, Quay, Heidi, Julia, Moritz, ..., all the nice colleagues at the Department of Statistics for the lovely atmosphere, the great mensa team, ...*

... *My friends, my training partners. My family and girlfriend who have been supporting me most of the time.*

Contents

Summary

A machine learning system, including when used in reinforcement learning, is usually fed with only limited data, while aimed at training a model with good predictive performance that can generalize to an underlying data distribution. Within certain hypothesis classes, model selection chooses a model based on selection criteria calculated from available data, which usually serve as estimators of generalization performance of the model.

One major challenge for model selection that has drawn increasing attention is the discrepancy between the data distribution where training data is sampled from and the data distribution at deployment. The model can over-fit in the training distribution, and fail to extrapolate in unseen deployment distributions, which can greatly harm the reliability of a machine learning system. Such a distribution shift challenge can become even more pronounced in high-dimensional data types like gene expression data, functional data and image data, especially in a decentralized learning scenario. Another challenge for model selection is efficient search in the hypothesis space. Since training a machine learning model usually takes a fair amount of resources, searching for an appropriate model with favorable configurations is by inheritance an expensive process, thus calling for efficient optimization algorithms.

To tackle the challenge of distribution shift, novel resampling methods for the evaluation of robustness of neural network was proposed, as well as a domain generalization method using multi-objective bayesian optimization in decentralized learning scenario and variational inference in a domain unsupervised manner.

To tackle the expensive model search problem, combining bayesian optimization and reinforcement learning in an interleaved manner was proposed for efficient search in a hierarchical conditional configuration space. Additionally, the effectiveness of using multi-objective bayesian optimization for model search in a decentralized learning scenarios was proposed and verified.

A model selection perspective to reinforcement learning was proposed with associated contributions in tackling the problem of exploration in high dimensional state action spaces and sparse reward. Connections between statistical inference and control was summarized.

Additionally, contributions in open source software development in related machine learning sub-topics like feature selection and functional data analysis with advanced tuning method and abundant benchmarking were also made.

Zusammenfassung

Ein Machine Learning System, auch in der Anwendung in Reinforcement Learning, wird in der Regel nur mit begrenzten Daten angereichert. Ziel ist das Lernen eines Modells mit guter Vorhersageleistung, das zugleich auf der zugrundeliegenden Datenverteilung verallgemeinern kann. Modellwahl in Machine Learning sucht ein Modell auf der Grundlage bestimmter Auswahlkriterien aus, die aus den verfügbaren Daten berechnet werden und normalerweise als Schätzer der Generalisierungsleistung des Modells dienen.

Eine große Herausforderung für Modellwahlverfahren, welche zunehmend an Aufmerksamkeit gewinnen, ist die Diskrepanz zwischen der Datenverteilung der Trainingsdaten der Verteilung beim späteren Einsatz des Modells. Ein potenzielles Problem ist die Überanpassung des Modells auf der Trainingsverteilung, die zu ungenügender Extrapolation auf ungesehenen Daten führt. Dies beinflusst die Zuverlässigkeit eines maschinellen Lernsystems stark. Eine solche Herausforderung manifestiert sich besonders bei hochdimensionalen Daten, wie Genexpressionsdaten, funktionalen Daten oder Bilddaten. Eine weitere Herausforderung für die Modellwahl ist die effiziente Suche im Hypothesenraum. Da das Trainieren solcher Verfahren in der Regel eine beträchtliche Menge an Ressourcen in Anspruch nimmt, ist die Suche nach einem geeigneten Modell mit günstigen Konfigurationen von Natur aus ein teurer Prozess und erfordert daher effiziente Optimierungsalgorithmen.

Um die Herausforderung der Verteilungsverschiebung zu bewältigen, werden eine neuartige Resampling-Methode zur Bewertung der Robustheit eines neuronalen Netzes vorgeschlagen, sowie eine Domänengeneralisierungsmethode unter Verwendung von bayesianischer multikriterieller Optimierung in einem dezentralen Lernszenario und approximate Bayesianische Inferenz in einer nicht überwachten Domäne.

Um das computational teure Problem der Modellsuche zu lösen, wird die Kombination von Bayesianischer Optimierung und Reinforcement Learning aufverschachtelte Weise für eine effiziente Suche in einem hierarchischen bedingten Konfigurationsraum vorgeschlagen. Zusätzlich wird die Effektivität des Einsatzes von multi-objektiver bayesianischer Optimierung für die Modellsuche in einem dezentralen Lernszenario vorgeschlagen und verifiziert.

Weiterhin präsentiert die vorliegende Arbeit eine neue Perspektive zur Modellauswahl in Reinforcement Learning mit zugehörigen Beiträgen zur Bewältigung von Exploration in hochdimensionalen Zustandsaktionsräumen und spärlicher Belohnung, sowie der Verbindungen zwischen statistischer Inferenz und Steuerung.

Zuletzt beinhaltet die Dissertation Beiträge zur Entwicklung von Open-Source-Software in verwandten Unterthemen des maschinellen Lernens, wie der Feature-Auswahl und Ma-

chine Learning mit funktionalen Daten mithilfe fortschrittlicher Tuning-Methoden und Bewertung auf Basis umfangreicher Benchmarks.

Chapter 1

Outline

The dissertation is organized as follows. Chapter2 introduces statistical learning theory and cross validation based model selection criteria. From the model selection aspect, conventional model selection criteria calculated based on cross validation resampling, as introduced in Chapter 2, mostly split the dataset into similar distributions as shown in Appendix A and the model selection criteria calculated thereof can hardly meet the requirements of a good generalization estimator under distribution shift. To alleviate this problem, in Sun et al. (2019b) in Appendix A, we proposed novel resampling methods by using artificially created distribution shift (mixture shift) using expectation maximization and variational inference methods, as introduced in Chapter 3. The proposed method is a potential candidate for evaluating robustness and reliability of a machine learning system under distribution shift, including bayesian convolutional neural network. In Sun et al. (2019a) in Appendix B, we proposed multi-objective model selection criteria and a novel learning framework of sending models to remote data sites and get the feedback signal to tune models using multi-objective bayesian optimization and showed its effectiveness for model selection under distribution shift. This algorithm also partially tackles the emerging challenge of data privacy and practical limitations in biomedical data transmission. Additionally, we developed novel domain generalization algorithms in Appendix C to tackle the distribution shift problem when data from several domains are available.

Training a function approximator in deep reinforcement learning also requires appropriate model selection criteria and Chapter 4 explains in detail a model selection perspective to deep reinforcement learning. In Zhao et al. (2019) in Appendix D, we proposed using an approximation to weighted entropy as surrogate to realize transitions replay buffer distribution manipulation to improve the training efficiency of the continuous action space agent on robot manipulation tasks. In Sun and Bischl (2019) in Appendix E, we summarized surrogate methods based on optimizing Evidence Lower Bound (ELBO) with respect to policies, dynamics, state space, etc.

From the model search aspect, model selection can be achieved through configuring the learning algorithm, including the composition of a pipeline which potentially consists of different stages of computations or data flows, as well as the hyper-parameter configuration for each computation stage as introduced in Chapter 5. When considering all available

configurations of several hypothesis classes together for each stage, the combined configuration search space is hierarchical and conditional. Inspired by hierarchical reinforcement learning, in Sun et al. (2019c) in Appendix F, we proposed a novel pipeline tuning methods by combing bayesian optimization and reinforcement learning which showed favorable results compared to several competitor methods. Additionally, in Appendix B, we showed the effectiveness of multi-objective model selection in distribution shift under decentralized learning scenario.

Additionally, efforts and contributions in the development of open source software with abundant benchmarking and advanced tuning methods for machine learning were also made. For example, many emerging data types like gene expression data and functional data are characterized by their high dimensionality, which calls for specific learning and model selection methods. In Chapter 6, several spline based Functional Data Analysis (FDA) methods are introduced to tackle the high dimensionality in time series data and how the smooth selection is conducted for model selection. In Pfisterer et al. (2019) in Appendix G, we compared FDA methods and non-functional machine learning methods in terms of predicative performance and the effectiveness of bayesian optimization in model selection of functional data. Besides, we designed and implemented `mlrFDA`, an extension of `mlr` for functional data classification and regression, as well as the tuning and benchmark of the incorporated algorithms, in comparison with recently published time series classification benchmark results. For vector valued high dimensional data types like gene expression data from micro array, one way to deal with high dimensionality is feature selection. In Bommert et al. (2020) in Appendix H, we introduced extension to `mlr` with several information theoretic feature filtering algorithms as explained in Chapter 7 and the benchmark of their performance on 16 high dimensional datasets with respect to other filtering based methods.

Chapter 2

Machine Learning and Model Selection

2.1 Introduction

Given a dataset $\{x_i, y_i\}_{i=1}^n$, where x_i is the covariate for observation i and y_i is the corresponding response, pragmatically, a machine learning model is usually trained by optimizing an objective function on a training split and evaluated with some measures on a hold-out split. The optimization usually comes with configurations, the so called hyper-parameters, that are not explicit in the prediction rule $\hat{y}_i = \hat{f}(x_i)$, where $\hat{f}$ is the trained model and $\hat{y}_i$ is the predicted value corresponding to x_i. However, $\hat{f}$ itself depends on these hyper-parameters. These hyper-parameters give a leverage to control or select the behavior of the trained model $\hat{f}$. The task of model selection is to estimate these hyper-parameters based on the data itself.

Section 2.2 introduces concepts from statistical learning theory and the use of cross validation for the computation of model selection criteria. Section 2.3 further explains some properties of Ordinary Cross Validation (OCV) from a regression perspective used in Functional Data Analysis in Chapter 6. Section 2.4 connects the chapter with the contributing articles of the dissertation.

2.2 Model Selection from Learning Theory

2.2.1 PAC Learning Theory and VC-dimension

Let $\mathcal{X}$ represent the domain of a machine learning algorithm and $\mathcal{Y}$ represent a corresponding concept space. The joint distribution $D_{x,y}(x, y)$ over $\mathcal{X} \times \mathcal{Y}$ can be decomposed to D_x and $D_{y|x}$. Training set S of size m[1] consists of $\{(x_1, y_1), (x_2, y_2), \cdots, (x_m, y_m)\}$, where $(x_i, y_i) \sim D_{x,y}(x, y)$. A machine learning algorithm A takes a training set S as input and outputs a hypothesis $h : \mathcal{X} \to \mathcal{Y}$. $\mathcal{H}$ represent the hypothesis class, or the space of hypothesis to be searched for. Use $Z = \mathcal{X} \times \mathcal{Y}$ to represent a general domain concept space,

[1] In this section, m instead of n is used to denote sample size to keep it consistent with other learning theory literatures.

the generalization error of a hypothesis h can be defined $L_D^Z(h) = \mathbb{E}_{z\sim D}[l(h,z)]$ where l is the instance wise loss function. Empirical Risk Minimization (ERM) searches $\arg\max_{h\in\mathcal{H}} L_S$, where L_S is the risk (loss) evaluated upon the Training Set S Vapnik (2013),Von Luxburg and Schölkopf (2011), Shalev-Shwartz and Ben-David (2014). Realizable Learning supposes $x \in \mathcal{X}$ contains all features required to decide $y \in \mathcal{Y}$, so $\exists h \in \mathcal{H}$ such that $L_S(h) = 0$. In the agnostic case, due to lack of features, the concept value y is not deterministic, instead, a probability of $P(y \mid x)$ characterizes the statistical relationship between feature vector x and concept y Shalev-Shwartz and Ben-David (2014). Bayesian Optimal Classifier Cawley and Talbot (2010); Von Luxburg and Schölkopf (2011) with $b(x) = I\left(P(y = 1 \mid x) > \frac{1}{2}\right)$ has minimal generlization error in the agnostic case, where I is the indicator function. Since $I(h(x) \neq y) = |h(x) - y|$, we have

$$
\begin{aligned}
L_D(h) &= \mathbb{E}_{\mathbb{X},\mathbb{Y}}[I(h(x) \neq y) \mid x] && (2.1)\\
&= \mathbb{E}_{\mathbb{X},\mathbb{Y}}[|h(x) - y| \mid x] && (2.2)\\
&= \mathbb{E}_{\mathbb{X}}[|h(x) - 0|P(y = 0 \mid x) + |h(x) - 1|P(y = 1 \mid x)] && (2.3)\\
&\geq \mathbb{E}_{\mathbb{X}}\left[\min\Big(P(y = 0 \mid x), 1 - P(y = 0 \mid x)\Big)\right] && (2.4)\\
&= \mathbb{E}_{\mathbb{X}} E_{Y|X}\left[I\left(P(y = 1 \mid x) >= \frac{1}{2}\right) \neq y\right] && (2.5)\\
&= \mathbb{E}_{\mathbb{X}} E_{Y|X}[b(x) \neq y] = L_D(b) && (2.6)
\end{aligned}
$$

For a learning algorithm A with hypothesis class H. Given confidence parameter δ and error tolerance ϵ, define sample complexity $m_H(\delta, \epsilon)$ with respect to hypothesis class H. H is Probably Approximately Correct (PAC) learnable Shalev-Shwartz and Ben-David (2014) when upon training set S with sample size $m >= m_H(\delta, \epsilon)$, algorithm A returns $h_S = A(S)$ satisfying $P\left(L_D^Z\left(h_S = A(S) = \arg\min_{h''\in\mathcal{H}} L_S^Z(h'')\right) > \min_{h'\in\mathcal{H}} L_D^Z(h') + \epsilon\right) < \delta$. PAC learnability ensures that, the two optimal hypothesis $\arg\min_{h''\in\mathcal{H}} L_S(h'')$ and $\arg\min_{h'\in\mathcal{H}} L_D(h')$ correspond to similar L_D with tolerance ϵ and confidence δ if sample size exceeds $m_H(\delta, \epsilon)$. Furthermore, to compare how approximate L_D and L_S is with respect to arbitrary $h \in H$ based on a choice of set S, define set S is ϵ representative Shalev-Shwartz and Ben-David (2014) when

$$\forall h \in \mathcal{H}, |L_S(h) - L_D(h)| <= \epsilon \tag{2.7}$$

To connect this ϵ representative concept as a property of a set S, with PAC Learnability: If sample size of S exceed $m_H^{UC}(\epsilon/2, \delta)$, then with δ confidence, the training set becomes $\epsilon/2$ representative, then hypothesis class H is defined to have Uniform Convergence property Vapnik (2013) and is $m_H(\epsilon, \delta)$ PAC learnable since

$$L_D(h_S) \leq L_S(h_S) + \epsilon/2 \leq L_S(h) + \epsilon/2 \leq L_D(h) + \epsilon/2 + \epsilon/2 = L_D(h) + \epsilon$$

The first inequality is due to definition of ϵ representative, the second inequality is due to the empirical optimization of the hypothesis, the third inequality is due to the definition

of ϵ representation again. Finite Hypothesis classes have uniform convergence property hence PAC learnability. For an infinite hypothesis class H, there are only a finite number of behaviors inside a set S_m of finite size m, to the maximum of 2^m. Effectively, within the scope of S_m, it looks like there are less candidates of H since some candidates behave the same inside S_m although they have a different behavior outside S_m, so the effective size of H being restricted to S_m Shalev-Shwartz and Ben-David (2014), denoted by $|H_{S_m}|$, is smaller than $|H|$. When sample size m of S_m grows, H_{S_m} grows accordingly. The Growth function Shalev-Shwartz and Ben-David (2014) $\tau_H(m) = \max_{|S_m|=m} |H_{S_m}|$ characterizes the maximum number of within set behaviors of H, or effective size of H, restricted by a set S_m of size m. If $|H_{S_m}| = 2^m$ then H shatters S_m Vapnik (2013). Note that it can be possible that H can not shatter another set of the same size m, when this happens, H has a finite Vapnik-Chervonenkis (VC) dimension. The Vapnik-Chervonenkis dimension (VC-dim) Vapnik (2013) of a function class H defined over an instance space or domain $\mathcal{X}$ is the largest size m of arbitrary subset $S_m \in \mathcal{X}$ of size m shattered by H.When set size m grows, there are more combinatorial possibilities of the sample labels in total 2^m, If $m > VCdim(H)$, then H covers only a portion of the combinatorial possibilities for any realization of S_m, and H could not shatter the set.

2.2.2 Model Selection via Risk Estimation through Validation

Given a learning task and a corresponding dataset S^m as a sample of size m from a distribution $\mathcal{D}$, model selection chooses among the available or considered hypothesis classes $h \in \cup_l H_l$ a favored model based on the data itself in terms of the risk or loss of the model $L_D(h)$ with respect to the distribution $\mathcal{D}$. Since $L_D(h)$ is intractable, an estimation is needed as a selection criteria. One solution is the Structural Risk Minimization (SRM) Vapnik (2013), where the empirical risk term based on the sampled data $L_S(h)$ is augmented with a complexity penalty which is usually utilized in the model training phase. Another solution is to estimate $L_D(h)$ through validation on an independent hold-out set V, where the loss $L_V(h)$ is used as a criteria to choose a hypothesis. In practice, to make full use of data, Cross Validation (CV) is used to aggregate performance through folds.

2.3 Cross Validation from a Regression Perspective

Since cross validation played such a big role in model selection, some properties are discussed further in the context Generalized Additive Model which is playing an essential role in Chapter 6 when Functional Data Analysis is discussed.

2.3.1 Generalized Additive Model

This dissertation considers linear smoother Hastie and Tibshirani (1990) based Generalized Additive Model (GAM) Wood (2017) in Equation 2.8, which plays an important role in Functional Data Analysis (FDA) in Chapter 6 and Appendix G. The corresponding model

selection criteria like Ordinary Cross Validation (OCV) and Generalized Cross Validation (GCV) are associated with nice theoretical insights and using cross validation for performance estimation is also pragmatic in general machine learning. A general expression for GAM can be formulated as in Equation 2.8.

$$\begin{aligned} &y_i \sim EF(\mu_i, \phi) \\ &g(\mu_i) := f_i = X_i^p \vec{\beta}_p + \sum_v F_{i,v}[f^v(x_{iv})] = X_i^p \vec{\beta}_p + \sum_v X_i^v \vec{\beta}_v = X_i \vec{\beta} \end{aligned} \tag{2.8}$$

In Equation 2.8, y_i is response of the ith observation corresponding to independent variable x_{iv} with v indexing its components, $EF(\mu_i, \phi)$ represent exponential family distribution with mean μ_i and scale parameter ϕ for observation i. X_i^p is the ith row of a parametric design matrix X^p, corresponding to global parametric coefficient vector $\vec{\beta}_p$. For simplicity of notation, we assume that x_{iv} and X_i^p disjunct. With v indexing the covariates, f^v is the corresponding smooth for x_{iv} and $F_{i,v}[f^v(x_{iv})]$ is a functional of the smooth $f^v(x_{iv})$ which can simply be $f^v(x_{iv})$ Wood (2017). X_i is the ith row of the design matrix X concatnating the parametric design matrix X^p and the smooth design matrix X^v. β is the concatenation of parametric coefficient β_p and the smooth counterpart β_v. See Chapter 6 for concrete expressions of GAM for functional data.

Estimation of Equation 2.8 comes with penalty terms for controlling the smoothness of the smoothers with a continuously valued hyper-parameter λ_v corresponding to $f^v(x_{iv})$. With Penalized Iterated Reweighted Least Squares (PIRLS) Wood (2017), fitting the corresponding working model Wood (2017) is equivalent to minimization of the objective function shown in Equation 2.9, where $z^{(r)}$ is the pseudo data Wood (2017) vector from the rth iteration, $W^{(r)}$ is the corresponding weight matrix from the r-th iteration, λ_v controlls the smoothness of the smoother through penalty matrix S_v for x_{iv}.

$$||z^{(r)} - X\beta||_{W^{(r)}} + \sum \lambda_v \beta_v^T S_v \beta_v \tag{2.9}$$

2.3.2 Ordinary Cross Validation (OCV) as Performance Estimator

Ordinary Linear Regression and Genearlized Linear Regression are special forms of GAM. Suppose a sample $\{x_i, y_i\}_{i=1}^n$ generated by $y_i = f(x_i) + \epsilon_i$ for each observation i with i.i.d $\epsilon_i \sim N(0, \sigma^2)$ and $\hat{f}_i = \hat{f}(x_i)$ is the estimated hypothesis for observation i with linear smoother Hastie and Tibshirani (1990). The Mean Squared Error (MSE) between the estimated hypothesis $\hat{f}$ and true f is

$$MSE(\hat{f}) = \frac{1}{n}\mathbb{E}[\sum_{i=1}^{n}(\hat{f}_i - f_i)^2] \tag{2.10}$$

Since f_i as well as ϵ_i are not observable, $MSE(\hat{f})$ has to be estimated. If $\hat{f}_i = Ay_i$, where A is the hat matrix or influcnce matrix of a linear smoother, the following holds:

$$\mathbb{E}[MSE(\hat{f})] = \mathbb{E}[\frac{1}{n}\sum_{i=1}^{n}(\hat{f}_i - y_i - \epsilon_i)^2] \tag{2.11}$$

$$= \mathbb{E}[MPE(\hat{f})] - \sigma^2 + 2\ tr(A)\sigma^2/n \tag{2.12}$$

, where $MPE(\hat{f}) = \mathbb{E}[\frac{1}{n}\sum_{i=1}^{n}(\hat{f}_i - y_i)^2$, $tr(A)$ is the trace of matrix A. When σ^2 is unknown, a first ReML estimate of σ^2 substituted into 2.12 will result in a model selection criteria that over-smooths the model. A better estimator is to estimate $\mathbb{E}[MSE(\hat{f})] + \sigma^2$ based on Ordinary Cross Validation (OCV) score.

Use $\hat{f}_i^{[-i]}$ to represent the i-th estimated hypothesis without taking obsevation i into consideration, the OCV score V_o can be defined as

$$V_o = \frac{1}{n}\sum_{i=1}^{n}(\hat{f}_i^{[-i]} - y_i)^2 \tag{2.13}$$

so

$$\mathbb{E}(V_o) = \frac{1}{n}\mathbb{E}(\sum_{i=1}^{n}[\hat{f}_i^{[-i]} - f_i)^2] + \sigma^2 \tag{2.14}$$

In the large sample approximation, $\hat{f}_i^{[-i]} \approx \hat{f}_i$ for any observation index i, so

$$\mathbb{E}(V_o) = \frac{1}{n}\mathbb{E}(\sum_{i=1}^{n}[\hat{f}_i^{[-i]} - f_i)^2] + \sigma^2 \tag{2.15}$$

$$\approx \frac{1}{n}\mathbb{E}[\sum_{i=1}^{n}(\hat{f}_i - f_i)^2] + \sigma^2 \tag{2.16}$$

$$= \mathbb{E}[MSE(\hat{f})] + \sigma^2 = \mathbb{E}[MPE(\hat{f})] - 2\ tr(A)\sigma^2/n \tag{2.17}$$

so V_0 can be used as an estimator for $\mathbb{E}[MSE(\hat{f})] + \sigma^2$ Wood (2017).

2.3.3 Computation of OCV and Generalized Cross Validation

Computation of OCV of Linear Smoother based Regression

If the response follows Gaussian distribution with identity link $g(\mu_i) = \mu_i$, the influence matrix (or hat matrix) A corresponding to Equation 2.9 at convergence only depends on the design matrix X with penalty $\sum \lambda_v \beta_v^T S_v \beta$. According to Wood (2017),

$$\hat{f}_i^{[-i]} = \hat{f}_i - A_{ii}y_i + A_{ii}\hat{f}_i^{[-i]} \tag{2.18}$$

so

$$V_o = \frac{1}{n}\sum_{i=1}^{n}(\hat{f}_i^{[-i]} - y_i)^2 = \frac{1}{n}\sum_{i=1}^{n}\frac{(y_i - \hat{\mu}_i)^2}{(1 - A_{ii})^2} \tag{2.19}$$

Generalized Cross Validation (GCV) of Linear Smooother

To make OCV score invariant to orthogonal transformation of the design matrix and avoid too much leverage of individual observations A_{ii}, reparameterization (Wood, 2017) using orthogonal matrix can be carried out, and the OCV expression in Equation 2.19 becomes the GCV score in Equation 2.20.

$$GCV = \frac{n||\vec{y} - A\vec{y}||^2}{(n - tr(A))^2} \tag{2.20}$$

2.4 Contributions and Prospects

This chapter introduces concepts from statistical learning theory and the commonly used validation set and cross validation as model selection criteria, which is also widely used in the community and the contributing articles Sun et al. (2019a,c); Pfisterer et al. (2019); Bommert et al. (2020); Sun et al. (2019b). Yet challenges from shifted distribution has recently drawn increasing attention in the research community Nalisnick et al. (2018); Ren et al. (2019); Sastry and Oore (2020). At deployment time, the data fed into a machine learning system can come from a different distribution from the training one. As showed empirically in the contributing articles Sun et al. (2019a), Sun et al. (2019b) and Gossmann et al. (2019), Ordinary Cross Validation (OCV) on the same dataset is not able to well characterize the train-test behavior under distribution shift.

In the contributing article Sun et al. (2019b), inspired by mixture shift Quionero-Candela et al. (2009), variational inference is used to create mixture component, and a leave one component out cross validation is developed as a criteria to access the robustness of neural networks against such artificially created distribution shift. The proposed mixture shift based performance estimation methods we coined `vgmm-vae` in the contributing article Sun et al. (2019b) acts as a within dataset near-worst-case estimation of out of distribution performance, and can potentially be used as benchmark methods for evaluating model robustness against distribution shift and out-of-distribution detection.

For future research, it is interesting to see a concrete use case of `vgmm-vae` resampling Sun et al. (2019b) as a benchmark criteria to compare different domain generalization Li et al. (2017) and domain adaptation Quiñonero-Candela et al. (2009); Hoffman et al. (2018) methods under mixture shift. Furthermore, it will also be of great interest to test if the new resampling method can be used to derive new domain adaptation or domain generalization methods. For example, if pre-training the resampled dataset with `vgmm-vae` using domain generalization techniques can improve out-of-distribution prediction performance.

Additionally, in the contributing article Sun et al. (2019a), although we tried to solve decentralized learning scenario under distribution shift, our multi-objective bayesian optimization solution can also be seen as optimizing a new multi-objective model selection criteria of considering predicative performance on all data sites. In Sun and Buettner

(2021), we proposed unsupervised domain generalization where model selection is carried out using extended Evidence Lower Bound (ELBO).

In Chapter 4, we offer a model selection perspective to reinforcement learning.

Chapter 3

Evidence based Model Selection

3.1 Introduction

Bayesian evidence and its lower bound optimization methods serve as alternative model selection criteria to cross validation based performance estimation methods Cawley and Talbot (2010).

The field of variational inference is still ongoing research Blei et al. (2017) and has been recently utilized in the deep learning community Kingma and Welling (2013), the basic idea of Variational Inference is introduced in this chapter, with EM algorithm being explained as a special case of Evidence Lower Bound optimization method. Then contributions of the dissertation in this area is summarized in Section 3.4.

3.2 Expectation Maximization

Given observation x, adding unobserved latent variable z enables more complex distributions. While the latent variable is not observed, it is possible to use the conditional distribution $p(x|z,\theta)$ and the prior $p(z|\theta)$ as a leverage, in an iterative method, each iteration leads to better assignment to the z variable and θ, with the likelihood increases.

$$p(x|\theta) = \sum_z p(x,z,\theta) = \sum_z p(z|\theta)p(x|z,\theta) \tag{3.1}$$

Our goal is maximum likelihood of observable probability with $\theta^* = \underset{\theta}{argmax} \log p(x|\theta)$, which can in principle be done by gradient decent or newton method. Since it can be difficult to get analytical form of $\log(p(x|\theta)) = \ell(\theta)$ and $\frac{\partial \ell(\theta)}{\partial \theta}$. Now we seek an alternative method that does not accent in the gradient direction or newton direction directly but goes in a less steeper way but ensures that each time it moves to a better place in an iterative way $\theta^{new} \leftarrow \theta^{old}$ which leads to better likelihood with $\ell(\theta^{old}) < \ell(\theta^{new})$.

The complete data likelihood $\ln p(x,z,\theta)$ can be decomposed as the product of marginal likelihood (the objective to be optimized) and posterior distribution of latent variable. Take

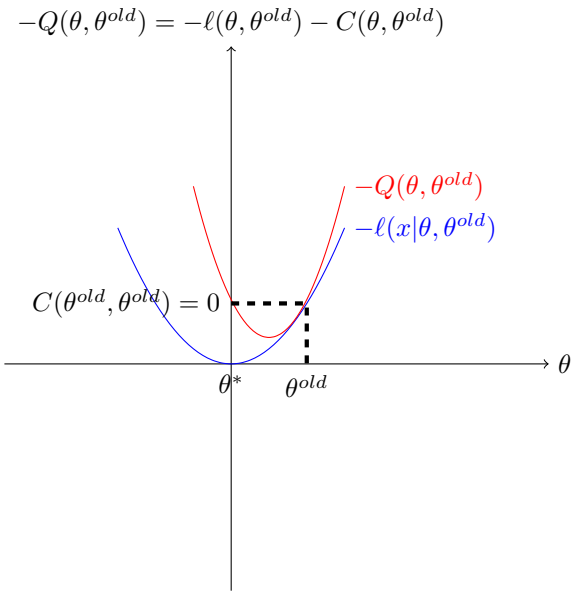


Figure 3.1: illustration of EM algorithm

the log likelihood of the complete data likelihood,

$$\ln p(x,z,\theta) = \ln\{p(x|\theta)\} + \ln\{p(z|x,\theta)\} \tag{3.2}$$

Since z as a latent random variable is not observed, the solution is to integrate out z in an expectation calculation with respect to the current iteration posterior with $p(z|x,\theta^{old})$. Starting from θ^{old}, with tractable posterior $p(z|x;\theta^{old})$, multiply both sides of Equation 3.2 by $p(z|x,\theta^{old})$ and integrate with respect to z we have

$$\int_z \ln\{p(x,z,\theta)\}p(z|x,\theta^{old})dz = \int_z \ln\{p(x|\theta)\}p(z|x,\theta^{old})dz + \int_z \ln\{p(z|x,\theta)\}p(z|x,\theta^{old})dz \tag{3.3}$$

or

$$Q(\theta,\theta^{old}) = \ell(\theta,\theta^{old}) + C(\theta,\theta^{old}) = \ell(\theta) + C(\theta,\theta^{old}) \tag{3.4}$$

where $Q(\theta,\theta^{old}) = \int_z \ln p(x,z,\theta)p(z|x,\theta^{old})dz$ is pivoted on θ^{old} in the parameter space as a surrogate to $\ell(\theta|x)$, $C(\theta,\theta^{old}) = \int_z \ln\{p(z|x,\theta)\}p(z|x,\theta^{old})dz < 0$ are two functional component of $\ell(\theta|x)$. Note our objective is to optimize $p(x|\theta)$, multiply the objective $p(x|\theta)$ by a constant $\int_z p(z|x,\theta^{old})dz = 1$ will not affect our goal $\mathbb{E}_{p(z|x,\theta^{old})}p(x|\theta) = \int_z p(x|\theta)p(z|x,\theta^{old})dz = p(x|\theta)\int_z p(z|x,\theta^{old})dz = p(x|\theta)$. So $\ell(\theta,\theta^{old}) = p(x|\theta)$ can be denoted as $\ell(\theta)$ as well.

If we have $Q(\theta^{new},\theta^{old}) \geq Q(\theta',\theta^{old})$ for any θ', equivalently we have $\ell(\theta^{new},\theta^{old}) + C(\theta^{new},\theta^{old}) \geq \ell(\theta',\theta^{old}) + C(\theta',\theta^{old})$, to ensure the likelihood increases, it has to be

$C(\theta^{new}, \theta^{old}) < C(\theta, \theta^{old})$, which can be shown by Jensen inequality.

$$C(\theta^{new}, \theta^{old}) - C(\theta^{old}, \theta^{old}) = \int_z \ln\{\frac{p(z|x, \theta^{new})}{p(z|x, \theta^{old}}\} p(z|x, \theta^{old}) dz \tag{3.5}$$

$$= -KL(p(z|x; \theta^{old})||p(z|x; \theta^{new})) \tag{3.6}$$

$$= \mathbb{E}_{\theta^{old}}[\ln\{\frac{p(z|x, \theta^{new})}{p(z|x, \theta^{old}}\}] < \ln \mathbb{E}_{\theta^{old}}(\frac{p(z|x, \theta^{new})}{p(z|x, \theta^{old}}) \tag{3.7}$$

$$= ln \int_z \{\frac{p(z|x, \theta^{new})}{p(z|x, \theta^{old}}\} p(z|x, \theta^{old}) dz = ln1 = 0 \tag{3.8}$$

EM algorithm is illustrated in Figure 3.1, where the E step is to compute the $Q(\theta^{'}, \theta^{old})$ surrogate, and the M step is to maximize the surrogate with respect to $\theta^{'}$. E step and M step are conducted iteratively, where θ^{old} serve as a pivot as shown in Figure 3.1.

3.3 Variational Inference

The core of EM algorithm is the calculation of

$$Q(\theta, \theta^{old}) = \int_z \ln p(x, z, \theta) p(z|x, \theta^{old}) dz = \int_z \ln[p(z, \theta) p(x|z; \theta)] \frac{p(z, \theta^{old}) p(x|z; \theta^{old})}{\int_z p(x, z'; \theta) dz'} dz \tag{3.9}$$

However, the posterior distribution $p(z|x, \theta^{old}) = \frac{p(z, \theta^{old}) p(x|z; \theta^{old})}{\int_{z'} p(x, z'; \theta) dz'}$ can be intractable due to the intractable partition function $\frac{1}{\mathcal{Z}} = \frac{1}{\int_z p(x, z'; \theta) dz'} = \frac{1}{p(x)}$, the solution is to replace the intractable posterior $p(z|x, \theta)$ with a proposal posterior $q(z|\phi)$ governed by variational parameter ϕ. When the partition function $\frac{1}{p(x)}$ is intractable, maximizing $\ln p(x)$ directly is not possible. The solution is to find a lower bound for $\ln p(x)$ as a surrogate to be optimized. But a tight bound like $Q(\theta, \theta^{old}) = \int_z \ln p(x, z, \theta) p(z|x, \theta^{old}) dz$ is not possible since the posterior $p(z|x, \theta^{old})$ is intractable due to intractable partition function.

To measure the closeness of the proposal posterior distribution $q(z|x, \phi)$ to the true posterior $p(z|x, \theta)$, KL divergence can be utilized. Use plus one term and minus one term trick to the $p(x, z, \theta)$ decomposition with the term being $q(z|x, \phi)$ to form KL divergence term, one gets

$$\ln p(x, z, \theta) = \ln\{p(x|\theta)\} + \ln\{q(z|x, \phi)\} - \ln\{q(z|x, \phi)\} + \ln\{p(z|x, \theta)\} \tag{3.10}$$

Express the marginal distribution of observables in terms of others

$$\ln\{p(x|\theta)\} = \ln\{p(x, z, \theta)\} - \ln\{q(z|x, \phi)\} + \ln\{q(z|x, \theta)\} - \ln\{p(z|x, \theta)\} \tag{3.11}$$

$$= \ln \frac{p(x, z, \theta)}{q(z|x, \phi)} + (-1) \ln \frac{p(z|x, \theta)}{q(z|x, \phi)} \tag{3.12}$$

Take expectation of both sides with respect to $q(z|x,\theta)$, with $\int_z q(z|x,\theta)dz = 1$.

$$\ln\{p(x|\theta)\} \cdot 1 = \int_z q(z|x,\theta) \ln \frac{p(x,z,\theta)}{q(z|x,\theta)} dz + (-1) \int_z q(z|x,\theta) \ln \frac{p(z|x,\theta)}{q(z|x,\theta)} dz \tag{3.13}$$

$$= -KL\left(q(z|x,\theta)||p(x,z,\theta)\right) + KL\left(q(z|x,\theta)||p(z|x,\theta)\right) \tag{3.14}$$

$$= ELBO(x,\phi,\theta) + KL\left(q(z|x,\theta)||p(z|x,\theta)\right) \tag{3.15}$$

Thus, $ELBO(x,\phi,\theta)$ as a lower bound for the evidence $\ln p(x|\theta)$ can be used as a surrogate to be optimized and has been recently extended to the stochastic optimization case Hoffman et al. (2013).

3.4 Contributions and Prospects

As explained in the contributing article Sun et al. (2019b), variational inference can be applied to weight distribution in bayesian neural networks. As explained in the contributing article Sun and Bischl (2019) variational inference can assist efficient exploration in high dimensional state and action spaces with sparse rewards, which can be seen as a special model selection method, as explained in Chapter 4. In the contributing article Sun et al. (2019a), a special case of Expectation Maximization (EM) algorithm Bishop (2006) was used to infer Mixture of Gaussian (MOG) distributions of a dataset. In the contributing article Sun et al. (2019b), variational inference was used to infer distributions of a dataset. Both methods were used to artificially create Mixture Shift Quionero-Candela et al. (2009) to further construct novel resampling methods in Sun et al. (2019a,b). In the contributing article Zhao et al. (2019), variational inference of MOG distributions was used for density estimation. In the contributing article Sun and Buettner (2021), variational inference was used to build novel algorithm for domain generalization.

Chapter 4

A Model Selection Perspective to Reinforcement Learning

4.1 Introduction: Challenges of High-Dimensionality and Sparse Reward

Many reinforcement learning scenarios are characterized with high dimensional state action spaces Zhao et al. (2019); Sun and Bischl (2019) including high dimensional state spaces like images Mnih et al. (2015), other high dimensional sensor data like data from tactile sensor Van Hoof et al. (2016) and continuous action spaces Silver et al. (2014). Additionally, in occasions like robot manipulation tasks it is natural and beneficial to design reward only based on if the agent accomplished the task or not Plappert et al. (2018), resulting in sparse reward Riedmiller et al. (2018). In deep reinforcement learning , deep neural networks are used as function approximators Sutton and Barto (2018). However, the high-dimensionality in both state and action spaces concerted with sparse rewards poses great challenges in robust behavior and efficient exploration in the environment, which greatly obviate the successful application of reinforcement learning algorithms Houthooft et al. (2016). Additionally, it is favorable for the learned policy to generalize across different goals Plappert et al. (2018).

In this chapter, a model selection perspective to reinforcement learning is given from different scopes to tackle the above mentioned challenges. A top level scope is evidence based model selection method introduced in Chapter 3 by optimizing Evidence Lower Bound (ELBO) for reinforcement learning, via connecting inference and control as explained in Section 4.2 and the contributing article Sun and Bischl (2019). The idea is to unroll the state action reward transitions into the whole trajectory, a probabilistic graphical model along the trajectory with latent variables is imposed and variational inference is used to infer the latent variable.

A lower level scope is choosing loss function for the value function estimation discussed in Section 4.3 and shifting training data distribution elaborated in Section 4.4.

4.2 Model Selection by Probabilistic Inference for Control

Deep reinforcement learning solves the optimal control Boudarel et al. (1971) objective in Equation 4.1.

$$\theta^* = \arg\max_{\theta} \mathbb{E}_{\mathcal{T}(\cdot|\mathcal{E},\pi_\theta)} \sum_{t=0}^{\infty} R(S_t, A_t^c \sim \pi_\theta(\cdot|S_t)) \tag{4.1}$$

In Equation 4.1, $\mathcal{T}(\cdot|\mathcal{E},\pi_\theta)$ is the trajectory distribution under the environment $\mathcal{E}$ and policy π_θ parameterized by θ. The policy distribution $\pi_\theta(\cdot|S_t))$ is responsible for taking the sequential decision conditioned on current state S_t. Each trajectory is supposed to start at $t = 0$ and R_t is the per step reward conditioned on the current state S_t and action A_t^c. In case of presence of discounted factor $\gamma = e^{1/T_c}$ characterizing a characteristic horizon T_c to ensure finite return in infinite horizon problem, $R_t = \gamma^t R_t^{'}$ can be defined with $R^{'}$ being the undiscounted reward. The optimal control problem in Equation 4.1 can be unrolled into a Probabilistic Graphical Model (PGM) and transformed into an inference problem of infering optimal action A_t^c under current state S_t Levine (2018); Sun and Bischl (2019). Accordingly, the optimal control objective in Equation 4.1 can be augmented by the entropy of policy $\mathcal{H}(\pi(\cdot|S_t))$, as in Equation 4.2 Levine (2018); Sun and Bischl (2019).

$$\theta^* = \arg\max_{\theta} \mathbb{E}_{\mathcal{T}(\cdot|\mathcal{E},\pi_\theta)} \sum_{t=0}^{\infty} R(S_t, A_t^c \sim \pi_\theta(\cdot|S_t)) + \mathcal{H}(\pi(\cdot|S_t)) \tag{4.2}$$

In the contributing article Sun and Bischl (2019), some further derivations about the connection between optimal control and probabilistic inference are given. Additionally, we introduced extra PGMs modeling other aspects of the reinforcement learning problem to deal with the high-dimensional state-action spaces and sparse reward problem discussed in Section 4.1. For example, the use of undirected graph Sallans and Hinton (2004) for policy inference. Using variational inference for inferring the dynamics of the environment can augment exploration encouraging pseudo rewards to alleviate the sparse reward challenge Houthooft et al. (2016). Dimensionality reduction using variational inference is also explained in the contributing article Sun and Bischl (2019), see detail in Appendix E.

4.3 Model Selection for Value Function Estimator

Dynamic programming Boudarel et al. (1971) can be used to optimize the objectives in Equation 4.1 and 4.2 which requires the value function. In deep reinforcement learning , an estimator to state value function $Q^\pi(s,a)$ with respect to policy π is required for both value function based methods Mnih et al. (2015); Van Hasselt et al. (2016); Haarnoja et al. (2017) and policy gradient based methods Degris et al. (2012); Schulman et al. (2015); Lillicrap et al. (2015); Haarnoja et al. (2018); Zhao et al. (2019). The estimator $\hat{Q}_w^\pi(s,a)$

is usually represented as a deep neural networks parameterized by w. The estimation of w is usually converted to a regression problem with loss function in Equation 4.3.

$$w^* = \arg\min_w \mathbb{E}_{s,a,s' \sim \mathcal{D}} \ \ell(Q^\pi(s,a), \hat{Q}_w(s,a)) \tag{4.3}$$

, where $Q^\pi(S_t = s, A_t^c = a) = \mathbb{E}_{S,A,S',R \sim \mathcal{E},\pi}[R(s,a) + \sum_{j=1}^{\infty} R(S_{t+j}, A_{t+j})]$ is approximated by $\hat{Q}_w(s,a)$ according to loss function ℓ under the transition sample distribution $\mathcal{D}$ and the model selection problem lies in selecting the transition distribution $\mathcal{D}$ and the loss function ℓ.

For example, the on-policy method chooses $\mathcal{D}$ to be rolled out transition samples from the policy π to be optimized, while the off-policy method chooses $\mathcal{D}$ to be generated from a behavior policy β and $\mathcal{D}$ is materialized by sampling from a replay memory to reuse the samples to increase sample efficiency Lin (1991).

Since the true state action value function $Q^\pi(s,a)$ is only partially observable through reward R, the loss function ℓ is constructed by taking a surrogate using the Bellmann operator $\ell\left(Q^\pi(s_t,a_t), \hat{Q}_w(s_t,a_t)\right) \approx (\Gamma^{s_{t+1}}\hat{Q}_w(s_t,a_t) - \hat{Q}_w(s_t,a_t))^2$. The Bellmann operator $\Gamma^{s_{t+1}}$ is a contraction satisfying $||\Gamma^{s_{t+1}}Q_1(\cdot,\cdot) - \Gamma^{s_{t+1}}Q_2(\cdot,\cdot)|| \leq \gamma||Q_1(\cdot,\cdot) - Q_2(\cdot,\cdot)||$, so a fixed point solution $\Gamma^{s_{t+1}}\hat{Q}_w(s_t,a_t)) = \hat{Q}_w(s_t,a_t)$ can be approximated. So the choice of loss function ℓ can be reduced to the choice of Bellmann operator Γ as explained below.

For Deep Q learning Mnih et al. (2015), $\Gamma^{s_{t+1}}\hat{Q}_w(s_t,a_t) = r_t + \max_a \gamma\hat{Q}_{\bar{w}}(s_{t+1},a)$ is the Bellman operator with $\bar{w}$ being the weight for the target network. For Double DQN Van Hasselt et al. (2016), $\Gamma^{s_{t+1}}\hat{Q}_w(s_t,a_t) = r_t + \gamma\hat{Q}_{\bar{w}}\left(s_{t+1}, \arg\max_a \hat{Q}_w(s_{t+1},a)\right)$, where value maximization is done by the update network w and evaluation is done by the target network $\bar{w}$. For Clipped Double Q Learning Fujimoto et al. (2018), as well as in Batch Constrained Q Learning Fujimoto et al. (2019). $\Gamma^{s_{t+1}}\hat{Q}_w(s_t,a_t) = r_t + \gamma\max_a[\lambda \min_{w_1,w_2} \hat{Q}_{w\in\{w_1,w_2\}}(s_{t+1},a) + (1-\lambda)\max_{w_1,w_2}\hat{Q}_{w\in\{w_1,w_2\}}(s_{t+1},a)]$ where w_1 and w_2 are two networks updated interchangeably.

In terms of continuous action space, we use Γ^π to represent the Bellmann operator. Soft Q learning Haarnoja et al. (2017) is an implicit actor-critic algorithm Haarnoja et al. (2017, 2018); Schulman et al. (2017). The Bellmann operator is $\Gamma^\pi\hat{Q}_w(s_t,a_t) = r_t + \gamma\mathbb{E}_{S_{t+1}\sim\mathcal{D}}\alpha\log\int_a \exp\frac{1}{\alpha}\hat{Q}_{\bar{w}}(S_{t+1},a)da$. $\alpha\log\int_a \exp\frac{1}{\alpha}\hat{Q}_{\bar{w}}(S_{t+1},a)da$ can be seen as taking log of a continuous softmax denominator, which is dominated by the maximum of action a, with α tuning the approximation level, also controlling the degree of the maximum entropy objective in Equation 4.2. For Deep Deterministic Policy Gradient (DDPG) Lillicrap et al. (2015), $\Gamma^\pi\hat{Q}_w(s_t,a_t) = r_t + \gamma\hat{Q}_{\bar{w}}(S_{t+1},\pi(S_{t+1}))$ with $S_{t+1} \sim \mathcal{D}$. For soft actor critic Haarnoja et al. (2018), $\Gamma^\pi\hat{Q}_w(s_t,a_t) = r_t + \gamma\mathbb{E}_{S_{t+1}\sim\mathcal{D},A_{t+1}^c\sim\pi}[\hat{Q}_{\bar{w}}(S_{t+1},A_{t+1}^c) - \log\pi(A_{t+1}^c|S_{t+1})]$.

4.4 Model Selection with respect to Shifted Distribution

The distribution $\mathcal{D}$ in Equation 4.1 and 4.2 depends on the evolved policy π and the environment $\mathcal{E}$. For off-policy reinforcement learning, when using uniform sampling, $\mathcal{D}$

corresponds to a mixture distribution of the evolved policies.

The training makes the neural network agree with these samples in terms of Equation 4.3, thus imposing the risk of over-fitting Q to the current state action transitions distribution $\mathcal{D}$. In low dimensional reinforcement learning with dense reward carried online, overfitting of the value function Q is not a pernicious problem since if the value function approximator has over optimistic extrapolation, the agent will be guided to explore these over optimistic regions and the dense reward will guide the agent learn new behavior to adapt to the new experiences Kumar et al. (2019); Fujimoto et al. (2019). Besides, strategies like the ϵ exploration Sutton and Barto (2018), Boltzmann exploration, Gibbs exploration and other noise injection methods Lillicrap et al. (2015) can also help the agent balance the exploration-exploitation trade off to gather diverse experiences that can better reflect a global picture of the transitions distribution under the current policy. However, it will be difficult for these exploration mechanisms to function well enough in case of high-dimensional state-action spaces, especially when continuous action and sparse reward is also present Zhao et al. (2019).

Note for sparse reward and continuous actions, Q parameterized by neural network in the non-reward region is just a smoothed flat surface and even with rewarded transitions, the reward may fail to be propagated Matheron et al. (2019). Taking DDPG Lillicrap et al. (2015) as an example, it is an off-policy algorithm for continuous action spaces. The too early overfitting of the critic Q imposes the risk that the agent can not optimize its policy so the agent will always be constrained in a sub-optimal region by a deadlock of saturation of Q and π Matheron et al. (2019), in consequence the agent has difficulty to explore to a wider region of the state-action space and fails to accomplish the task.

To alleviate the problem, a conjecture can be made in the language of distribution shift. Suppose an oracle can gather transition samples in the whole state-action space, forming a wider behavior policy β^{oracle} for the current policy π to be optimized, corresponding to a global transition distribution $\mathcal{D}^{oracle}$ with bigger entropy. $\mathcal{D}^{oracle}$ might slow down the over-fitting of the function approximator $Q^{\beta^{oracle}}$ thus obviates the deadlock proposed in Matheron et al. (2019).

The contributing article Zhao et al. (2019) can be seen to simulate this conjecture. To select a better model for multi-goal reinforcement learning under the sparse reward, a weighted entropy objective is proposed to encourage both exploration and task accomplishment. A lower bound approximation as an optimization surrogate given the state action reward transitions in the replay buffer corresponds to using density prioritized sampling to artificially create distribution shift for the replay buffer of off-policy DDPG algorithm, leading to an improved entropy of the $\mathcal{D}$, which is similar to the effect of an oracle. This sampling leads to improved performance of DDPG. For future work, more theoretical and empirical evidences are needed to support this conjecture.

Chapter 5

Bayesian Optimization and Reinforcement Learning for Model Selection

5.1 Introduction

This chapter takes a algorithm configuration Bischl et al. (2016) point of view for model selection. A machine learning algorithm is taken as it is with available configurations as hyper-parameters which affect the behavior of the model. The model selection problem then becomes an algorithm configuration problem to search for configurations or hyper-parameters to optimize performance.

Models are often optimized not alone, but as part of a machine learning pipeline. Steps in a pipeline are often configurable resulting in a hierarchical configuration space with options depending on the chosen pipeline step. In this dissertation, bayesian optimization is used to efficiently search the hyper-parameter space, which is extended to multi-objective bayesian optimization when there are multiple criteria to assess the performances. Combined with reinforcement learning , the pipeline with hierarchical configurations can also be efficiently searched, see detail in the contributing article Sun et al. (2019c) in Appendix F. Section 5.2 explains bayesian optimization with Gaussian Process, which offers both mean and variance prediction. Multi-objective optimization is introduced in Section 5.3 and optimization on hierarchical parameter spaces is introduced in Section 5.4.

5.2 Bayesian Optimization with Gaussian Process

Evaluation of a configuration of a hyper-parameter set requires training a machine learning model which is usually expensive. To save cost, based on some initial design, subsequent proposals of hyper-parameters can be evaluated sequentially based on a sequential decision mechanism Mockus (2012). Constrained by computational resources, it is favorable to build a predictive model, the so called response surface Jones et al. (1998), on the

perspective performance in unknown regions. For simplicity, hyper-parameters are also referred to as covariates to the response surface in the following texts, while the corresponding performance like classification error are referred to as response. Minimization of objective is taken as the default optimization goal. The search of favorable hyper-parameters purely based on the response surface can lead to local minima Jones et al. (1998), and there is a need to balance exploiting the response prediction and exploring uncertainties in unknown regions. Thus, to build the response surface, both the mean prediction and uncertainty prediction are needed and a stochastic process approach is favored, by regressing on the function behavior conditioned on the available observations, while the conditioned distribution of functions offers uncertainty estimation. From a regression perspective, the error terms corresponding to a response surface should have a correlation structure dependent on the distance of the covariates to the response surface Jones et al. (1998). This section introduces Gaussian Process regression to build response surfaces with mean and variance prediction and construct infill criteria like Expected Improvement accordingly.

5.2.1 Gaussian Process

A direct view of Gaussian Process can be seen as a collection of response variables $f(x)$ corresponding to covariate x (hyper-parameters for example) which follow a joint Gaussian distribution denoted as

$$f(x) \sim \mathcal{GP}\left(m(x), k(x, \cdot)\right) \tag{5.1}$$

, where the mean response $m(x)$ and covariance of response $k(x, x')$ are

$$m(x) = \mathbb{E}[f(x)] \tag{5.2}$$
$$k(x, x') = \mathbb{E}[(f(x) - m(x))\,(f(x') - m(x'))] \tag{5.3}$$

Gaussian Process as a non-parametric model can be viewed as Bayesian linear regression and a detailed derivation is given in Appendix I.1, where the prior probability of "parameters/weights" over basis functions is implicitly defined by kernels. A simple case of kernel matrix would be a quadratic form like the inner product with respect to the covariance matrix of the weight distribution, see Equation (I.23). From the Bayesian linear regression perspective in Appendix I.1, Let $f(x) = w^T\phi(x)$, and $w \sim \mathcal{N}(0, \Sigma_p)$, the corresponding mean and covariance function of the response $f(x)$ is

$$\mathbb{E}[f(x)] = \mathbb{E}[w^T\phi(x)] = \mathbb{E}[\phi(x)^T w] = 0 \tag{5.4}$$
$$\mathbb{E}\ \left[f(x) - \mathbb{E}[f(x)]\right]\left[f(x^{'}) - \mathbb{E}[f(x^{'})]\right] = \mathbb{E}[\phi(x)^T w w^T \phi(x)] \tag{5.5}$$
$$= \phi(x)^T E[ww^T]\phi(x) = \phi(x)^T \Sigma_p \phi(x) = k(x, x^{'}) \tag{5.6}$$

Gaussian process can also be seen as penalized regression where the penalty is the Reproducing Kernel Hibert Space (RKHS) norm of the fitting function Rasmussen (2003).

For example, when taking second derivative as the RKHS norm Wood (2017), the minimizer to

$$\sum_{i=1}^{n}\{y_i - f(x_i)\}^2 + \lambda \int f''(x)^2 dx \tag{5.7}$$

where n is the number of observations, can be written as

$$\hat{f}(x) = \beta_0 + \beta_1 x + \sum_{i=1}^{n} \delta_i k(x_i, x) \tag{5.8}$$

, where $\delta_{\mathbf{i}} = (K + \sigma_n^2 I)^{-1}\mathbf{y_i}$ with σ_n^2 being the noise estimator and k is the reproducing kernel Wood (2017), K is the matrix of $k(x_i, x_j)$ for observations x_i, x_j. When ignoring the linear regression part, for a new data point x_*, the posterior uncertainty of the response $\hat{f}(x_*)$ is

$$\mathcal{V}[\hat{f}(x_*)] = k(x_*, x_*) - [k(x_1, x_*), \ldots, k(x_n, x_*)](K + \sigma_n^2 I)^{-1}[k(x_1, x_*), \ldots, k(x_n, x_*)]^T \tag{5.9}$$

5.2.2 Expected Improvement and Efficient Global Optimization

Having built the response surface, proposal on new point can be made based on an infill criteria Bischl et al. (2014) to balance local exploitation and global exploration. One commonly used infill criteria is the Expected Improvement in Equation 5.10, where $f_{\min}$ is the current minimum observed response value, Y is a random variable characterizing the posterior response distribution, where $Y \sim \mathcal{N}(\mu(x), s^2(x))$, with mean response $\mu(x)$ and posterior variance of the response $s^2(x)$.

$$EI(x) = \mathbb{E}_{Y \sim \mathcal{N}(\mu(x), s^2(x)])}[\max(f_{\min} - Y, 0)] \tag{5.10}$$

With Gaussian Process response surface, the Expected Improvement can be written as Equation 5.11 Jones et al. (1998)

$$EI(x) = (f_{\min} - \mu(x))\Phi(\frac{f_{\min} - \mu(x)}{s(x)}) + s(x)\phi(\frac{f_{\min} - \mu(x)}{s(x)}) \tag{5.11}$$

, where Φ is the cumulative distribution function and ϕ is the probability density function of the standard Gaussian distribution. The mean prediction $u(x)$ can correspond to Equation 5.8 without the linear regression term and the variance prediction $s^2(x)$ can correspond to Equation 5.9. The Efficient Global Optimization (EGO) iterates between maximizing the Expected Improvement using branch and bound algorithm Jones et al. (1998), sampling new points and re-estimate the response surface until convergence.

5.3 Multi-Objective Optimization

5.3.1 Pareto Dominance

Consider the multi-objective optimization problem with map $\mathcal{T} : \mathcal{X} \to \{\mathcal{Y}^l\}_{l=1}^{L}$, where $\mathcal{X}$ is the space of configuration, corresponding to different objectives $\mathcal{Y}^l$ indexed by l.

A point $A \in \mathcal{X}$ is said to dominate point $B \in \mathcal{X}$ if for each objective $Y^l(A)$ it is not worse than $Y^l(B)$ and for at least one objective, it is better than B. If in a set, none of the points can dominate any other point, this set is called a non dominating set. If a point $X \in \mathcal{X}$ can not be dominated by any other point, it is called Pareto optimal. The set of points represented in $\mathcal{X}$ configuration space that can not be dominated by any other points are called Pareto set. Pareto front is the $\mathcal{Y}$ space of Pareto Set. The goal of multi-objective optimization is to search in the space of $\mathcal{X}$ and to find a Pareto Set. To compare between different Pareto fronts, a popular indicator is to compute the volume of the dominated part of the objective space with respect to a chosen reference point, which is the dominated hyper-volume.

To solve multi-objective optimization problem, Deb et al. (2002) proposed a fast elitist non-dominated sorting Genetic Algorithm for Multi-Objective Optimization called $NSGA2$. The basic idea is a rank could be pertained by non-dominated sorting, so elite points could be obtained, new generation could be spawned using Evolutionary Algorithm. The ranking is first computed based on non-dominance. In the whole population, the non-dominating set is computed and assigned rank 1. From the remaining points, it is again possible to get a non-dominating set and so on. To assign fine ranking to each point, crowd distance sorting is used which assigns more score to points that is more diverse from the rest of the points. The diverse score is calculated based on the summed up distance to the nearest neighbors.

5.3.2 Model Based Multi-Objective Optimization

This dissertation uses bayesian optimization for multi-objective problems. Multi-objective Bayesian Optimization is model based multi-objective optimization Horn et al. (2015) with the following categories in accordance with the taxonomy in Horn et al. (2015).

Scalarization based methods

Scalarization based methods transform multi-objective optimization problem into single objective optimization through the augmented weighted Tchebycheff norm Dächert et al. (2010); Knowles (2006); Horn et al. (2015) $||\vec{f} - \vec{r}||_{\rho}^{\vec{w}} = \max_{k=1,\cdots,K} w_k |f_k(x) - r_k| + \rho \sum_{k=1}^{K} w_k |f_k(x) - r_k|$, where $\vec{r}$ consisting of $\{r_k\}_{k=1}^{K}$ is the reference vector with respect to the K objectives, $\vec{f}$ consisting of $\{f_k\}_{k=1}^{K}$ is the objective vector, $\vec{w}$ consisting of $\{w_k\}$ is the weight vector, ρ balances the first term which is the most significant weighted objective

and the second term which is the average weighted objective. Pareto Efficient Global Optimization (ParEGO) Knowles (2006) samples the weight vector uniformly in each iteration.

Pareto based method and Direct Indicator based method

MultiEGO Jeong and Obayashi (2005) compute the Expected Improvment for each objective function and optimize multiple Expected Improvements with MultiObjective Genetic Algorithm (MOGA). Direct indicator based method transforms the multi-objective optimization problem to single objective using S-metric instead of a linear combination of different objectives. S-metric is defined to be the hyper-volume of a Pareto set (instead of a simple point) with regard to a reference point. For example, in Ponweiser et al. (2008), separate models are fitted with respect to each objective to surrogate the expensive evaluation and conditioned on the current Pareto front approximation, a new point is judged by its contribution to the change of hyper-volume based on the separate surrogate evaluations of each objective as infill criteria. Additionally, penalty terms are added to the infill criteria if a new point falls into dominated area.

5.4 Challgenges of Hierarchical Conditional Configuration Space

When combining several preprocessors and learners into a joint space, a union of hypothesis classes are formed. Yet the corresponding configuration space is not flat anymore, but conditional and hierarchical, which brings challenges to efficient model search. Table 5.1 is an example of a sample of such a combined configuration space of a random forest with hyper-parameter `n_tree` and a support vector machine with hyper-parameter C and `kernel`. Note when the kernel is `poly` representing polynomial kernel, there is an additional hyper-parameter n specifying the degree of the polynomial. When the kernel is `rbf`, there is an additional hyper-parameter `gamma`. Such a hierarchical conditional configuration space poses challenges for the bayesian optimization based hyper-parameter search algorithm.

5.5 Contributions and Prospects

Regarding the challenges of Section 5.4, inspired by hierarchical reinforcement learning Dietterich (2000); Kulkarni et al. (2016); Co-Reyes et al. (2018), in the contributing article Sun et al. (2019c), we use bayesian optimization to handle hyper-parameter tuning sub-tasks of a particular pipeline and let reinforcement learning act as a meta-controller to choose between different sub-tasks or pipelines, which we coined `ReinBo` and generated favorable results compared to competitor algorithms, see details in Appendix F. To our best knowledge, our ReinBo method Sun et al. (2019c) was the first to combine bayesian optimization and reinforcement learning for efficient model search in a hierarchical conditional configuration space and our method showed favorable performance.

```
##     model      C   n_tree  kernel  n    gamma
##  1:    RF     NA      100  <NA>  NA       NA
##  2:    RF     NA      90   <NA>  NA       NA
##  3:   SVM     10.4    NA   poly  2        NA
##  4:    RF     NA      50   <NA>  NA       NA
##  5:    RF     NA      60   <NA>  NA       NA
##  6:   SVM     89.9    NA   rbf   NA       4.2
##  7:   SVM     95.4    NA   poly  7        NA
##  8:    RF     NA      500  <NA>  NA       NA
##  9:    RF     NA      80   <NA>  NA       NA
## 10:   SVM     0.01    NA   rbf   NA       10.0
```

Tabelle 5.1: A random sample of size 10 from the hierarchical conditional configuration space of a random forest(RF) with hyper-parameter `n_tree` and a support vector machine (SVM) with hyper-parameter C and `kernel` type. NA means not applicable. Note when the kernel is `poly` representing polynomial kernel, there is an additional hyper-parameter n specifying the degree of the polynomial. When the kernel is `rbf`, there is an additional hyper-parameter `gamma`.

In terms of conventional hyper-parameter tuning, we showed in several contributions, Pfisterer et al. (2019); Sun et al. (2019a,c), the effectiveness of Bayesian Optimization on choice of hyper-parameters of machine learning algorithms. Specifically, in Sun et al. (2019a), we showed multi-objective bayesian optimization is effective in decentralized learning with distribution shift, where we set different objectives to be the performance of the machine learning model on different data sites and we coined the learning scenario Restrictive Federated Model Selection (RFMS).

For future work, it is interesting to extend RFMS Sun et al. (2019a) to multiple data types including images data, refinement of the R package for the ReinBo Sun et al. (2019c) algorithm. It is also interesting to see how different multi-objective optimization algorithms will perform for RFMS learning scenario. Learning theories can also be developed for RFMS.

Chapter 6

Functional Data Analysis and Model Selection

Functional Data Analysis (FDA) works on data with function structure as the unit of observation. The function structure can occur both in the covariates (the so called funcitonal covariates) and the response. This dissertation mostly deals with scalar response Functional Data.

From an ordinary linear regression perspective, Functional Data is characterized in its high dimensionality. The number of variables in one functional covariate, which are considered measurement grids in FDA literature, are usually more than the number of observations. This high dimensionality problem can be solved by projecting the functional covariate to basis functions like B-splines, which are both more computationally favorable and more interpretable compared to classical regularization approaches that can deal with high-dimensional data such as ridge regression. The basis functions encode our prior knowledge about the structure of the linear regression coefficient with respect to the functional covariate observation grids, the coordinate of the functional covariate now becomes low dimensional. Another characteristic of Functional Data is its correlation structure of measured values along the direction of the measurement grids. This correlation problem can be alleviated by enforcing a coefficient vector based on a smooth basis as a template to weigh different locations of the functional covariate Marx and Eilers (1999).

Both the high dimensionality and the correlation problems can be solved with basis functions like B-splines with controllable smoothness. In Section 6.1, the Functional Linear Model as an example to basic principles of solving the functional data regression problem is introduced. In Section 6.2, B-splines and a corresponding smoothness control method are covered. In Section 6.3, some selected FDA algorithms used in the contributing article Pfisterer et al. (2019) in Appendix 6 are summarized. Section 6.4 conclude the chapter and highlight some contributions to the community we made in the contributing article in Appendix G.

6.1 Functional Linear Model for Scalar on Function Regression

Consider a functional data sample $\{x_i(t), y_i\}_{i=1}^n$ where signal $x_i(t)$ as functional covariate for the ith observation, consisting measurement grids of size J_i, corresponds to scalar response y_i. A Functional Linear Model (FLM) Ramsay (2004) assumes

$$y_i = a + \int_t \beta(t) x_i(t) dt \tag{6.1}$$

, where both the regression function $\beta(t)$ and the functional covariate $x_i(t)$ can be projected into some low dimensional spaces spanned by basis functions. Let $\beta(t) = \sum_{k_b=1}^{K_\beta} b_{k_b}(t)\theta_{k_b}$, and $x_i(t) = \sum_{k_x=1}^{K_x} c_{i,k_x}\phi_k(t)$, we have

$$\begin{aligned} y_i &= a + \int_t \beta(t) x_i(t) dt && (6.2)\\ &= a + \int_t \left(\sum_{k_b=1}^{K_\beta} \theta_{k_b} b_{k_b}(t) \right) \left(\sum_{k_x=1}^{K_x} c_{i,k_x} \phi_{k_x}(t) \right) dt && (6.3)\\ &= a + \int_t \left\langle \vec{b}(t), \vec{\theta} \right\rangle \left\langle \vec{\phi}(t), C_i \right\rangle dt && (6.4)\\ &= a + C_i \Phi \vec{\theta} && (6.5) \end{aligned}$$

, where $\Phi_{k_x,k_b} = \int_t \phi_{k_x}(t) b_{k_b}(t) dt$ and $\vec{b}(t) = [b_1(t), \cdots, b_{K_\beta}(t)]^T$. $\vec{\theta} = [\theta_1, \cdots, \theta_{K_\beta}]^T$ and $C_i = [c_{i,1}, \cdots, c_{i,K_X}]$ is the i-th row of matrix C in Equation 6.6.

Taking n observations into consideration, the corresponding linear system for the low dimensional regression problem can be written as

$$Y = [1_n, C\Phi][a, \vec{\theta}^T]^T \tag{6.6}$$

FLM is a good example to the principles mentioned in the beginning of the chapter, yet in terms of Model Selection, K_β and K_x have to be chosen to specify the dimension of the subspaces. In the following section, B-splines are introduced as a basis system and how to control the effective dimension continuously for model selection is discussed.

6.2 Model Selection with Spline Smoother

6.2.1 B-splines

Let $\tau := \{\tau_k\}_{k=1}^K$ be a non-decreasing sequence of knots of length K (consider $K \to \infty$ for simplicity). B-splines $b_k^{d,\tau}(t)$ as basis functions in a domain indexed by t, with degree d,

indexed at knot k of $\{\tau_k\}_{k=1}^{K}$ can be computed recursively with respect to shift $k+1$ and lower degree $d-1$ in Equation 6.7 De Boor et al. (1978).

$$b_k^{d,\tau}(t) = \begin{cases} \frac{t-\tau_k}{\tau_{k+d}-\tau_k} b_k^{d-1,\tau}(t) + (1 - \frac{t-\tau_{k+1}}{\tau_{k+d+1}-\tau_{k+1}}) b_{k+1}^{d-1,\tau}(t) & d > 0 \text{ (divide by zero default to 0)} \\ I(\tau_k \leq t \leq \tau_{k+1}) & d = 0,\ \tau_k < \tau_{k+1} \\ 0 & d = 0,\ \tau_k = \tau_{k+1} \end{cases} \tag{6.7}$$

$b_k^{d,\tau}(t)$ consist of $d+1$ piece-wise polynomials, positively supported over $d+2$ knots or $d+1$ knot intervals. $d+1$ pieces join at d knots, so that the resulting spline is smooth up to its derivative of order d. At fixed t, only $d+1$ number of basis functions are nonzero and $\sum_k b_k^{d,\tau}(t) = 1$.

B-splines span the linear space of splines. An arbitrary spline $f^{d,\{c_{k'}\}_{k'=1}^{K'},\{\tau_k\}_{k=2}^{K'+d}}(t)$ of degree d interpolating K' control points $\{c_{k'}\}_{k'=1}^{K'}$ with $K'+d-1$ knots $\{\tau_k\}_{k=2}^{K'+d}$ can be represented via linear combination of B-splines as

$$f^{d,\{c_{k'}\}_{k=1}^{K'},\{\tau_k\}_{k=2}^{K'+d}}(t) = \sum_{k'=1}^{K'} \alpha_{k'} b_{k'}^{d,\tau}(t) \tag{6.8}$$

Consider fitting or interpolating J points $\{t_j, y_j\}_{j=1}^{J}$, with an arbitrary spline function $f^{d,\tau}(t)$ of degree d without control point specification, over knot sequence τ with lower boundary knot $\tau_l \leq t_1$ and upper boundary knot $\tau_u \geq t_J$, where $\tau_l \leq \tau_j \leq \tau_u$. $f^{d,\tau}(t)$ as a piece-wise function where each piece is a degree d polynomial has to join at each interior knot $\tau_l < \tau_j < \tau_u$ with continuous derivative of order up to d. Counting piece-wise, a unconstrained spline piece has $d+1$ Degrees of Freedom (DoF), passing at each internal knot, there are d constraints, so subsequent pieces each only have 1 remaining DoF, the total number of DoF is $(d+1) + (K-2) = K + d - 1$ where $K-2$ is the number of interior knots. The DoF has to be bigger or equal to the number of observations in case of interpolation. In fitting or smoothing, the DoF can be smaller than the number of observations. The bigger the DoF, the more flexible $f^{d,\tau}(t)$ can be in approximating the data.

If we were to use B-splines to represent $f^{d,\tau}(t)$, since $b_k^{d,\tau}(t)$ in Equation 6.7 is recursively defined with one knot shift per degree, there has to be extra d knots added to the left side of τ_l to define B-spline basis starting at τ_l. At arbitrary fixed t, including τ_l and τ_u, $d+1$ number of $b_j^{d,\tau}(t)$ are nonzero, and $\sum_{k'=k}^{k+d} b_{k'}^{d,\tau}(t) = 1$. An extra d knots can be added to the right side of τ_u so that B-spline is also defined at τ_{u+d} to make the sum to 1 possible also at τ_u so the sum to 1 applies everywhere in the internal knots interval. If we divide the range of $\{t_j\}$ with K'' interior knots, in total $2d + K''$ knots are needed to define the B-spline basis. Thus a projection of t on the B-spline basis can be calculated.

Given observations $\{t_j, s_j\}_{j=1}^{J}$, where s_j is the corresponding response to t_j, design matrix $B^{d,\tau}$ can be formed with $B_{j,k}^{d,\tau} = b_k^{d,\tau}(t_j)$. Using $\alpha = \{\alpha_k\}$ to represent the coordinate of $f^{d,\tau}(t)$ with respect to the B-spline basis $b_k^{d,\tau}(t)$, we have $\hat{s}_j(t_j) := \sum_k \alpha_k b_k^{d,\tau}(t_j)$

and a likelihood function $\ell(\hat{s}_j, s_j)$ can be defined. In case of Gaussian likelihood, the B-spline coordinate $\alpha = \{\alpha_k\}$ can be calculated as $\hat{\alpha} = \underset{\alpha}{arg\,min}(B^{d,\tau}\alpha - S)^T(B^{d,\tau}\alpha - S) = (B^TB)^{-1}_{J\times K}BS_{J\times 1}$ where $S = [s_1, s_2, \cdots, s_J]^T$ and B is the design matrix made of the B-splines basis.

6.2.2 Smoothness Selection

In Eilers and Marx (1996), the authors assume an equidistant knot sequence with $h = t_{k+1} - t_k$, so the second derivative (use $\mathcal{D}^{(2)}$ as the second derivative operator) of the spline smooth can be represented as

$$\mathcal{D}^{(2)} f^{d,\tau}(t) = \sum_k \alpha_k \mathcal{D}^{(2)} b_k^{d,\tau}(t) = \frac{1}{h^2}\sum_k (\Delta^{(2)}\alpha_k) b_k^{d-2,\tau}(t) \tag{6.9}$$

with $\Delta^{(2)}\alpha_k = \alpha_k - 2\alpha_{k-1} + \alpha_{k-2}$. Thus, the square of the second derivative as smoothness penalty integrated over the t domain $\lambda' \int_{\tau_l}^{\tau_u} \left(\mathcal{D}^{(2)} f^{d,\tau}(t)\right)^2 dt = \lambda' \int_{\tau_l}^{\tau_u} \left(\sum_j \alpha_j \mathcal{D}^{(2)} b_j^{d,\tau}(t)\right)^2 dt$ can be approximated with $\frac{c_1}{h^2}\lambda' \sum_k (\Delta^{(2)}\alpha_k)^2$, where $c_1 = \int_{\tau_l}^{\tau_u} \left(b_j^{1,\tau}(t)\right)^2 dt$ when using cubic splines and can be absorbed into the λ'. Cross terms with respect to shift $c_2 \sum_j \Delta^{(2)}\alpha_j \Delta^{(2)}\alpha_{k+1}$, where $c_2 = \int_{\tau_l}^{\tau_u} b_k^{1,\tau}(t) b_{k-1}^{1,\tau}(t) dt$ are dropped. The objective function to be optimized then becomes

$$L = \ell(\{t_j, s_j\}_{j=1}^J; \{\alpha_k\}_{k=1}^K, \tau) - \lambda/2 \sum_{k=d+1}^{K} (\Delta^{(2)}\alpha_k)^2 \tag{6.10}$$

$\{\Delta^{(2)}\alpha_k\}_{k=d+1}^K$ can be represented by a $(m-2)\times m$ upper triangular band matrix D with $D_{k,k} = 1, D_{k,k+1} = -2, D_{k,k+2} = 1$ and zero elsewhere, so $\sum_{k=d+1}^m (\Delta^{(2)}\alpha_k)^2 = (D\alpha)^T(D\alpha)$, and in the Gaussian likelihood case, the optimization reduces to a linear system of

$$B^TY = (B^TB + \lambda D^TD)\alpha \tag{6.11}$$

The smoother matrix (hat matrix) is

$$A = B(B^TB + \lambda D^TD)^{-1}B^T \tag{6.12}$$

The trace $tr(A)$ of the smoother matrix A, as sum of its eigenvalues can be used to represent the effective dimensions Hastie and Tibshirani (1990). So model selection criteria like AIC can be used to balance the trade off between deviance and model dimension (DoF).

6.3 Spline based models for Functional Data

6.3.1 Functional Generalized Additive Model (FGAM)

Functional Generalized Additive Model (FGAM) McLean et al. (2014) is defined as

$$g\{\mathbb{E}(Y_i|\tilde{X}_i)\} = \theta_0 + \int F\{X_i(t), t\} dt \tag{6.13}$$

, where i is the observation index, $\tilde{X}_i$ is the functional covariate of the ith observation, measured in grids indexed by t with value $X_i(t)$ at measurement grid t. Y_i is the scalar response of the ith observation, with global link function g transforming the conditional mean of response Y_i to be addition of the intercept θ_0 with additive effect $F\{X_i(t), t\}$ along measurement grids.

A special case of FGAM is when

$$F\{X_i(t), t\} = X_i(t)\beta(t) \tag{6.14}$$

, where $\beta(t)$ defines a constant coefficient at time t with respect to any $X_i(t)$ value, which correspond to the Functional Linear Model (FLM). In practice, $F\{X_i(t), t\}$ is estimated using tensor-product B-splines, with K_X basis functions in the $X(t)$ direction marginal B-spline, and K_T basis functions in the t direction marginal B-spline. $F(x,t) = \sum_{k_x=1}^{K_X}\sum_{k_t=1}^{K_T}\theta_{k_x,k_t}B_{k_x}^X(X_i(t))B_{k_t}^T(t)$, which maps the measurement value $X_i(t)$ at the measurement grid t to the additive effect $F\{X_i(t), t\}$ through $X_i(t) \to B_{k_x}^X(X_i(t))$ and $t \to B_{k_t}^T(t)$, wth θ_{k_x,k_t} being the coefficient. The integral in Equation 6.13 thus becomes

$$\int F\{X_i(t), t\}dt = \sum_{k_x=1}^{K_X}\sum_{k_t=1}^{K_T}\theta_{k_x,k_t}\int B_{k_x}^X(X_i(t))B_{k_t}^T(t)dt = \sum_{k_x=1}^{K_X}\sum_{k_t=1}^{K_T}\theta_{k_x,k_t}Z^{k_x,k_t}(X_i(t),t) \tag{6.15}$$

, where $Z^{k_x,k_t}(X_i(t),t) = \int B_{k_x}^X(x)B_{k_t}^T(t)dt$. Therefore, $g\{\mathbb{E}(Y_i|\tilde{X}_i)\}$ is θ_0 plus a path integral of $\tilde{X}_i$ along surface $F\{X_i(t), t\}$. Roughly speaking, in the language of FLM, at measurement grid t, the $\beta(t)$ equivalent of FGAM in Equation 6.14 is not constant with respect to every measurement value $X_i(t)$, thus enabling flexibility and non-linearity.

However, this flexibility of FGAM over FLM also comes with expenses. Since measurement value $X_i(t)$ has different ranges at different measurement grid t, a single knot sequence corresponding to basis dimension K_X spanning the largest range in the $X_i(t)$ direction can result in zero columns in the tensor product design matrix composed of $Z^{k_x,k_t}(X_i(t)) = \int B_{k_x}^X(X_i(t))B_{k_t}^T(t)dt$. Note B-spline basis of degree d is only supported on $d+1$ knot intervals, thus for some (k_x, k_t) combination, $Z^{k_x,k_t}(X_i(t)) = \int B_{k_x}^X(X_i(t))B_{k_t}^T(t)dt$ will always be zero since it can be that $X_i(t)$ for any observation i has no value at the interval supporting $B_{k_x}^X$. The solution is to transform the functional covariate $\tilde{X}$ to have equal range at each measurement grid t. Then $B_j^X(X(t)) : k_x = 1, \cdots, K_X$ are spline basis on $[0, 1]$.

For calculation of the design matrix entry, $Z^{k_x,k_t}(X_i(t)) = \int B_{k_x}^X(X(t))B_{k_t}^T(t)dt$ can be approximated by Simpson's rule in the form based on available measurement grids in Equation 6.16.

$$Z^{k_x,k_t}(X_i(t)) = \int B_{k_x}^X(X(t))B_{k_t}^T(t)dt \approx \sum_{j=1}^{J_i}B_{k_x}^X(X_i(t_{ij}))B_{k_t}^T(t_{ij})\delta_{ij} \tag{6.16}$$

, where δ_{ij} is the Simpson integration weight for the measurement grid t_{ij} of observation i, J_i is the size of the measurement grids for observation i. Therefore, Equation 6.13 becomes a low dimensional regression problem

$$g\{\mathbb{E}(Y_i|\tilde{X}_i)\} = [1, \mathbb{Z}_i] \begin{bmatrix} \theta_0 \\ \theta \end{bmatrix} \tag{6.17}$$

6.3.2 Model Selection with Boosting: Funtional Linear Array Model (FLAM)

Functional Linear Array Model (FLAM) Brockhaus et al. (2015) models the conditional relationship of the ith functional response $\tilde{y}_i$ with respect to the ith functional covariate $\tilde{x}_i$ as an additive model of base learners $h^l(\tilde{x})$ in Equation 6.18, where l is the base learner index of in total L base learners, i is the observation index.

$$\zeta(\tilde{y}_i|\tilde{x}_i) = \sum_{l=1}^{L} h^l(\tilde{x}_i)(t) \tag{6.18}$$

$\tilde{x}_i \in \mathcal{X}$ comes from the space of functional covariate $\mathcal{X}$, response $\tilde{y} \in \mathcal{Y} \subset \mathcal{L}^2(\mathcal{T}, \mu)$, with $\mathcal{T}$ representing a time interval, μ representing the measure defined on it and $\mathcal{L}^2(\mathcal{T}, \mu)$ represent the space of square integrable functions. For scalar on function regression, $\mathcal{T}$ reduces to a single point and μ becomes dirac measure. In Generalized Linear Model terminology, the transform ζ can be the concatenation of a link function g and expectation: $\zeta = g \circ \mathbb{E}_{\tilde{x},\tilde{y}}$. The base learner $h^l(\tilde{x}_i)(t)$ in FLAM is defined as Kronecker Product of functional covariate effect and response effect as

$$h^l(\tilde{x}_i)(t) = \left([b^l(\tilde{x}_i)]^T \otimes [b^l_Y(t)]^T\right) \theta^l \tag{6.19}$$

When considering scalar on function regression, $[b^l_Y(t)]^T = [1]$, an example of $[b^l(\tilde{x}_j)^T]\theta^l$ can be

$$h^l(\tilde{x}_i) = [b^l(\tilde{x}_i)]^T \vec{\theta^l} \simeq \int_s x_i(s)\beta^l(s)ds \approx [\delta^j(s_1)x_i(s_1), \cdots, \delta^j(s_R)x_i(s_R)]\Phi_l^{R\times K_l}\vec{\theta^l} \tag{6.20}$$

, where the effect $[b^l(\tilde{x}_i)]^T = [\delta^l(s_1)x_i(s_1), \cdots, \delta^l(s_R)x_i(s_R)]\Phi_l^{R\times K_l}$, taken inner product with $\vec{\theta}$, is an Riemann integration approximation to $\int_s x_i(s)\beta^l(s)ds$ with $\delta(s_r)$ being the integration weight. The linear coefficient functional $\beta^l(s)$ is taken to be a linear combination of the B-spline basis of dimension K_l, with $\Phi_l^{R\times K_l}$ being the B-spline basis evaluated at the observation grids of length R.

FLAM is estimated via component wise gradient boosting, which is an alternative method to estimate (generalized) additive models. At each boosting iteration, all base learners representing different functional effects as different components are fitted to the gradient of the dedicated loss, with respect to which the best base learner is selected and the fitted θ is being incremented.

In fitting of the base learners to the gradient of the dedicated loss, FLAM extends Generalized Linear Array Model (GLAM) Currie et al. (2006) to functional data.

Estimation with component wise boosting enables FLAM to deal with mixed data type and datasets of less observations than covariates. Additionally, the capability from component wise boosting to do variable selection and model selection is also inherited.

6.4 Contributions and Prospects

In Pfisterer et al. (2019) in Appendix G, we integrated many Functional Data Analysis methods into the `mlr` package, and benchmarked their performance. Our work aims at building a bridge between the Functional Data Analysis community and the time series analysis community. We showed the effectiveness of tuning algorithm hyper-parameters for Functional Data.

For future work, it is interesting to see a deeper and interpretable analysis of how different Functional Data Analysis methods work, and in which scenario they work better.

Chapter 7

Information Theory for Reinforcement Learning and Feature Filtering

7.1 Introduction

This chapter introduces basic concepts and properties of information theory and how to use it for feature filtering in the contributing article Bommert et al. (2020). Information theory is also used in efficient exploration in deep reinforcement learning in the contributing articles Sun and Bischl (2019), in constructing objective function in the contributing article Zhao et al. (2019). In Appendix I.2, basic information theory is introduced.

7.2 Mutual Information and Empowerment in Reinforcement Learning

The definition of mutual information can be written as in Equation 7.1, which depends on $p(x)$ and $p(y|x)$.

$$I(X,Y) = E_{p(x,y)} \log \frac{p(x,y)}{p(x)p(y)} = E_{p(x),p(y|x)} \log \frac{p(x)p(y|x)}{p(x)\sum_x p(x)p(y|x)} \tag{7.1}$$

When modeling random variable X as information sender and random variable Y as information receiver through a communication channel, the capacity of the communication channel is defined to be the biggest possible mutual information between the sender and the receiver as in Equation 7.2.

$$C(X \to Y) = \max_{p(x)} I(X,Y) \tag{7.2}$$

Such a capacity definition can be used as a measure of causal influence of X to Y Klyubin et al. (2005). In reinforcement learning , the vital causal influence is the effects of a series of actions taken to the future state distribution, which is defined to be the empowerment using

the capacity formula in Equation 7.2 Klyubin et al. (2005). Let $\tilde{A}_t^J = \{A_t, \ \cdots, A_{t+J-1}\}$ be a series of actions of length J, with its lower case counterpart $\tilde{a}_t^J$ be the realization of the capitalized version. Let S_{t+J} be the state random variable after J actions with its lower case counterpart s_{t+J} be the realization, empowerment is defined to be the channel capacity of sending information from the series of actions of length J to the state distribution after J steps as in Equation 7.3.

$$\text{Empowerment}(t, J) = C(\tilde{A}_t^J \to S_{t+J}) \tag{7.3}$$

Additionally, as introduced in the contributing article Sun and Bischl (2019), mutual information can also be used to augment intrinsic reward signals to encourage exploration in the sparse reward scenario Houthooft et al. (2016).

7.3 Mutual Information based Feature Filtering

In Brown et al. (2012), a systematic perspective on feature selection is given by modeling the joint distribution of covariate x and response y in a likelihood function $\ell(y|x) = \log p(y|x)$ and uses a k-dimensional binary vector θ indicating if the feature is selected or not, so x_θ is the features selected and $x_{\tilde{\theta}}$ is the unselected features. Then the feature selection problem is connected to maximizing the conditional likelihood $p(y|x)$, $\theta^* = \underset{\theta}{argmax}\, \ell(y|x_\theta)$. $p(y|x)$ can be approximated by hypothetical predicative model $q(y|x, \tau)$ with τ represent predicative parameters, which can be represented by the same graphical model or in other words, they have the same conditional independence structure.

Use the minus one term and plus one term trick twice, one gets

$$\begin{aligned}
\ell &= \frac{1}{N}\sum \log q(y^i|x_\theta^i, \tau) && (7.4)\\
&= \frac{1}{N}\sum \log \frac{q(y^i|x_\theta^i, \tau)}{p(y_i|x_\theta^i)} + \frac{1}{N}\sum \log p(y_i|x_\theta^i) && (7.5)\\
&= \frac{1}{N}\sum_i \log \frac{q(y^i|x_\theta^i, \tau)}{p(y_i|x_\theta^i)} + \frac{1}{N}\sum_i \log \frac{p(y_i|x_\theta^i)}{p(y_i|x_i)} + \frac{1}{N}\sum_i \log p(y_i|x_i) && (7.6)\\
&\approx E_{x,y}\frac{q(y^i|x_\theta^i, \tau)}{p(y_i|x_\theta^i)} + E_{x,y}\log \frac{p(y|x_\theta)}{p(y|x)} + E_{x,y}H(Y|X) && (7.7)
\end{aligned}$$

where the sum in Equation 7.6 approximate the expectation with respect to $p(x, y)$ in Equation 7.7. The first term can be viewed as the KL divergence between $q(y^i|x_\theta^i, \tau)$ and $p(y_i|x_\theta^i)$ when θ is optimal and the third term is the constant which is the irreducible since x could not fully explain y.

The second term in Equation 7.7 can be further expanded.

$$\frac{1}{N}\sum \log \frac{p(y_i|x_\theta^i)}{p(y_i|x_i)} \approx E_{xy} \log \frac{p(y|x_\theta)}{p(y|x)} = \frac{1}{N}\sum p_{xy} \log \frac{p(y|x_\theta)}{p(y|x)} \frac{p(x_{\tilde{\theta}}|x_\theta)}{p(x_{\tilde{\theta}}|x_\theta)} \tag{7.8}$$

$$= \frac{1}{N}\sum p_{xy} \log \frac{p(y|x_\theta)}{1} \frac{p(x_{\tilde{\theta}}|x_\theta)}{p(y; x_{\tilde{\theta}}|x_\theta)} \tag{7.9}$$

$$= -I(x_{\tilde{\theta}}; y|x_\theta) \tag{7.10}$$

where $I(x_{\tilde{\theta}}; y|x_\theta) = \sum p(x_{\tilde{\theta}}; y) \log \frac{p(y;x_{\tilde{\theta}}|x_\theta)}{p(x_{\tilde{\theta}}|x_\theta)p(y|x_\theta)}$ which is the conditional mutual information between the irrelevant feature $x_{\tilde{\theta}}$ and the label, conditional on the selected features x_θ and Equation 7.9 is because $p(y|x)p(x_{\tilde{\theta}}|x_\theta) = p(y|x_{\tilde{\theta}}, x_\theta)p(x_{\tilde{\theta}}|x_\theta) = \frac{p(x_{\tilde{\theta}},y,x_\theta)}{p(x_\theta)} = p(x_{\tilde{\theta}}, y|x_\theta)$

$\theta^* = \underset{\theta}{argmax}\ \ell(y|x_\theta)$ can be done by retrofitting features conditioned on the selected features $\theta^{(r)}$ in the rth iteration. For example, retrofitting feature selection can be based on Conditional Mutual Information (CMI) $J_{cmi}(x_k) = I(x_k, y|x_{\theta^{(r)}}) = I(X_k, x_\theta|y) - I(X_k, x_\theta) + I(X_k, y)$ due to the identity $I(X, Y|Z) - I(X, Y) = I(X, Z|Y) - I(X, Z)$ in Equation I.50.

For continuous features, estimation of the mutual information is done through feature discretization using the entropy minimization split with respect to the class label Fayyad and Irani (1993) following the Minimal Description Length (MDL) Shalev-Shwartz and Ben-David (2014).

7.4 Contributions and Prospects

In the contributing article Bommert et al. (2020) we provide a comprehensive benchmark study across several families of filtering based methods as feature selection methods. For future work, more recently proposed non-filtering based methods can be added to the benchmarks. In the contributing article Zhao et al. (2019), we used weighted entropy as optimization objective in deep reinforcement learning, and proposed its lower bound approximation. The entropy improvement is also proved. In the contributing article Sun and Bischl (2019), we summarized several use cases of information and entropy in deep reinforcement learning.

Chapter 8

References

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

Bischl, B., Kerschke, P., Kotthoff, L., Lindauer, M., Malitsky, Y., Fréchette, A., Hoos, H., Hutter, F., Leyton-Brown, K., Tierney, K., et al. (2016). Aslib: A benchmark library for algorithm selection. *Artificial Intelligence*, 237:41–58.

Bischl, B., Wessing, S., Bauer, N., Friedrichs, K., and Weihs, C. (2014). Moi-mbo: multiobjective infill for parallel model-based optimization. In *International Conference on Learning and Intelligent Optimization*, pages 173–186. Springer.

Bishop, C. M. (2006). Pattern recognition and machine learning.

Blei, D. M., Kucukelbir, A., and McAuliffe, J. D. (2017). Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112(518):859–877.

Bommert, A., Sun, X., Bischl, B., Rahnenfuehrer, J., and Lang, M. (2020). Benchmark for filter methods for feature selection in high-dimensional classification data. *Computational Statistics & Data Analysis*, 143:106839.

Boudarel, R., Delmas, J., and Guichet, P. (1971). *Dynamic programming and its application to optimal control*. Elsevier.

Brockhaus, S., Scheipl, F., Hothorn, T., and Greven, S. (2015). The functional linear array model. *Statistical Modelling*, 15(3):279–300.

Brown, G., Pocock, A., Zhao, M.-J., and Luján, M. (2012). Conditional likelihood maximisation: a unifying framework for information theoretic feature selection. *The journal of machine learning research*, 13(1):27–66.

Cawley, G. C. and Talbot, N. L. (2010). On over-fitting in model selection and subsequent selection bias in performance evaluation. *The Journal of Machine Learning Research*, 11:2079–2107.

Co-Reyes, J. D., Liu, Y., Gupta, A., Eysenbach, B., Abbeel, P., and Levine, S. (2018). Self-consistent trajectory autoencoder: Hierarchical reinforcement learning with trajectory embeddings. *arXiv preprint arXiv:1806.02813*.

Currie, I. D., Durban, M., and Eilers, P. H. (2006). Generalized linear array models with applications to multidimensional smoothing. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(2):259–280.

Dächert, K., Gorski, J., and Klamroth, K. (2010). An adaptive augmented weighted tchebycheff method to solve discrete, integer-valued bicriteria optimization problems. Technical report, Technical Report BUWAMNA-OPAP 10/06, University of Wuppertal, FB Mathematik.

De Boor, C., De Boor, C., Mathématicien, E.-U., De Boor, C., and De Boor, C. (1978). *A practical guide to splines*, volume 27. springer-verlag New York.

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197.

Degris, T., White, M., and Sutton, R. S. (2012). Off-policy actor-critic. *arXiv preprint arXiv:1205.4839*.

Dietterich, T. G. (2000). Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of artificial intelligence research*, 13:227–303.

Eilers, P. H. and Marx, B. D. (1996). Flexible smoothing with b-splines and penalties. *Statistical science*, pages 89–102.

Fayyad, U. and Irani, K. (1993). Multi-interval discretization of continuous-valued attributes for classification learning.

Fujimoto, S., Meger, D., and Precup, D. (2019). Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*, pages 2052–2062. PMLR.

Fujimoto, S., Van Hoof, H., and Meger, D. (2018). Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*.

Gossmann, A., Cha, K. H., and Sun, X. (2019). Variational inference based assessment of mammographic lesion classification algorithms under distribution shift. In *NeurIPS 2019 workshop: Medical Imaging meets NeurIPS 2019*.

Haarnoja, T., Tang, H., Abbeel, P., and Levine, S. (2017). Reinforcement learning with deep energy-based policies. *arXiv preprint arXiv:1702.08165*.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*.

Hastie, T. J. and Tibshirani, R. J. (1990). *Generalized additive models*, volume 43. CRC press.

Hoffman, J., Tzeng, E., Park, T., Zhu, J.-Y., Isola, P., Saenko, K., Efros, A., and Darrell, T. (2018). Cycada: Cycle-consistent adversarial domain adaptation. In *International conference on machine learning*, pages 1989–1998. PMLR.

Hoffman, M. D., Blei, D. M., Wang, C., and Paisley, J. (2013). Stochastic variational inference. *The Journal of Machine Learning Research*, 14(1):1303–1347.

Horn, D., Wagner, T., Biermann, D., Weihs, C., and Bischl, B. (2015). Model-based multi-objective optimization: taxonomy, multi-point proposal, toolbox and benchmark. In *International Conference on Evolutionary Multi-Criterion Optimization*, pages 64–78. Springer.

Houthooft, R., Chen, X., Duan, Y., Schulman, J., De Turck, F., and Abbeel, P. (2016). Vime: Variational information maximizing exploration. *Advances in neural information processing systems*, 29:1109–1117.

Jeong, S. and Obayashi, S. (2005). Efficient global optimization (ego) for multi-objective problem and data mining. In *2005 IEEE congress on evolutionary computation*, volume 3, pages 2138–2145. IEEE.

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

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Klyubin, A. S., Polani, D., and Nehaniv, C. L. (2005). Empowerment: A universal agent-centric measure of control. In *2005 IEEE Congress on Evolutionary Computation*, volume 1, pages 128–135. IEEE.

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

Kulkarni, T. D., Narasimhan, K., Saeedi, A., and Tenenbaum, J. (2016). Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *Advances in neural information processing systems*, 29:3675–3683.

Kumar, A., Fu, J., Soh, M., Tucker, G., and Levine, S. (2019). Stabilizing off-policy q-learning via bootstrapping error reduction. In *Advances in Neural Information Processing Systems*, pages 11784–11794.

Levine, S. (2018). Reinforcement learning and control as probabilistic inference: Tutorial and review. *arXiv preprint arXiv:1805.00909*.

Li, D., Yang, Y., Song, Y.-Z., and Hospedales, T. M. (2017). Deeper, broader and artier domain generalization. In *Proceedings of the IEEE international conference on computer vision*, pages 5542–5550.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Lin, L.-J. (1991). Self-improvement based on reinforcement learning, planning and teaching. In *Machine Learning Proceedings 1991*, pages 323–327. Elsevier.

Marx, B. D. and Eilers, P. H. (1999). Generalized linear regression on sampled signals and curves: a p-spline approach. *Technometrics*, 41(1):1–13.

Matheron, G., Perrin, N., and Sigaud, O. (2019). The problem with ddpg: understanding failures in deterministic environments with sparse rewards. *arXiv preprint arXiv:1911.11679*.

McLean, M. W., Hooker, G., Staicu, A.-M., Scheipl, F., and Ruppert, D. (2014). Functional generalized additive models. *Journal of Computational and Graphical Statistics*, 23(1):249–269.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529.

Mockus, J. (2012). *Bayesian approach to global optimization: theory and applications*, volume 37. Springer Science & Business Media.

Nalisnick, E., Matsukawa, A., Teh, Y. W., Gorur, D., and Lakshminarayanan, B. (2018). Do deep generative models know what they don't know? *arXiv preprint arXiv:1810.09136*.

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

Plappert, M., Andrychowicz, M., Ray, A., McGrew, B., Baker, B., Powell, G., Schneider, J., Tobin, J., Chociej, M., Welinder, P., et al. (2018). Multi-goal reinforcement learning: Challenging robotics environments and request for research. *arXiv preprint arXiv:1802.09464*.

Ponweiser, W., Wagner, T., Biermann, D., and Vincze, M. (2008). Multiobjective optimization on a limited budget of evaluations using model-assisted s-metric selection. In *International Conference on Parallel Problem Solving from Nature*, pages 784–794. Springer.

Quiñonero-Candela, J., Sugiyama, M., Schwaighofer, A., and Lawrence, N. D. (2009). On the training/test distributions gap: A data representation learning framework.

Quionero-Candela, J., Sugiyama, M., Schwaighofer, A., and Lawrence, N. D. (2009). *Dataset shift in machine learning*. The MIT Press.

Ramsay, J. O. (2004). Functional data analysis. *Encyclopedia of Statistical Sciences*, 4.

Rasmussen, C. E. (2003). Gaussian processes in machine learning. In *Summer school on machine learning*, pages 63–71. Springer.

Ren, J., Liu, P. J., Fertig, E., Snoek, J., Poplin, R., Depristo, M., Dillon, J., and Lakshminarayanan, B. (2019). Likelihood ratios for out-of-distribution detection. In *Advances in Neural Information Processing Systems*, pages 14707–14718.

Riedmiller, M., Hafner, R., Lampe, T., Neunert, M., Degrave, J., Van de Wiele, T., Mnih, V., Heess, N., and Springenberg, J. T. (2018). Learning by playing-solving sparse reward tasks from scratch. *arXiv preprint arXiv:1802.10567*.

Sallans, B. and Hinton, G. E. (2004). Reinforcement learning with factored states and actions. *Journal of Machine Learning Research*, 5(Aug):1063–1088.

Sastry, C. S. and Oore, S. (2020). Detecting out-of-distribution examples with gram matrices. In *International Conference on Machine Learning*, pages 8491–8501. PMLR.

Schulman, J., Chen, X., and Abbeel, P. (2017). Equivalence between policy gradients and soft q-learning. *arXiv preprint arXiv:1704.06440*.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015). Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897.

Shalev-Shwartz, S. and Ben-David, S. (2014). *Understanding machine learning: From theory to algorithms*. Cambridge university press.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms.

Sun, X. and Bischl, B. (2019). Tutorial and survey on probabilistic graphical model and variational inference in deep reinforcement learning. In *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*, number 1908.09381.

Sun, X., Bommert, A., Pfisterer, F., Rahnenführer, J., Lang, M., and Bischl, B. (2019a). High dimensional restrictive federated model selection with multi-objective bayesian optimization over shifted distributions. In *Intelligent Systems and Applications*, pages 629–647. Springer International Publishing, Cham.

Sun, X. and Buettner, F. (2021). Hierarchical variational auto-encoding for unsupervised domain generalization.

Sun, X., Gossmann, A., Wang, Y., and Bischl, B. (2019b). Variational resampling based assessment of deep neural networks under distribution shift. In *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*, number 1906.02972.

Sun, X., Lin, J., and Bischl, B. (2019c). Reinbo: Machine learning pipeline conditional hierarchy search and configuration with bayesian optimization embedded reinforcement learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 68–84. Springer, Cham.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Van Hasselt, H., Guez, A., and Silver, D. (2016). Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30.

Van Hoof, H., Chen, N., Karl, M., van der Smagt, P., and Peters, J. (2016). Stable reinforcement learning with autoencoders for tactile and visual data. In *2016 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 3928–3934. IEEE.

Vapnik, V. (2013). *The nature of statistical learning theory*. Springer science & business media.

Von Luxburg, U. and Schölkopf, B. (2011). Statistical learning theory: Models, concepts, and results. In *Handbook of the History of Logic*, volume 10, pages 651–706. Elsevier.

Wood, S. N. (2017). *Generalized additive models: an introduction with R*. Chapman and Hall/CRC.

Zhao, R., Sun, X., and Tresp, V. (2019). Maximum entropy-regularized multi-goal reinforcement learning. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 7553–7562, Long Beach, California, USA. PMLR.

Appendix A

Variational Resampling Based Assessment of Deep Neural Networks under Distribution Shift

Contributing Article Sun, Xudong; Gossmann, Alexej; Wang, Yu; Bischl, Bernd; Variational Resampling Based Assessment of Deep Neural Networks under Distribution Shift, 2019 IEEE Symposium Series on Computational Intelligence (SSCI), 1906.02972, 2019.

Author Contributions Xudong Sun initiated the idea, designed the algorithm and workflow. He wrote the first version of the manuscript. He refactored codes for the Bayesian CNN and implemented wasserstein distance calculation, maintained and refactored the whole code base. Alexej Gossmann generated most of the experimental results with plots and tables used in the paper independently. He contributed most of the experimental parts of the paper, discussed core ideas and mathematical expressions in the paper with Xudong Sun, and elaborated intensively on the manuscript. Xudong Sun and Alexej jointly resolved several vital implementation bugs in the code base. Yu Wang implemented cluster labeling, cross validation with merge and `tnse` visualization, refactored the code of VAE,

while Xudong Sun iterated on the same codes and refined Yu's work for usability.

Follow-up study Gossmann, Alexej; Cha, Kenny Heekon; Sun, Xudong; Variational inference based assessment of mammographic lesion classification algorithms under distribution shift, NeurIPS 2019 workshop: Medical Imaging meets NeurIPS 2019.

Variational Resampling Based Assessment of Deep Neural Networks under Distribution Shift

1st Xudong Sun*, 1st Alexej Gossmann†, 3nd Yu Wang‡, and 4th Bernd Bischl§
* *Department of Statistics*
Ludwig Maximillian University of Munich, Munich, Germany
smilesun.east@gmail.com
† *Center for Devices and Radiological Health*
U.S. Food and Drug Administration, Silver Spring, MD, USA
alexej.gossmann@fda.hhs.gov
‡ *Technical University of Munich*, Munich, Germany
§ *Department of Statistics*
Ludwig Maximillian University of Munich, Munich, Germany
bernd.bischl@stat.uni-muenchen.de

***Abstract*—A novel variational inference based resampling framework is proposed to evaluate the robustness and generalization capability of deep learning models with respect to distribution shift. We use Auto Encoding Variational Bayes to find a latent representation of the data, on which a Variational Gaussian Mixture Model is applied to deliberately create distribution shift by dividing the dataset into different clusters. Wasserstein distance is used to characterize the extent of distribution shift between the generated data splits. In experiments using the Fashion-MNIST data, we assess several popular image classification Convolutional Neural Network (CNN) architectures and Bayesian CNN models with respect to their robustness and generalization behavior under the deliberately created distribution shift, which is analyzed in contrast to random Cross Validation. Our method of creating artificial domain splits of a single dataset may also be used to establish novel model selection criteria and assessment tools in machine learning, as well as for benchmark methods in the areas of domain adaptation and domain generalization.**

***Keywords*—Bayesian CNN, Variational Inference, Resampling, Distribution Shift, Wasserstein Distance, Domain Adaptation, Domain Generalization, Transfer Learning, Model Selection, Cross Validation, Generalization, Robustness**

I. Introduction

Recent studies have shown that deep learning methods may not generalize well beyond the training data distribution. For instance, deep learning models are vulnerable to adversarial perturbations [1], are prone to biases and unfairness [2], or may significantly but unknowingly depend on confounding variables resulting from the training data collection process [3]. In this work we focus on distribution shift, which is another important phenomenon that can have a significant negative impact on the performance of deep learning models [4]. Addressing the problems related to distribution shift is especially crucial for medical applications of machine learning [5], [6], and other high-risk application areas.

Areas of machine learning research related to distribution shift include Transfer learning, which is the process of improving the predictive performance on a target domain by using related information from the source domain [7]. Domain Adaptation adapts the source domain distribution to the target domain distribution to improve the performance of a target learner in transfer learning . In contrast to domain adaptation, Domain Generalization is the process of utilizing data from several domains to train a system that will generalize to previously unseen domains [8].

Although several domain adaptation and domain generalization benchmark datasets exist [9], they are either curated by human experts as combinations of multiple datasets with distribution shifts available a priori, or obtained through specific data manipulation techniques such as rotations [8]. Therefore, these datasets depend on domain knowledge and are restricted to specific tasks. For new applications, collecting datasets with distribution shift for evaluation of algorithms may be expensive or even intractable. Thus, to facilitate the study of distribution shift, as well as domain adaptation and domain generalization, there is a need for a general and efficient method to create benchmark datasets for evaluation of these approaches.

Furthermore, the robustness of machine learning models to distribution shift between subsets of the same dataset or subdomains of a single domain seems to not have been studied to a sufficient extent in the past. For the various practical applications of machine learning these subsets or subdomains may also represent different sources of data, subpopulations within the target population, as well as other types of stratification, and the variability in performance of machine learning systems between them is often not considered although it is substantial in many cases.

While Cross Validation is a widely used resampling technique for model selection in Machine Learning [10], [11] which generates random splits on a datasets, a resampling technique that can generate splits with distribution shift to evaluate a machine learning model does not seem to be known. Thus, inspired by the resampling technique used in Restrictive Federated Model Selection over shifted distribution [5] we propose a resampling technique to artificially create pseudo subdomains, which can serve as a benchmark method

to evaluate distribution shift related problems and potential solutions.

Specifically, in this work we are interested in characterizing changes in model test performance under distribution shift over different subsets of the same dataset, i.e., when the feature distribution (but not the label distribution) of the test data is shifted relative to the distribution of features in the training data although both are subsets of the same dataset.

Our major contributions are:

- We propose a resampling framework, operating on a given dataset, which creates splits corresponding to different distributions by estimating the pseudo domain label of each instance using variational inference. We use the Wasserstein distance to quantify the amount of distribution shift created by our method. Our method can be used for the creation of benchmark datasets for domain adaptation and domain generalization.
- We assess the robustness and generalization behavior of several image classification CNN architectures, including their corresponding Bayesian versions, under the distribution shift artificially created by our resampling method. Comparisons with random Cross Validation show that our method can efficiently create distribution shift. The proposed resampling approach can be used to aid in model selection, where it targets robustness and generalizability.

II. Prerequisite

Auto-encoding Variational Bayes, and Variational Autoencoder (VAE) [12]: To model the likelihood of data $p(x)$, a latent variable model, characterized by the posterior $p(z|x)$, is approximated by a variational posterior distribution $q_\phi(z|x)$, where the latent variable in the variational posterior is reparameterized as $z = g(\epsilon, x)$ with an auxiliary random variable $\epsilon \sim p(\epsilon)$ following an appropriate distribution. The conditional distribution $p_\theta(x|z)$ can be modeled as a Gaussian distribution with mean and variance parameters computed from z by a decoder neural network. Evidence lower bound (ELBO) of the likelihood is optimized with respect to θ and ϕ using stochastic gradient descent (SGD) over the Monte Carlo estimation. As in an Auto Encoder [13] the auxiliary variable z also serves as a latent representation of an instance.

Bayesian Neural Network [14]: Different from generative representation learning methods such as VAE, which model a variational approximator to the model posterior on the hidden units, a Bayesian Neural Network builds variational models on the weights of the network, which can also be used for exploration in Reinforcement Learning [14], [15]. The negative of ELBO, which is the variational free energy $F(D, \phi)$, is optimized. In particular, $F(D, \phi) = KL(q(w|\phi)||p(w)) - E_q[\log p(D \mid w)]$, where D is the data, w is the vector of weights, ϕ represents the variational mean and variance parameters for the weight distribution, $KL(q(w|\phi)||p(w))$ is the KL-divergence between the prior distribution $p(w)$ and the variational posterior $q_\phi(w|D)$, and $E_q[\log p(D \mid w)]$ is the expectation of log-likelihood under the distributional distribution. With the variational free energy $F(D, \phi)$ as loss function, where weights w are implicitly represented by ϕ, backpropagation with respect to the weights could be translated to variational parameter. Like Auto-encoding Variational Bayes, Bayes by Backprop [14] also starts from an independent noise distribution, but instead of transforming the noise together with observation data to latent units, Bayes By Backprop associates each weight with a variational mean and scale parameter to mix with the noise. Bayesian Convolutional Neural Network with Variational Inference (Bayesian CNN) [16] extends the Bayes By Backprop approach [14] to CNNs and utilize the local reparameterization trick [17]–[19] which we will elaborate in the method section.

Variational Gaussian Mixture Model: Variational Learning of Gaussian Mixture Models (VGMM) [20] uses joint Normal-Wishart distributions for the means and inverse covariance matrices in a mixture of Gaussians, and a Dirichlet distribution for the mixing parameters. Instead of a point estimate of the mean vector, VGMM uses a Normal distribution characterized by the hypermean. VGMM result in a superior data estimation compared to simple Gaussian Mixtures.

Distribution Shift: Let the random vector x represent the features and let the random variable y be the class label. In this work, we investigate the conditional distribution shift over $p(x|y)$ between datasets (e.g., between the training and the test data), while the marginal distribution $p(y)$ is shared across all datasets (cf. [21]).

Wasserstein Distance: Wasserstein distance between two distributions p_x and p_y can be defined as $\phi_W(p_x, p_y) = \inf_{\gamma \in \prod(p_x, p_y)} \underset{(x,y)\sim\gamma(x,y)}{E}[c(x, y)]$ [22]–[24], where $\gamma \in \prod(p_x, p_y)$ is the transportation plan or joint distribution of (x, y), with marginal distributions p_x and p_y respectively, and $c(x, y)$ is the cost of moving x to y. The Wasserstein distance is calculated by taking the infimum with respect to the transportation plan $\gamma \in \prod(p_x, p_y)$. Wasserstein distance can be approximated to optimize Generative Adversarial Networks [25]. Compared to KL divergence, Wasserstein distance experiences no numerical problems even when the two distributions have no overlap. Hence, we use the Wasserstein distance to measure the distribution shift between two subsets of data on the latent space.

t-SNE: Stochastic Neighborhood Embedding [26] uses a Gaussian density to model the conditional similarity between two points in a high dimensional space and a corresponding low dimensional embedding. The KL-divergence between the conditional similarity distributions is used as objective and is optimized with stochastic gradient descent. The t-SNE algorithm [27] extends the conditional similarity to a symmetric version by adding the conditional similarity of both directions. Furthermore, it uses a Student-t distribution instead of a Gaussian distribution in the embedding.

III. Methods

A. Motivation

Image data from different sources can come from different distributions, even when the same data collection process is

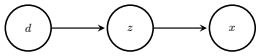


Fig. 1. Data generation process from domains

meticulously followed [28]. This phenomenon can be modeled by a directed probabilistic graphical model [29] shown in Figure 1, where the source or domain label d generates the latent representation z, which further generates the observed image x. Given an observation x, we infer the latent representation z and the domain label d as described below and given in Algorithms 1 and 2. The proposed resampling method can be regarded as a worst case analysis aiming to identify the largest possible distribution shifts that can occur when a single dataset is split into multiple folds.

B. VGMM-VAE-CV resampling scheme

In the deep learning field a given dataset $\mathcal{D}$ is typically split into disjoint subsets as $\mathcal{D} = D_{train} \bigcup D_{val} \bigcup D_{test}$, where the training dataset D_{train} is used for model training, the validation dataset D_{val} aids with model selection (e.g., along the epochs), and the test dataset D_{test} is used to evaluate the performance of the final model. Many deep learning papers benchmark the performance of different models relying on the train-test split provided along with the dataset.

Another popular approach is k-fold cross validation, which splits the data randomly into k disjoint subsets (folds or splits) of equal size, and which therefore should result in subsets with the same distribution. Similarly to k-fold cross validation in this work we split the data into k subsets. However, we aim to split the dataset such that each subset follows a different conditional distribution $p(x|y)$ of the features x given a label y, as discussed in the following.

Since high dimensional clustering is challenging, we first train a representation of the dataset using a VAE (see Section II). Instead of the observed distribution of x, for clustering we use the distribution of the latent space representation z of x, denoted by $q_\phi(z|x)$. We apply a VGMM (see Section II) on the latent representation space to assign each instance to a cluster. When we cluster within subsets defined by each class label y separately, our procedure corresponds to the conditional distribution shift, which refers to a change in $p(x|y)$, while $p(y)$ remains shared among the clusters.

Among the resulting clusters, one is used for testing and the remaining clusters are used for training and validation. We use random splitting to form the training and validation sets, which therefore share the same distribution. Analogously to conventional cross validation, the train-test process is repeated with each cluster playing the role of the test dataset once. The combination of assignments to $(D_{train} \bigcup D_{val}) \bigcup D_{test}$ yields a variation estimate on how good a model could generalize to D_{test}, similar to cross validation. However, compared to conventional cross validation, our method provides a way to characterize the deep learning model's ability to generalize under distribution shift.

Algorithm 1 VGMM-VAE-CV

Input: dataset $\mathcal{D}$, number of classes C, number of folds K
for $c = 1$ **to** C **do**
  $z^c = vae(D^c)$ # train VAE on data belonging to class c
  $d_1^c, ..., d_K^c = vgmm(z^c)$
end for
$D_1, ..., D_K = merge(\{d_k^c\}, repeat = 1)$
for $k = 1$ **to** K **do**
  $D_{test} = D_k$
  $TrainValSet = D \setminus D_k$
  $D_{train}, D_{val} = randomSplit(TrainValSet)$
  $m_k = model\_init()$
  for $epoch = 1, 2, \ldots$ **do**
    $PerfTrain_k, m_k = train(D_{train}, m_k)$
    $PerfVal_k = test(D_{val}, m_k)$
  end for
  $PerfTest_k = test(D_{test}, m_k)$
end for

Algorithm 2 merge

Input: $repeat \in \{1, 2, \ldots\}$, d_k^c for $c = 1$ **to** C and $k = 1$ **to** K
for $i = 1$ **to** $repeat$ **do**
  if $repeat > 1$ **then**
    for $c = 1$ **to** C **do**
      $d_1^c, ..., d_K^c$ = Shuffle($d_1^c, ..., d_K^c$)
    end for
  end if
  for $k = 1$ **to** K **do**
    $D_k = \bigcup_{c=1}^{C} d_k^c$
  end for
  yield $D_1, D_2, ..., D_K$
end for

The whole process is summarized in Algorithm 1, along with Algorithm 2. Algorithm 2 has an input argument $repeat$, which we set to 1 in Algorithm 1 for simplicity. We also only use $repeat = 1$ in the experiments in this work due to the limited computational resources available to us. In larger scale benchmarks the modification $repeat = m > 1$ in our method would be analogous to how simple k-fold cross validation compares to m-times repeated k-fold cross validation.

C. Architecture of the VAE

Following [30], given a single-channel input image of size 28×28 (see Sec. V), as the first layer the encoder uses convolution filters of size 4×4 with stride 2 and 64 output channels followed by a leaky ReLU activation function. The resulting activation maps go into another convolutional layer with a 128 channel output using the same filter and stride as well as batch normalization and leaky ReLU. Subsequently the results are fed into a fully connected 1024-dimensional layer followed by batch normalization and leaky ReLU. The latent variational parameter vector is chosen to be 62-dimensional and connects linearly with the layer before. After z is sampled,

the decoder maps it to a fully connected layer with output dimension 1024 followed by a batch normalization layer and a ReLU activation function. It is followed by another fully connected layer with output dimensions $7 \times 7 \times 128$, also using batch normalization and ReLU. Then a deconvolution (or transpose convolution) operation is applied resulting in a $14 \times 14 \times 64$ dimensional output, again followed by batch normalization and ReLU. Finally, another deconvolution produces a $28 \times 28 \times 1$ output that is followed by sigmoid activation function.

D. Bayesian Neural Network

Suppose that the predictive function of a classification neural network is solely determined by its weight vector w. If based on training data X and Y we got a posterior distribution over the weights $p(w \mid X, Y) = \frac{p(Y|X,w)p(w)}{\int_{w'} p(Y|X,w')p(w')dw'}$, then for a new input x^* we have that

$$p(y^* \mid x^*; X, Y) = \int_w p(y \mid x^*, w)p(w \mid X, Y)dw. \quad (1)$$

To deal with the intractable posterior distribution $p(w \mid X, Y)$, a variational posterior $q_\phi(w)$, characterized by a parameter vector ϕ, is used to approximate it. We have that

$$\begin{aligned} \log p(y|x) &= E_{q(w)} \log p(y \mid w, x) - D_{KL}(q_\phi(w)||p(w)) \\ &\quad + D_{KL}(q_\phi(w)||p(w \mid x, y)) \\ &= ELBO(\phi) + D_{KL}(q_\phi(w)||p(w \mid x, y)), \end{aligned} \quad (2)$$

where we define

$$ELBO(\phi) = E_{q(w)} \log p(y \mid w, x) - D_{KL}(q_\phi(w)||p(w)). \quad (3)$$

Optimization of $ELBO(\phi)$ with respect to ϕ is equivalent to optimizing the KL divergence between the variational posterior $q_\phi(w)$ and the posterior $p(w \mid x, y)$.

The variational parameter ϕ characterizes a Gaussian distribution on the weights by $q_\phi(w_{ij}) = \mathcal{N}(\mu_{ij}, \sigma_{ij}^2)$, where i is the input index and j is the output index. With more complicated index conventions the following analysis can be generalized to convolution operations as well.

1) Local reparameterization: Suppose that the weights W connecting two layers follow an isotropic Gaussian distribution, i.e., $q(w_{ij}) = \mathcal{N}(\mu_{ij}, \sigma_{ij}^2)$ as mentioned above. Given an input A the pre-activations $B = AW$ have elements $b_{mj} = \sum_i a_{mi} w_{ij}$, where b_{mj} is the jth output for the mth instance in the minibatch. Each b_{mj} follows a Gaussian distribution with mean and variance given by

$$E(b_{mj}) = \sum_i a_{mi} \mu_{ij}, \quad (4)$$

$$var(b_{mj}) = \sum_i a_{mi}^2 \sigma_{ij}^2, \quad (5)$$

So instead of sampling noise variables to reparameterize the weights, a computational graph could directly connect variational parameter of the weight to pre-activations, and sample pre-activations by $b_{mj} = \sum_i a_{mi}\mu_{ij} + \zeta_m \sqrt{\sum_i a_{mi}^2 \sigma_{ij}^2}$, where $\zeta_m \sim \mathcal{N}(0, 1)$. As pointed out in [17] the minibatch variance of the ELBO estimator depends on the covariance of the ELBO estimator across instances in a minibatch. Local reparameterization on the pre-activations, separately for each instance, makes the covariance zero, which could reduce the variance of the ELBO estimator, compared to global reparamerization on the weights. Further arguments on how local reparameterization reduces the variance on the gradients of ELBO with respect to σ_{ij} are presented in [17].

2) Connection with dropout and variational dropout: With dropout the pre-activations are given by $B = (A \odot \frac{D^p}{1-p})W$, where $\odot$ represent element wise product, D^p is the corresponding dropout variable, and p is the dropout rate of input. Suppose that d_{mi}^p is the dropout variable for the mth instance, corresponding to input position i, then $b_{mj} = \sum_i a_{mi} \frac{d_{mi}^p}{1-p} w_{ij}$. If $d_{mi}^p \sim Bernoulli(1-p)$, then $E(b_{mj} \mid W) = \sum_i a_{mi} w_{ij}$ and $var(b_{mj} \mid W) = \sum_i a_{mi}^2 w_{ij}^2 \frac{p}{1-p}$ Thus, approximately $b_{mj} \mid W \sim \mathcal{N}(\sum_i a_{mi} w_{ij}, \sum_i a_{mi}^2 w_{ij}^2 \frac{p}{1-p})$, which is equivalent to a Gaussian dropout with dropout noise $\mathcal{N}(1, \alpha)$ and $\alpha = \frac{p}{1-p}$. Comparing with Equation (4) and (5), this is equivalent to parametrizing the variational distribution of the weights as $q(w_{ij}) = \mathcal{N}(\mu_{ij}, \alpha\mu_{ij}^2)$, where σ_{ij} is replaced with $\alpha\mu_{ij}^2$. To make it fully consistent with Gaussian Dropout, the prior $p(w)$ in Equation (3) has to be an improper log-uniform prior so that the second term in Equation(3) does not depend on μ_{ij}. However, in this paper, we use a Gaussian proper prior to make it a more general Bayesian neural network. For the calculation of KL divergence instead of using an approximation formula for the improper prior, we simply use the variational Monte Carlo samples as in [14] to calculate the KL divergence as an expectation problem of the log of the likelihood ratio. In terms of architecture, we replace the ReLU to be the softplus activation function in the original CNN architecture.

IV. Related Work

Transfer Learning, Domain Adaptation, Domain Generalization. As mentioned in Section I, many domain adaptation and domain generalization benchmark datasets are curated by human experts or are combinations of multiple datasets. Our proposed technique, however, offers the possibility to create alternative benchmark datasets based on only one dataset.

Robustness and Generalization of Neural Networks. Much attention has recently been paid to the robustness of neural networks. For example, Adversarial examples, which are created by distorting clean images only slightly, have been shown to confuse classifiers. Adversarial robustness measures the worst case performance, while corruption robustness measures the classifier's average performance on image corruptions, and perturbation robustness measures the prediction stability and consistency under perturbations [31]. Benchmark datasets have been created [31] for robustness of neural networks under corruptions and perturbations. One way to address robustness to distribution shift, where models may silently fail on out-of-distribution samples, is to predict whether samples are out-of-distribution at test time [32]. In [33], it is reported that an out-of-distribution dataset was assigned higher confidence,

when training flow based generative models. In the above works, however, distribution shift arises from either changing data or the use of multiple datasets, while the proposed resampling technique uses only one dataset to deliberately generate distribution shifted splits, and the data itself are left intact.

To quantify generalization, measures such as the margin distribution as a predictor for the generalization gap is studied [34]. It would be interesting to evaluate similar measures on data splits created by our resampling technique.

Disentanglement. Disentanglement tries to find a latent representation of data that aligns with independent data generalization factors for interpretation and robust classification [35]. Our method does not aim at achieving disentanglement, i.e., the inferred subdomain labels do not necessarily correspond to any independent data generalization factors. However, our method could potentially testify if disentangled representations can help to overcome model performance deterioration caused by distribution shift.

V. Experiments

We use the Fashion-MNIST [36] data for the initial examples. Fashion-MNIST consists of $70,000$ grayscale fashion product images of size 28×28 pixels, which fall into 10 classes (7000 images per class). The original $60,000$ training and $10,000$ test images are combined before the application of cross validation and our resampling method discussed below. Source code and further datasets can be found at https://github.com/compstat-lmu/paper_2019_variationalResampleDistributionShift.

We compare our resampling method with 5-fold cross validation to demonstrate empirically that distribution shift is indeed a problem, and we investigate how different CNN models are affected by distribution shift.

A. Data Splitting with Distribution Shift

We purposefully obtain data splits with distribution shift using the transformed latent representation from the trained VAE model by methods described in Section III (see Algorithm 1). In order to visualize the distribution shift we apply the t-SNE algorithm to the total data from all classes latent representation by training a separate VAE, then color data from each cluster from our method to represent the split, as shown in Figure 2. As a quantitative assessment we calculate the Wasserstein distances shown in Section V-E, from which it is apparent that distances between the splits resulting from our resampling technique are much larger compared to random splitting.

B. Assessment of Performance Deterioration of CNN Models due to Distribution Shift

For the first experiment we use the well known AlexNet [37] and LeNet [38] CNN architectures to perform an image classification task on the Fashion-MNIST data as described in Section III. We also use a simple neural network with three convolutional and three fully connected layers, denoted by 3conv3fc.

The goal of this experiment is two-fold. (1) We show that we can indeed deliberately subsample a given dataset to create several subsets, which are affected by distribution shift with respect to $p(x|y)$ but roughly share a common distribution $p(y)$ among the clusters (we confirmed this post-hoc empirically). (2) We demonstrate that distribution shift between the training and the test data substantially reduces the classification accuracy of CNN models on the test data, and it furthermore largely increases the variability in the reported accuracy values.

Both the conventional 5-fold cross validation and the approach of Algorithm 1 yield five (train-validation)-test configurations each, where validation takes 20 percent of the train-validation splits randomly. Thus, each considered CNN is trained and tested five times using conventional cross-validation for data splitting, and five times using our proposed approach. All models are trained for 100 epochs, and we record the training, validation, and test accuracies. Figures 3a, 4a, and 5a show line plots of accuracy by epoch for AlexNet, LeNet, and 3conv3fc respectively, which are separated into individual panels according to the data split (training, validation, test) and data splitting procedure (conventional cross-validation and Algorithm 1). In particular, the first row of the panels in each figure shows the results from using conventional cross validation for data splitting (i.e., no distribution shift), where we see that the test accuracy is almost identical to the validation accuracy (as one would expect). The second row of the panels in each figure, however, shows that the test accuracy curves behave wildly different than the validation accuracy curves. Specifically, the test accuracy is on average substantially reduced when the data are split according to Algorithm 1, i.e., when there is a shift in the conditional feature distribution $p(x|y)$ between the training and the test splits. Furthermore, we see that distribution shift in $p(x|y)$ also leads to a large increase in the variance of the obtained test accuracy values. These results are also summarized in Table I.

Thus, the comparison between conventional cross validation results (where all training and test distributions are equal) and our resampling approach for data splitting clearly shows that a shift in the conditional feature distributions $p(x|y)$ can lead to a massive deterioration in test data performance, even when the label distributions $p(y)$ are equal.

C. Bayesian CNNs under Distribution Shift

The Bayesian approach to deep learning uses distributions over parameters instead of point estimates to represent the model. This makes Bayesian deep neural networks more robust to overfitting [16], and suggests that they may be less affected by distribution shift. In our second experiment we investigate whether Bayesian CNNs are more robust to distribution shift introduced by our proposed resampling strategy, compared to the conventional CNNs considered in Section V-B.

We use the Bayesian counterparts of the same CNN architectures as considered in Section V-B, such as the Bayesian versions of AlexNet [37] and LeNet [38] introduced by [16]. Apart from the substitution of the Bayesian CNN models in place of the frequentist CNN architectures, the experiments

are identical to those described in Section V-B. Figures 3b, 4b, and 5b as well as Table I show the results in the same format as described in Section V-B. While it is apparent from Figures 3b and 4b that Bayesian CNNs are less prone to overfitting to the training data than their conventional CNN counterparts, their vulnerability with respect to distribution shift seems to be about the same.

Although the Bayesian Neural Network is trained with respect to the variational free energy objective, which shows better generalization to data from the same distribution compared to the frequentist approach [14], the gradients with respect to the variational parameters are still only based on the training data distribution. In future work, it would also be interesting to investigate if the expressive power of the variational distribution on the weights would be a potential factor to improve.

D. Comparison of CNN models with respect to their robustness to distribution shift

Because under our proposed resampling approach the validation and the training data share the same distribution but the conditional feature distribution $p(x|y)$ of the test data is shifted, the robustness of a CNN to distribution shift can be quantified by comparing the test accuracy curves to the validation accuracy curves in our experiments (see Figures 3, 4, and 5). There are different approaches to carry out such a comparison. However, for simplicity in this work we compare only the empirical mean and standard deviation values at the last epoch. Table I summarizes these values. We see that the classification accuracy on the test data reduces by about 26.0 points on average due to distribution shift. In addition, in the presence of distribution shift the standard deviation of the reported test accuracy values is about 14 times larger than the standard deviation of the accuracy values on the validation data.

While the degree of performance deterioration as measured by these analyses seems to be about the same between all considered CNN models, it is conceivable that some models will be more or less affected by distribution shift, which will be reflected in the values and accuracy curves as analyzed above. Hence, our framework provides a way to quantitatively compare the robustness to distribution shift between different models.

As an additional point of reference, Table II contains the classification accuracies after 100 training epochs for the same CNN architectures on the original train-test split provided in the Fashion-MNIST [36] data. Note that there is no distribution shift between the training and the testing data in this case, and the training dataset is larger than in the experiments of Sections V-B and V-C.

E. Computed pairwise Wasserstein distances

With the python package `POT` [24] one can compute the pairwise Wasserstein distance between two clusters of data. In Table III we computed the pairwise Wasserstein distances across the 5 clusters created based on VGMM as described in Section III, which correspond to a conditional distribution shift in $p(x|y)$. In Table IV the pairwise Wassserstein distances

Model	Train. acc.	Val. acc.	Test acc
AlexNet	98.97 (0.13)	90.93 (0.43)	66.51 (4.88)
Bayesian AlexNet	96.33 (0.29)	91.58 (0.25)	64.21 (5.67)
LeNet	98.03 (0.25)	91.44 (0.29)	65.43 (5.00)
Bayesian LeNet	94.04 (0.27)	90.54 (0.82)	63.18 (4.72)
3conv3fc	97.97 (0.13)	91.91 (0.28)	67.92 (5.18)
Bayesian 3conv3fc	98.69 (0.19)	91.26 (0.43)	64.44 (4.19)

TABLE I
Average classification accuracies on the training, validation, and testing data splits after 100 training epochs for several CNN models. Empirical mean values with standard deviation in parentheses are computed across the five data splits which are obtained by Algorithm 1.

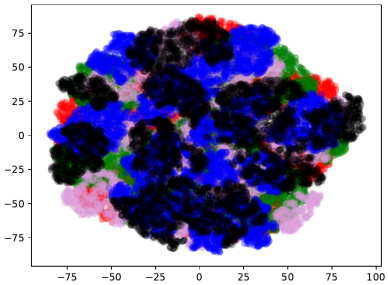


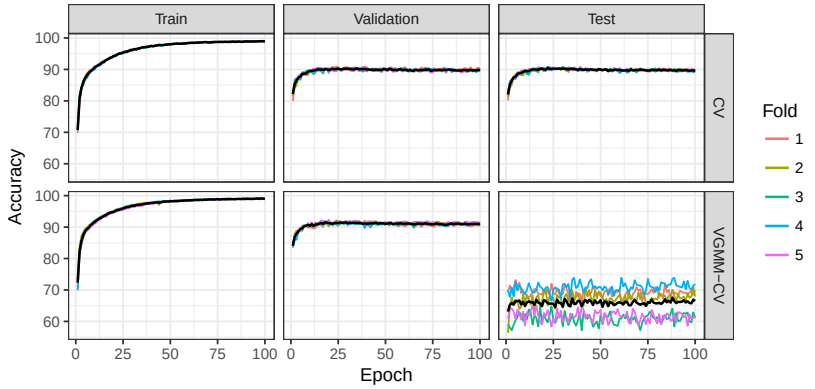
Fig. 2. Scatter plot of t-SNE transformed 2-d values from joint data latent representations. Colors indicate different clusters from VGMM-VAE-CV.

are computed based on random splits as in conventional cross validation. It can be clearly seen that the VGMM variant creates larger pairwise Wasserstein distances, which testifies that our proposed method generates splits of data with significant distribution shift, as intended.

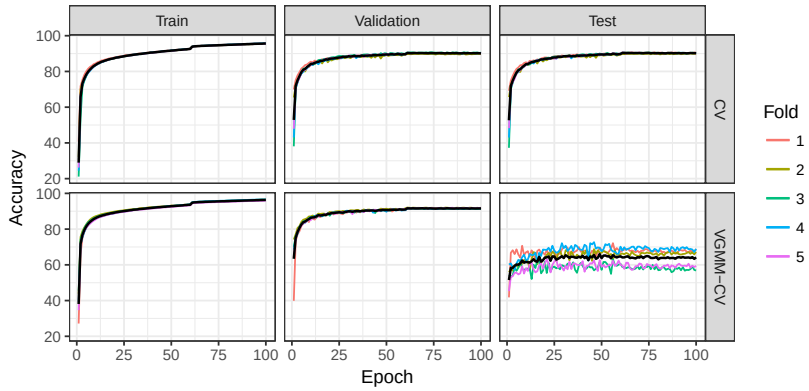
VI. Summary and Conclusion

We propose a new resampling technique to create pseudo subdomains over one dataset. Our resampling strategy purposefully identifies data splits with distribution shift with respect to the conditional distribution $p(x|y)$ of features x given the label y by utilizing the latent representation of data through generative models. Variational methods are used to assign instances to the pseudo subdomains, which are represented as clusters in the latent space.

We use our new resampling technique to assess the robustness of deep neural networks in terms of generalization ability to distribution shift. We show that CNN models display



(a) AlexNet



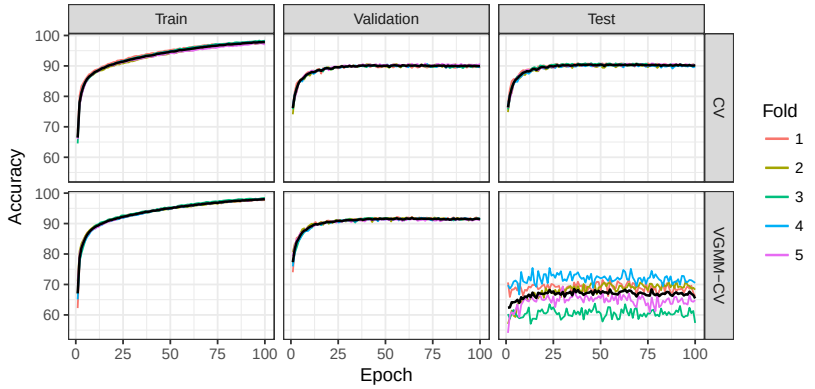
(b) Bayesian AlexNet

Fig. 3. (a) AlexNet [37] and (b) Bayesian AlexNet [16] classification accuracies on the Fashion-MNIST data are shown by epoch (1-100), data split (training, validation, test), and data splitting procedure ("CV" and "VGMM-CV"). In the first row of panels, entitled "CV", the data are split randomly (conventional cross validation). In the second row of panels, entitled "VGMM-CV", the data are split as in Algorithm 1 leading to a conditional shift in $p(x|y)$ between the training and the test splits. The thicker black line represents the average value across the data splits. We see that a shift in the conditional feature distribution $p(x|y)$ of the test data leads to a reduced accuracy as well as an increased variance.

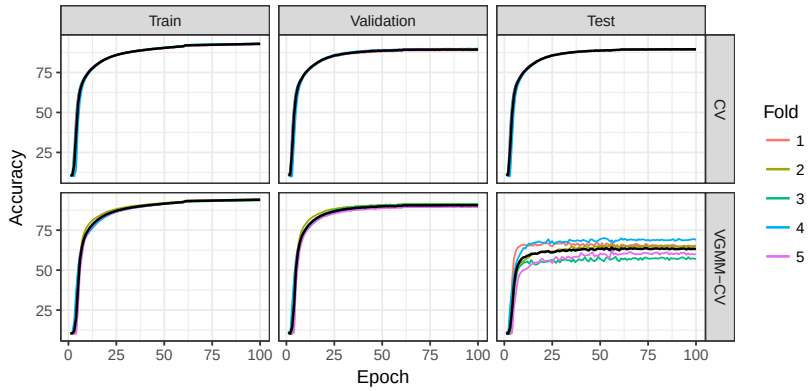
substantial reductions in performance and an increase in variability under the proposed resampling technique compared to conventional cross validation. This demonstrates the severe problem that the performance of CNN models is strongly affected by changes in the conditional distribution $p(x|y)$ even when the label distribution $p(y)$ remains unchanged and all data originate from the same domain. In addition, we observe that this problem persists for Bayesian CNNs considered in this work, even though Bayesian CNNs otherwise are known to possess superior generalization properties at least for data from the same distribution. Possibly since the gradients with respect to the variational parameters are also based on data from the training distribution, it makes it difficult to generalize to another distribution.

Our approach can be used for the evaluation of the generalization ability of deep learning models and inform model selection, alongside conventional performance evaluation approaches such as cross validation and testing on holdout data. For instance, Automatic Machine Learning (AutoML) [39] methods should also take distribution shift into account when searching for a model, where our method could easily create splits to serve within an objective function to be optimized during the AutoML process.

There remain some open questions and potential drawbacks



(a) LeNet



(b) Bayesian LeNet

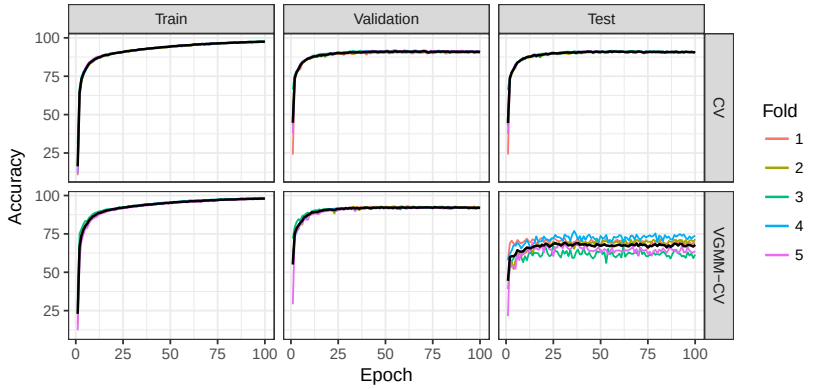
Fig. 4. (a) LeNet [38] and (b) Bayesian LeNet [16] classification accuracies on the Fashion-MNIST data are shown by epoch (1-100), data split (training, validation, test), and data splitting procedure ("CV" and "VGMM-CV"). In the first row of panels, entitled "CV", the data are split randomly (conventional cross validation). In the second row of panels, entitled "VGMM-CV", the data are split as in Algorithm 1 leading to a conditional shift in $p(x|y)$ between the training and the test splits. The thicker black line represents the average value across the data splits. We see that a shift in the conditional feature distribution $p(x|y)$ of the test data leads to a reduced accuracy as well as an increased variance.

of our method. It is yet unknown what model architecture or choice of hyperparameters will affect the created subdomains. Additionally, our artificially created pseudo subdomains do not necessarily correspond to real world (sub)domains, and it is not clear to what extent the artificially created distribution shift is comparable to the types of distribution shift observed between different (sub)domains in the real world.

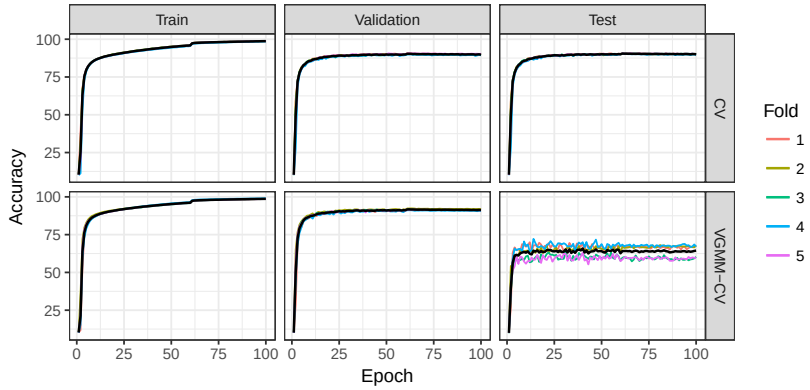
In future work, methods similar to Restrictive Federated Model Selection [5] could be used to adapt to the distribution shift generated by the methods proposed in this work. In addition, it would be interesting to see new methods that not only create distribution shift, but also allow to control the extent of distribution shift by use of appropriate hyperparameters, as well as methods which could create pseudo subdomains on tasks other than classification, such as recommendation systems [40]. Furthermore, it would be interesting to see how our approach would serve as a benchmark method to evaluate different domain adaptation and domain generalization algorithms.

References

[1] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 39–57.



(a) 3conv3fc



(b) Bayesian 3conv3fc

Fig. 5. (a) A simple neural network with three convolution and three fully connected layers (3conv3fc) and (b) Bayesian 3conv3fc classification accuracies on the Fashion-MNIST data are shown by epoch (1-100), data split (training, validation, test), and data splitting procedure ("CV" and "VGMM-CV"). In the first row of panels, entitled "CV", the data are split randomly (conventional cross validation). In the second row of panels, entitled "VGMM-CV", the data are split as in Algorithm 1 leading to a conditional shift in $p(x|y)$ between the training and the test splits. The thicker black line represents the average value across the data splits. We see that a shift in the conditional feature distribution $p(x|y)$ of the test data leads to a reduced accuracy as well as an increased variance.

[2] I. Chen, F. D. Johansson, and D. Sontag, "Why Is My Classifier Discriminatory?" in *Advances in Neural Information Processing Systems 31*, 2018, pp. 3539–3550.

[3] M. A. Badgeley, J. R. Zech, L. Oakden-Rayner, B. S. Glicksberg, M. Liu, W. Gale, M. V. McConnell, B. Percha, T. M. Snyder, and J. T. Dudley, "Deep learning predicts hip fracture using confounding patient and healthcare variables," *npj Digital Medicine*, vol. 2, no. 1, p. 31, Apr. 2019.

[4] J. Wen, C.-N. Yu, and R. Greiner, "Robust learning under uncertain test distributions: Relating covariate shift to model misspecification." in *ICML*, 2014, pp. 631–639.

[5] X. Sun, A. Bommert, F. Pfisterer, J. Rahnenführer, M. Lang, and B. Bischl, "High dimensional restrictive federated model selection with multi-objective bayesian optimization over shifted distributions," *arXiv preprint arXiv:1902.08999*, 2019.

[6] B. Nestor, M. B. A. McDermott, G. Chauhan, T. Naumann, M. C. Hughes, A. Goldenberg, and M. Ghassemi, "Rethinking clinical prediction: Why machine learning must consider year of care and feature aggregation," *arXiv preprint arXiv:1811.12583*, Nov. 2018.

[7] K. Weiss, T. M. Khoshgoftaar, and D. Wang, "A survey of transfer learning," *Journal of Big data*, vol. 3, no. 1, p. 9, 2016.

[8] K. Akuzawa, Y. Iwasawa, and Y. Matsuo, "Adversarial invariant feature learning with accuracy constraint for domain generalization," *arXiv preprint arXiv:1904.12543*, 2019.

[9] M. Long, H. Zhu, J. Wang, and M. I. Jordan, "Deep transfer learning with joint adaptation networks," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 2208–2217.

[10] I. Guyon, A. Saffari, G. Dror, and G. Cawley, "Model selection: Beyond the bayesian/frequentist divide," *Journal of Machine Learning Research*,

MODEL	TRAIN. ACC.	TEST ACC
ALEXNET	98.78	89.42
BAYESIAN ALEXNET	95.86	90.56
LENET	97.71	89.77
BAYESIAN LENET	93.38	89.06
3CONV3FC	97.37	91.08
BAYESIAN 3CONV3FC	98.74	90.58

TABLE II
CLASSIFICATION ACCURACY AFTER 100 TRAINING EPOCHS OF MULTIPLE CNN MODELS ON THE ORIGINAL TRAIN-TEST SPLIT PROVIDED IN THE FASHION-MNIST [36] DATA.

TABLE III
PAIRWISE WASSERSTEIN DISTANCE ACROSS 5 CLUSTERS CREATED BY VARITIONAL GAUSSIAN MIXTURE MODELS ON THE DATA LATENT SPACE

$$\begin{bmatrix} 0.0 & 0.22072112 & 0.21187810 & 0.22447902 & 0.21012454 \\ 0.22068973 & 0.0 & 0.22250834 & 0.21232671 & 0.20131575 \\ 0.21465105 & 0.22328222 & 0.0 & 0.23423279 & 0.21377970 \\ 0.23307045 & 0.20407709 & 0.23252131 & 0.0 & 0.20784507 \\ 0.20973221 & 0.20124379 & 0.21279558 & 0.20822021 & 0.0 \end{bmatrix}$$

TABLE IV
PAIRWISE WASSERSTEIN DISTANCE ACROSS 5 CLUSTERS CREATED BY RANDOM SPLITTING OF THE DATA

$$\begin{bmatrix} 0.0 & 0.13235691 & 0.14389180 & 0.14445968 & 0.13845333 \\ 0.13221454 & 0.0 & 0.13702394 & 0.13594290 & 0.13507776 \\ 0.14411578 & 0.13698794 & 0.0 & 0.14563888 & 0.14467288 \\ 0.14428186 & 0.13616575 & 0.14567640 & 0.0 & 0.13870558 \\ 0.13836885 & 0.13548402 & 0.14428162 & 0.13905402 & 0.0 \end{bmatrix}$$

vol. 11, no. Jan, pp. 61–87, 2010.

[11] A. Bommert, X. Sun, B. Bischl, J. Rahnenführer, and M. Lang, "Benchmark for filter methods for feature selection in high-dimensional classification data," *Computational Statistics & Data Analysis*, p. 106839, 2019.

[12] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[13] Z. Jiulong, G. Luming, Y. Su, S. Xudong, and L. Xiaoshan, "Detecting chinese calligraphy style consistency by deep learning and one-class svm," in *2017 2nd International Conference on Image, Vision and Computing (ICIVC)*. IEEE, 2017, pp. 83–86.

[14] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight uncertainty in neural networks," *arXiv preprint arXiv:1505.05424*, 2015.

[15] X. Sun and B. Bischl, "Tutorial and survey on probabilistic graphical model and variational inference in deep reinforcement learning," 2019.

[16] K. Shridhar, F. Laumann, and M. Liwicki, "A comprehensive guide to bayesian convolutional neural network with variational inference," *arXiv preprint arXiv:1901.02731*, 2019.

[17] D. P. Kingma, T. Salimans, and M. Welling, "Variational Dropout and the Local Reparameterization Trick," in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 2575–2583.

[18] D. Molchanov, A. Ashukha, and D. Vetrov, "Variational dropout sparsifies deep neural networks," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 2498–2507.

[19] Y. Gal and Z. Ghahramani, "Bayesian convolutional neural networks with bernoulli approximate variational inference," *arXiv preprint arXiv:1506.02158*, 2015.

[20] N. Nasios and A. G. Bors, "Variational learning for gaussian mixture models," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 36, no. 4, pp. 849–862, Aug 2006.

[21] K. Zhang, B. Schölkopf, K. Muandet, and Z. Wang, "Domain adaptation under target and conditional shift," in *International Conference on Machine Learning*, 2013, pp. 819–827.

[22] G. Peyr, M. Cuturi, and J. Solomon, "Gromov-wasserstein averaging of kernel and distance matrices," in *Proceedings of The 33rd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. F. Balcan and K. Q. Weinberger, Eds., vol. 48. New York, New York, USA: PMLR, 20–22 Jun 2016, pp. 2664–2672. [Online]. Available: http://proceedings.mlr.press/v48/peyre16.html

[23] F. Mémoli, "Gromov–wasserstein distances and the metric approach to object matching," *Foundations of computational mathematics*, vol. 11, no. 4, pp. 417–487, 2011.

[24] R. Flamary and N. Courty, "Pot python optimal transport library," 2017. [Online]. Available: https://github.com/rflamary/POT

[25] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein gan," *arXiv preprint arXiv:1701.07875*, 2017.

[26] G. E. Hinton and S. T. Roweis, "Stochastic neighbor embedding," in *Advances in neural information processing systems*, 2003, pp. 857–864.

[27] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[28] B. Recht, R. Roelofs, L. Schmidt, and V. Shankar, "Do ImageNet Classifiers Generalize to ImageNet?" Feb. 2019. [Online]. Available: http://arxiv.org/abs/1902.10811

[29] D. Koller and N. Friedman, *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[30] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel, "Infogan: Interpretable representation learning by information maximizing generative adversarial nets," in *Advances in neural information processing systems*, 2016, pp. 2172–2180.

[31] D. Hendrycks and T. Dietterich, "Benchmarking neural network robustness to common corruptions and perturbations," *arXiv preprint arXiv:1903.12261*, 2019.

[32] S. Liang, Y. Li, and R. Srikant, "Enhancing the reliability of out-of-distribution image detection in neural networks," *arXiv preprint arXiv:1706.02690*, 2017.

[33] E. Nalisnick, A. Matsukawa, Y. W. Teh, D. Gorur, and B. Lakshminarayanan, "Do deep generative models know what they don't know?" *arXiv preprint arXiv:1810.09136*, 2018.

[34] Y. Jiang, D. Krishnan, H. Mobahi, and S. Bengio, "A margin-based measure of generalization for deep networks," 2019. [Online]. Available: https://openreview.net/pdf?id=HJlQfnCqKX

[35] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner, "beta-vae: Learning basic visual concepts with a constrained variational framework." *ICLR*, vol. 2, no. 5, p. 6, 2017.

[36] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[37] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105.

[38] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," pp. 2278–2324, 1998.

[39] X. Sun, J. Lin, and B. Bischl, "Reinbo: Machine learning pipeline search and configuration with bayesian optimization embedded reinforcement learning," *arXiv preprint arXiv:1904.05381*, 2019.

[40] N. Kushwaha, X. Sun, B. Singh, and O. Vyas, "A lesson learned from pmf based approach for semantic recommender system," *Journal of Intelligent Information Systems*, vol. 50, no. 3, pp. 441–453, 2018.

Appendix B

High Dimensional Restrictive Federated Model Selection with multi-objective Bayesian Optimization over shifted distributions

Contributing Article Sun, Xudong; Bommert, Andrea; Pfisterer, Florian; Rahnenfuehrer, Joerg; Lang, Michel; Bischl, Bernd; High Dimensional Restrictive Federated Model Selection with multi-objective Bayesian Optimization over shifted distributions, Intelligent Systems and Applications, 629–647, 2019, Springer International Publishing, Cham.

Author Contributions Xudong Sun proposed the idea of Restrictive Federated Model Selection (RFMS) based on earlier research ideas of Michel Lang, Joerg Rahnenfuehrer and Bernd Bischl of multi-objective tuning of cohort. Michel Lang and Bernd Bischl suggested the evaluation strategy for openbox and curator datasite. The implementation of RFMS is done by Xudong Sun, together with the benchmark codes. He conducted the experiments and collected the results. Andrea Bommert wrote some of the critical analytical code for result analysis with refinements by Xudong Sun. Florian Pfisterer wrote a first version of code for the clustering of data to simulate distribution shift with refinement by Xudong Sun. Xudong Sun wrote the manuscript, with refinements from all authors, especially Andrea Bommert.

High Dimensional Restrictive Federated Model Selection with Multi-objective Bayesian Optimization over Shifted Distributions

Xudong Sun[1(✉)], Andrea Bommert[2(✉)], Florian Pfisterer[1(✉)], Jörg Rähenfürher[2(✉)], Michel Lang[2(✉)], and Bernd Bischl[1(✉)]

[1] LMU Munich, Munich, Germany
smilesun.east@gmail.com,
{florian.pfisterer,bernd.bischl}@stat.uni-muenchen.de
[2] TU Dortmund, Dortmund, Germany
{lang,bommert,rahnenfuehrer}@statistik.tu-dortmund.de

Abstract. A novel machine learning optimization process coined Restrictive Federated Model Selection (RFMS) is proposed under the scenario, for example, when data from healthcare units can not leave the site it is situated on and it is forbidden to carry out training algorithms on remote data sites due to either technical or privacy and trust concerns. To carry out a clinical research in this scenario, an analyst could train a machine learning model only on local data site, but it is still possible to execute a statistical query at a certain cost in the form of sending a machine learning model to some of the remote data sites and get the performance measures as feedback, maybe due to prediction being usually much cheaper. Compared to federated learning, which is optimizing the model parameters directly by carrying out training across all data sites, RFMS trains model parameters only on one local data site but optimizes hyper parameters across other data sites jointly since hyper-parameters play an important role in machine learning performance. The aim is to get a Pareto optimal model with respective to both local and remote unseen prediction losses, which could generalize well across data sites. In this work, we specifically consider high dimensional data with different distributions over data sites. As an initial investigation, Bayesian Optimization especially multi-objective Bayesian Optimization is used to guide an adaptive hyper-parameter optimization process to select models under the RFMS scenario. Empirical results shows that solely using the local data site to tune hyper-parameters generalizes poorly across data sites, compared to methods that utilize the local and remote performances. Furthermore, in terms of hypervolumes, multi-objective Bayesian Optimization algorithms show increased performance across multiple data sites among other candidates.

Keywords: Federated learning · Multi-objective Bayesian Optimization · High dimensional data · Differential privacy · Distribution shift · Model selection

Y. Bi et al. (Eds.): IntelliSys 2019, AISC 1037, pp. 629–647, 2020.
https://doi.org/10.1007/978-3-030-29516-5_48

1 Introduction

1.1 Background

Federated Learning [20,24] has drawn increasing attention recently due to overwhelmingly growing data volume and an emerging request for privacy protection from the perspective of individuals, as well as the perspective of data owners, e.g. due to GDPR [26]. Usually in federated learning, a server moderates several data sites to carry out optimization iterations, like gradient descent updates, on each data site. Each data site then sends an intermediate result to the server. The server side aggregates the results and distributes it, so that each data site obtains an updated model. This distributed model training process circumvents the bottleneck of data transmission and prevents private data from leaving the data center. To further increase privacy security against attacks [26], differential private federated learning algorithms have been proposed [19,37].

Current federated learning algorithms rely on an efficient and synchronized communication protocol [20,25] across the server and different data sites as well as the availability that data on each data site can be used for training. However, it might also be expensive to meet the technical requirements to have a synchronized communication framework needed by federated learning.

From a privacy protection perspective, several attacks and defenses that undermine privacy in a federated learning context have been proposed [3,27,32]. Differential private federated learning algorithms [12,26,37] are based on standard Federated Learning algorithms, with some detail being tailored to fit the need for differential privacy.

However, there might be restrictions that the data from the remote data site can not be used for training at all. Especially when there is no established trust between parties, privacy protection and attack becomes an arm race, in which case, data owners might want to restrict the access of the data to a maximum extent but still want to participate in the community to build a predictive model that could benefit all sides. To the best of our knowledge, this is a problem that current differential private federated learning algorithms do not address yet.

In both restricted cases, sending a model to the remote data sites and asking for how good the sent model performs on the remote data sites comes at a certain cost (transmission cost and prediction computation cost for instance). This is comparably acceptable, as only aggregated statistics (typically a single number) need to be reported back.

We coin this new learning scenario Restrictive Federated Learning, emphasizing the point that only data situated locally could be used for model training, while data on the other data sites are partially observed in the sense that the analyst could only observe a scalar performance measure of a sent model on the remote data site, which is restrictive.

In this restrictive learning scenario, we could only access limited data locally for training a machine learning model, but still want to have a model that could generalize well across the data sites. Therefore, how to do model selection in this special restricted federated learning scenario is of significant interest.

Bayesian Optimization has proved to be really successful in optimizing machine learning hyper-parameters [34]. In this work, we want to investigate how it works under the RFMS scenario.

1.2 Challenges

A critical challenge in federated learning is unevenly distributed data. For example, there are situations where most features are not available on all data sites [19] or the class distribution is extremely unbalanced across different computation nodes [43].

In RFMS, there is also the challenge that data can be differently distributed on each data site. Specifically, in this work we consider the challenge that distribution of features from one data site might be considerably different from another, due to different sub-populations frequenting a given clinic for example.

Furthermore, the number of observations in clinical research is usually relatively small, while with the inclusion of genetic data, the number of features can be rather large. This makes model selection [6,14] quite challenging. Finding stable predictive models that could generalize well to data collected from different clinical studies or cohorts is difficult.

2 Problem Statement

2.1 Terminology and Notation

To clearly address the problem, at the first step, terminologies and notations used throughout the remainder of this paper are explained.

Data Site: Data of a specific domain, clinical research for example, could be located in different places and it is expensive to carry data from one site to another due to technical or privacy concerns. We denote one of such a integrated data unity as a data site. There is a need to train a specific machine learning model for the domain, which requires collaboration across data sites. We consider data sites of following types.

Openbox Data Site D_{ob}: On the openbox data site, the analyst has full access to the data. A machine learning model can be trained locally using the data situated on openbox data site.

Curator Data Site D_{cu}: From the openbox side, curator data site can be queried for model performance, which can assist the analyst on the openbox data site to get a better model that might generalize across data sites. The curator data site D_{cu} can only be queried with respect to predictive performance, i.e. a single aggregate statistic, but the analyst from the openbox side can not access the data in any other way. This name stems from the field of differential privacy [9] where there is a curator that controls the data flow which acts like a firewall to D_{cu}. The curator has full access to D_{cu} but decides on its strategy w.r.t. which feedback value to give to the statistical query by actively perturbing and coordinating the answers given to the queries. In this work, we assume a honest answer to the query except otherwise specified.

Lockbox Data Site D_{lb}: Lockbox [13] data site refers to data sites which the analyst from the openbox side can not access by any means. In practice, lockbox correspond to data sites that could not contribute in the process of building a machine learning model due to various reasons, but are likely to participate in the future or simply benefit from the model built. From a model evaluation perspective, D_{lb} on the other hand could measure how good a machine learning model generalizes to completely unseen data.

Inbag and Outbag: For evaluation purposes, we hold out a fraction (say 20%) of the curator and openbox data which we call **outbag**, denoted by D_{cu}^{og} and D_{ob}^{og}, the leftover is called **inbag**, which is D_{cu}^{ig} and D_{ob}^{ig}. For simplicity, we use D_{ob} to represent D_{ob}^{ig} when the context is about learning and use D_{ob} to represent D_{ob}^{og} when the context is about evaluating how good a method is. Also, we define the inbag and outbag of lockbox to be identical to lockbox itself, i.e. $D_{lb} = D_{lb}^{ig} = D_{lb}^{og}$.

Model Parameter θ and Hyper-parameter ϕ: A machine learning algorithm, given a dataset D_l, where l means "learn" or "local", $D_l = D_{ob}^{ig}$, for example, and a set of hyperparameters ϕ, learns a model specified by a set of model parameters $\theta = \mathcal{L}(D_l \mid \phi)$ where $\mathcal{L}$ represent the learning process to map a dataset D_l associated with a set of hyper-parameter ϕ to a machine learning model parameter θ.

Model Performance and Loss: The performance of a model characterized by θ to a data site D is given by

$$\mathcal{F}\left(D \mid \theta = \mathcal{L}(D_l \mid \phi)\right)$$

where $\mathcal{F}$ computes an estimate of predictive performance on D, under model parameter θ trained from dataset D_l, based on hyper-parameter ϕ. By convention, we use J to represent a regret that need to be minimized, which could be $1 - accuracy$ for example.

Restricted Federated Model Selection (RFMS) Scenario: The analyst from the openbox side want to initiate a study to a specific domain (clinical studies like cancer research for example). A machine learning model that fits the data well on the openbox side, as well as one that could generalize to a certain extent across the other data sites is required. Due to privacy sensitivity or technical difficulty, some data sites could only collaborate in a model selection process in the form of curators. Each query to the curator from the openbox side is at a certain cost. Note that all forms of data sites including openbox, curator and lockbox should be used to evaluate the selected model whenever possible.

2.2 An Example of RFMS on High Dimensional Unevenly Distributed Data

Gene Expression Omnibous (GEO) is a public available functional genomics data repository with array and sequence based data that researchers from around the world could contribute to. Although the data in GEO is publicly available instead of privacy sensitive, the origin that the datasets in GEO comes from

different sources makes it a perfect example of RFMS. We use the breast cancer datasets GSE16446, GSE20194, GSE20271, GSE32646, and GSE6861 from the GEO database [8,23]. Each dataset we consider here could be regarded as a data site due to the fact that they come from different sources, by different contributors.

The publicly available microarray gene expression datasets were accessed via tools provided by the Gene Expression Omnibus (GEO) data repository. Frozen robust multiarray analysis (fRMA) [23] was used for normalization. All breast cancer datasets were checked for duplicates and a pair of patients was considered duplicate when the correlation of their expression values was at least 0.999. Duplicates were removed. The response variable is binary (classes "pathological complete response" and "residual disease") for all datasets. The six observations with a missing value for the response variable are omitted. The resulting numbers of observations per dataset are displayed in Table 1. The datasets contain clinical and gene expression data. We do not consider the clinical variables because many values are missing. The gene expression data has been measured on three different types of microarray chips (HG-U133-Plus2 for GSE16446 and GSE32646, HG-U133-A for GSE20194 and GSE20271, and HG-U133-X3P for GSE6861). As the measured genes differ between the three chips, we only consider the genes that are measured on all of the chips. Out of these 1965 genes, we only use the 1000 genes with the highest variances across all patients and datasets.

Table 1. Number of observations per GEO dataset

GEO-ID	16446	20194	20271	32646	6861
Observations	114	211	178	115	161

It can be assumed that the relation between the response variable and the covariates is not identical across the datasets and the features distribution also varies from data site to data site. This is typical for gene expression data, especially if it has been measured on different chips, at different times, at different places and after different times until the tissue was frozen. A T-SNE [22] plot by pooling the feature part of the data together from these data sites can be found in Fig. 1 where the colors indicate different data sites. From Fig. 1, it is obvious that the data sites lie on different locations in the low dimension embedding, which is a clear indicator of distribution shift across data sites. We will use this example as a major case in this paper.

2.3 Evaluation Criteria

To further explain the problem, before discussing any potential solution, we first address the question of how to evaluate model performance, which will help deeper understanding of the problem.

634 X. Sun et al.

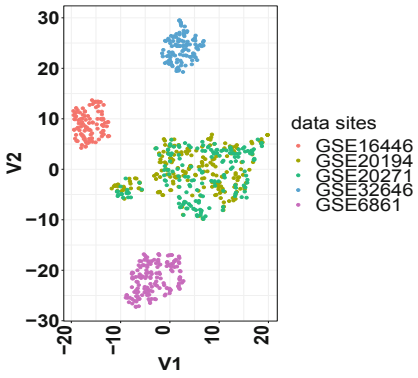


Fig. 1. T-SNE plot for the GEO datasets over data sites.

In RFMS, we want to obtain a model that generalize well for the openbox, curator and hopefully for the lockbox as well, which is a multi-objective problem. Accordingly, the selected model should also be evaluated with method that could take different objectives into consideration.

Dominated Hypervolume: A natural criterion is to measure the Dominated Hypervolume [2] of the model performance on the outbag part of openbox and curator site, as well as the lockbox, as in Eq. (1)

$$
\begin{aligned}
&J^{hv}(\phi \mid D_{ob}^{og},\ D_{cu}^{og},\ D_{lb}) = \mathcal{H}\left[f_{ob}^{og},\ f_{cu}^{og},\ f_{lb}\right], \\
&f_{ob}^{og} = \mathcal{F}\left(D_{ob}^{og} \mid \theta = \mathcal{L}(D_{ob}^{ig} \mid \phi)\right), \\
&f_{cu}^{og} = \mathcal{F}\left(D_{cu}^{og} \mid \theta = \mathcal{L}(D_{ob}^{ig} \mid \phi)\right), \\
&f_{lb} = \mathcal{F}\left(D_{lb} \mid \theta = \mathcal{L}(D_{ob}^{ig} \mid \phi)\right).
\end{aligned} \tag{1}
$$

where, $\mathcal{H}$ represent the calculation of the Dominated Hypervolume, and the performance on each data site outbag part is represented as $f_{ob}^{og},\ f_{cu}^{og},\ f_{lb}$ respectively. Dominated Hypervolume Indicator is also known as Lebesgue Measure or S-Metric which is the hypervolume between a non-dominated front and a reference point. Due to space limit, we invite readers who are not familiar with these multi-objective concepts to refer to the references.

3 Related Work

In this section, we review recent works that has connections with RFMS.

Nested Cross Validation (NCV): NCV [14] uses an outer loop cross validation to safe guard the risk of overfitting during the hyper-parameter tuning process. However, RFMS does not allow cross validation due to the constraint that remote data site can not be used for training.

Federated Learning: Federated learning [24] also consider situations where data is distributed non-i.i.d. across several data sites and possibly unbalanced, but they assume scenarios where data is fully accessible over a huge amounts of data sites compared to a smaller number of data points available at each site. This is different from RFMS, where we consider data can only be accessed through prediction. Moreover, in RFMS, we consider a relatively small amount of data sites with less instances but high dimensional data.

Distribution Shift: Distribution Shift refers to a mismatch in distribution between the data an algorithm was trained on, and data used for model validation or prediction. Detecting and characterizing such shift remains an open problem [29,42]. In this work, we do not drive deeper in theory of the data shift problem, but provides an empirical study which partially addresses the data shift problem, especially when feature distribution varies across data sites.

Train On Validation: In [36] the authors use parts of the validation dataset for training to generate a stable algorithm. In [41], a progressive resampling process is used. However, both works assume that all the data in question is available for training, which is not possible in RFMS.

Thresholdout Family: The author in [7] shows that differential privacy is deeply associated with model generalization and propose the Thresholdout algorithm to avoid overfitting on the validation set due to repetitive usage. [13] extends the instance wise Thresholdout to AUC measures. However, these methods rely on the i.i.d assumption of data which does not fit our scenario here.

Adaptive Regularization: In [30], the author proposed an alternative update method for model parameter θ and hyper-parameter $\phi = \lambda$ of a recommendation system [21], where the λ is the regularization parameter. In adaptive regularization, the update for the λ is based on the "future" value of performance which is also similar to the EM algorithm update process. However, adaptive regularization only works with gradient based algorithms. Especially, it is only implemented for Factorization Machine in libFM. So in general it does not work for non-gradient based optimization typed machine learning models.

Model Agnostic Meta Learning (MAML): Model Agnostic Meta Learning [10] originates from few shot learning. It aims at adapting to new instances, in which sense is similar to RFMS. However, MAML works only with gradient based method and pre-assumes that the algorithm could see the full subsequent dataset which is not possible in RFMS problem setting.

4 Methods

In this section, we first describe the general RFMS process in Sect. 4.1, then in Sect. 4.2, we propose how to handle the RFMS process with Bayesian Optimization.

4.1 Restrictive Federated Model Selection

The general process of RFMS is illustrated in Fig. 2, which depicts an asynchronous communication process during optimization. At step i, based on hyper-

parameter ϕ_i, the machine learning model is trained on D_{ob} to get the model parameter $\theta^i = \mathcal{L}(D_{ob} \mid \phi^i)$.

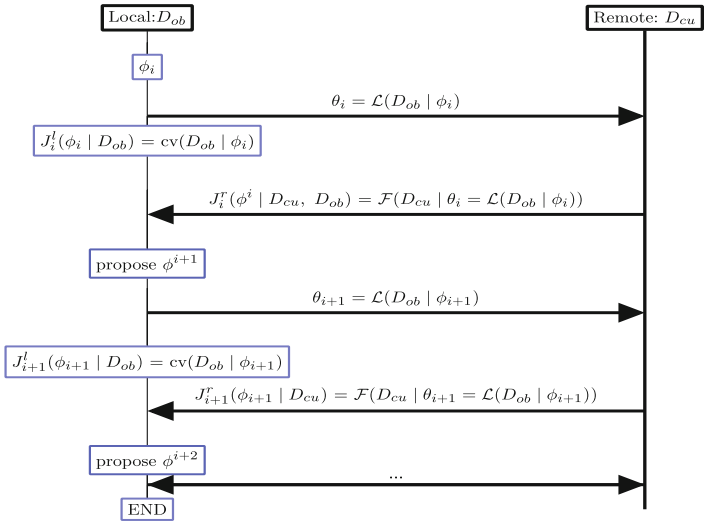


Fig. 2. Restrictive Federated Model Selection starting from step i

With the same hyper-parameter ϕ_i, a 10-fold cross validation is carried out on the openbox inbag part D_{ob}, which gives us one loss function in Eq. (2).

$$J_i^l(\phi_i \mid D_{ob}^{ig}) = cv(D_{ob}^{ig} \mid \phi_i) \tag{2}$$

where $cv(D_{ob}^{ig} \mid \phi_i)$ represent the average loss of the cross validation and J_i^l means local loss at the ith step.

Another loss function is obtained by sending the model parameters θ^i to the remote side as shown in Eq. (3)

$$J_i^r(\phi_i \mid D_{cu}^{ig}) = \mathcal{F}(D_{cu}^{ig} \mid \ \theta_i = \mathcal{L}(D_{ob}^{ig} \mid \phi_i)) \tag{3}$$

Here J_i^r means loss on the remote curator at the ith step.

At the next step, a decision process β (see Algorithm 1) based on all historical observations will propose a new hyper-parameter to be tried out for a potential better performance. This process is repeated until budget reached. The process should return the optimal hyper-parameters. The complete procedure is listed in Algorithm 1, where the decision process β to generate the proposal is approximately greedily taking the optimal of a Gaussian Process originated surrogate $\mu(\phi \mid \mathcal{R},\ \Phi)$, Expected Improvement [33], for instance. We use Φ (with an initial design sized n^{ini}) to represent the hyper-parameter buffer and $\mathcal{R}$ to represent the corresponding objective(s) buffer.

4.2 Bayesian Optimization and Baselines

Bayesian optimization tries to solve the problem of optimizing (often expensive-to-evaluate) black-box functions by using an internal empirical performance model which learns a surrogate model of the objective function while optimizing it. A widely used application for Bayesian Optimization [16] is the optimization of hyperparameters [1,33] of machine learning algorithms. Its aim is to find an optimal configuration $\phi^\star$ from the feasible region. The choice of hyperparameters for a machine learning model influences the learned model and can thus result in different performances (cf. [28,31]).

Since the distribution of the data across different data sites is unknown, we propose to treat the model selection approach as a black box optimization problem. Specifically, we use Bayesian Optimization in Algorithm 1 to solve the Restrictive Federated Model Selection problem with the following variants.

Local Single Objective (lso) Bayesian Optimization: In local single objective (lso) Bayesian Optimization, we set objective function as cross validation performance on the local openbox data site, hyper-parameters are tuned based on $J^{lso}(\phi) = J^l = cv\left(D_{ob}^{ig} \mid \phi\right)$ where J^l is defined in Eq. (2).

Federated Single Objective (fso) Bayesian Optimization: In Federated Single Objective Bayesian Optimization, we combine the openbox cross validation aggregated results in Eq. (2) and curator performance in Eq. (3) linearly as objective function, hyper-parameters are tuned based on

$$\begin{aligned} &J^{fso}(\phi) = \alpha\ J^l(\phi \mid D_{ob}^{ig}) + (1-\alpha)\, J^r(\phi \mid D_{cu}^{ig}) \\ &\alpha \in [0,\ 1]\,. \end{aligned} \tag{4}$$

Specifically, we use $fso2$ to represent $\alpha = 0.2$ and $fso8$ to represent $\alpha = 0.8$ and so on. Note that $\alpha = 1$ corresponds to lso. We use different α to check if there is an obvious effects by changing α.

Federated Multiobjective Objective (fmo) Bayesian Optimization: Multiobjective Bayesian Optimization [15] optimizes multiple objectives simultaneously, by random linear combination or optimization a S-metric based objective, which avoid deciding which linear combination parameter α to choose. In this work, we use the Parego algorithm [18] to optimize the local objective in Eq. (2) and remote objective in Eq. (3) jointly.

Random Search Multiobjective (rand_mo): To evaluate whether Bayesian optimization makes sense, we randomly search the hyper-parameter space and select the pareto front [38] as final output, which we call random search multi-objective.

4.3 Semi-simulation of Data Sites

Publicly available datasets which could fit into the RFMS scenario intrinsically are rare. To get data from a diversified source aside from the Gene Expression Ominbus, we turn to approximate the RFMS scenario by splitting an existing dataset into different parts as if each part sits on a different data site. In practice,

638 X. Sun et al.

Algorithm 1. RFMS with Bayesian Optimization (RFMS-BO)

procedure RFMS-BO ▷ data site notation here refer to the **inbag** part
  $\Phi_{1:n^{ini}} = \{\phi_1, \ldots, \phi_{n^{ini}}\}$ ▷ initial design as hyper-parameter buffer
  $\mathcal{R}_0 = \emptyset$ ▷ objective buffer
  for i in $1 : n^{ini}$, ϕ_i in $\Phi_{1:n^{ini}}$ **do**
    $J_i^l(\phi_i \mid D_{ob}) = cv(D_{ob} \mid \phi_i)$ ▷ Cross validation performance aggregation as loss
    $\theta_i = \mathcal{L}(D_{ob} \mid \phi_i)$ ▷ training on D_{ob} with ϕ_i
    $J_i^r(\phi_i \mid D_{cu}, D_{ob}) = \mathcal{F}(D_{cu} \mid \theta_i)$ ▷ test on curator
    $\mathcal{R}_i = \mathcal{R}_{i-1} \| \left[J_i^l, J_i^r\right]$ ▷ populate objective buffer
  end for
  fit $\mu(\phi \mid \mathcal{R}_i,\ \Phi_{1:n^{ini}})$ ▷ train Surrogate Function
  $j = i + 1$
  while budget not reached **do**
    $\phi_j = \beta(\mu(\phi \mid \mathcal{R}_{j-1},\ \Phi_{1:j-1}))$ ▷ propose new hyper-parameter
    $\Phi_{1:j} = \Phi_{1:j-1} \| [\phi_j]$ ▷ populate hyper-parameter buffer
    $J_j^l(\phi_j \mid D_{ob}) = cv(D_{ob} \mid \phi_j)$
    $\theta_j = \mathcal{L}(D_{ob} \mid \phi_j)$
    $J_j^r(\phi_j \mid D_{cu}) = \mathcal{F}(D_{cu} \mid \theta_j)$
    $\mathcal{R}_j = \mathcal{R}_{j-1} \| \left[J_j^l,\ J_j^r\right]$ ▷ populate objective buffer
    $j \leftarrow j + 1$
    update $\mu(\phi \mid \mathcal{R}_j,\ \Phi_{1:j})$ ▷ update surrogate
  end while
  $i^* = \arg\max_i(\mathcal{R})$
  $\{\phi^*\} = \Phi_{i*}$
  $\{\theta^*\} = \mathcal{L}(D_{ob}; \phi^*)$
  return ϕ^*, θ^*
end procedure

we always split an existing dataset into 5 parts to keep consistence with our GEO datasets.

Since we use real data, but kind of simulate to split the dataset into different data sites to fit into the RFMS scenario, we call this semi-simulation of data sites. We propose the following strategy to semi-simulate the data sites.

Stratified Random Split (SRS): First, split the dataset into two parts according to a factor column. Specifically, we use the target column in a classification dataset. Then, each factor part is randomly split into 5 buckets. The positive class part got $b_1^p, \ldots, b_5^p$ and the negative class part got $b_1^n, \ldots, b_5^n$, where b_i^n and b_i^p represent the ith bucket in the negative part and positive part respectively. Lastly, sort the buckets in each factor part according to the number of instances and combine the buckets in reversing order to form each data site, i.e., $d_i = b^n_{s^n(i)} \| b^p_{s^p(6-i)}$, where d_i represents the ith combined data site, s^n and s^p are the sorted index vector of each part. We use $\|$ to denote pooling two data buckets.

Dimension Reduction and Clustering (DRC): First, carry out a dimension reduction technique on the dataset like Principal Component Analysis. Then

split the dataset into positive class part and negative class part. Cluster each part into 5 clusters, i.e. $c_1^n, \ \ldots, \ c_5^n$ for the negative class part and $c_1^p, \ \ldots, \ c_5^p$ for the positive class part. Sort the clusters with respect to the cluster size in each part and combine them in reversed order to form each data site, i.e., $d_i = c^n_{s^n(i)} \| c^p_{s^p(6-i)}$, where d_i represent the ith combined data site, s^n and s^p are the sorted index vector of each part. We use $\|$ to denote pooling two data together.

We choose Mixture of Gaussian Model (MOG) for the clustering, due to consideration that MOG could also serve as a density estimator.

$$p(X) = \sum_{k=1}^{5} c_k \mathcal{N}(X|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{5}$$

In MOG, each cluster is represented by a Gaussian distribution $\mathcal{N}(X|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ with its own parameters $\boldsymbol{\mu}_k$(mean) and $\boldsymbol{\Sigma}_k$(covariance), as shown in Eq. (5), c_k is the mixing coefficient of each cluster. For each of the chosen datasets in Table 2, we model the data distribution as $p(X)$ in Eq. (5) and approximately, each cluster resulted data site represent a different distribution. For simplicity, we assume all clusters are with different mean vectors but share the same covariance matrix to assemble a distribution shift. The T-SNE plot is done to the SRS scenario (Fig. 3) and the DRC scenario. In DRC, we use PCA as dimension reduction, keeping 10% (Fig. 4) and 50% (Fig. 5) of the total variance to tell if the reduced dimension makes a big difference in generating an unevenly distributed data sites scenario). From these figures, we do not observe a big difference between different percentage of variance to reserve in PCA, but observe a big difference between SRS and DRC where SRS generates a more evenly distributed data sites, while DRC generates more uneven distributions across different clusters (data sites).

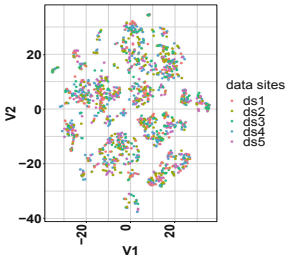


Fig. 3. Stratified random split (SRS)

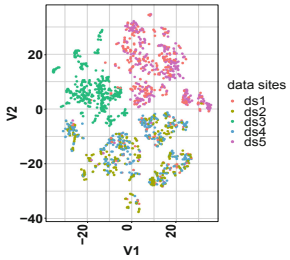


Fig. 4. DRC with PCA and keep 10% variance

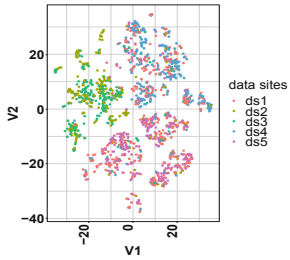


Fig. 5. DRC with PCA and keep 50% variance

5 Experiment

5.1 Settings

Since we have selected 5 datasets from the Gene Expression Ominibus to represent 5 data sites, we will consider the exemplary problem of 5 data sites for the remainder of the paper.

In the experiment, one of the 5 data sites is used as openbox D_{ob}, another one as lockbox D_{lb} and the three left over are used as curators D_{cu}. We choose to have only one openbox to simulate the scenario, that usually only local data at the current data sites are fully available to the analyst. We choose to have 3 curators and only 1 lockbox to simulate the scenario that more data sites want to collaborate with the openbox data site. Curator data site losses are weighted by the size of the each curator data site during optimization. With this strategy, there are in total $5 \times 4 = 20$ combinations of openbox-curator-lockbox on the 5 datasets. Each openbox and lockbox combination defines one scenario. Each scenario is repeated 10 times (10 replications) where we call each replication one experiment. We sequentially run all RFMS methods, described in Sect. 4.2, with 3 machine learning algorithms (kernel support vector machine, random forest and elastic net). Thus, we have in total $20 \times 10 \times 3 = 600$ experiments given a RFMS problem with 5 data sites. All Bayesian Optimization procedures share the same initial design of 20 randomly selected configurations, and are then run for another 40 iterations. Thus in total we have a budget of 60 evaluations. To have a fair comparison, Random Search use the same number of evaluations.

In order to evaluate our method, we randomly partition openbox and the curator into two parts, namely an inbag part (80%) and an outbag part (20%). Replications mentioned above could average out the random splits and other stochastic factors. We use D_{ob}^{ig} for training a model, and use D_{cu}^{ig} as well as D_{ob}^{ig} for model selection. The outbag parts of openbox and curator are reserved for post-hoc analysis. This allows us to assess, whether our methods overfit in each of the two boxes. Additionally, performance is also recorded on the lockbox site for another aspect of evaluation. We then compare the different methods described in Sect. 4.2 on the outbag portion of the respective boxes (as noted in Sect. 2.1, all data of lockbox belongs to outbag).[1]

5.2 Selection of Dataset for Semi-simulation

In order to validate our results on different data sources, we obtain additional data sets from OpenML [39]. As no datasets with an intrinsic splitting mechanism such as the GEO dataset (where each dataset comes from a particular source) are available, we simulate the RFMS scenario according to the strategies described in Sect. 4.3.

Model generalization becomes more difficult when there are comparatively more features than instances. Therefore, we restrict ourselves to datasets with a

[1] source code in https://github.com/compstat-lmu/paper_2019_multiobjective_rfms

relatively high-dimension characteristics: Since we intend to split a dataset into 5 parts as 5 data sites, the number or instances in each data site is approximately reduced by 5 times compared to the original dataset (we rebalanced cluster results which generate too small clusters but adding instances to the smallest cluster from the biggest cluster until the smallest cluster reaches 10% of the total number of instances), but the number of features over the number of instances get to be approximately 5 times of the original ratio, so a p (number of features) over n (number of instances) ratio of more than 0.2 in the original dataset corresponds to $\frac{p}{n} = 1$ in each data site, thus we consider datasets with $\frac{p}{n}$ ratio around 0.2 to be high-dimensional.

Too few instances is more prone to problems in data resampling processes like cross validation. For example, one fold of the cross validation might contain no instance from the underrepresented class. Thus we do not want too extremely unbalanced classification datasets. In order to have a sufficient amount of data in each of the 5 boxes, we select only data sets with more than 500 instances. For the purpose of simplicity, we additionally restrict our data set selection to data sets that are (i) binary class, (ii) do not have missing values. As a result, we use the data sets in Table 2 to provide additional validation of the proposed methods.

Table 2. List of datasets from OpenML

Name	n	p	p/n	Class ratio
gina agnostic	3468	970	0.28	0.97
Bioresponse	3751	1776	0.47	0.84
$fri\_c4\_500\_100$[a]	500	100	0.2	0.77

[a]https://www.openml.org/d/742

Since close or even identical predicative performance values on a problem can occur for varying machine learning hyperparameters, when the predicative performance is used as the target for Gaussian Process regression, it can create numerical difficulties, so hyperparameter tuning might fail for a particular algorithm, even though we use a nugget value of $1e-6$. Therefore, to get fair comparison, all algorithms are run sequentially over a problem on the same computing node. Only those experiments with all algorithms finished are used for analysis, where in practice, we only get neglectable number of experiments (around 100 out of 1800 experiments, which is 5 percent) within which at least one algorithm is not finished, see Fig. 9 and Fig. 11. The Winner-vs-Loser plots are more effective than carrying out statistical tests.

5.3 Machine Learning Algorithms and Hyper-Parameters

We choose 3 machine learning algorithms (which we call learner) based on the consideration that the learners should be representative to different mechanisms

of various machine learning algorithms. Elastic net logistic regression (implemented in R package *glmnet* [11]) is a good representative for linear classifier which could deal with high dimensional data (**classif.glmnet**), thus chosen because according to [35], one should not rule out simple models prematurely. R package *ranger* [40] implements a random forest (**classif.ranger**) which is a state of art non-linear learner that has shown outstanding performance. Kernel support vector machine (*ksvm*) (**classif.ksvm**) implemented in [17], is a nonlinear classifier which could deal with high dimensional data. The hyper-parameters to be optimized with their ranges are shown in Table 3. Hyper-parameter tuning is done with *mlr*[4] and *mlrMBO*[5]. Meaning of hyper-parameters can be found in respective packages.[2]

Table 3. List of hyperparameters

Classifier	Hyperparameter	Type	Range
glmnet	alpha	numeric	(0, 1)
glmnet	s	numeric	$(2^{-10}, 2^{10})$
ksvm	C	numeric	$(2^{-15}, 2^{15})$
ksvm	sigma	numeric	$(2^{-15}, 2^{15})$
random forest	num.trees	integer	(100, 5000)
random forest	min.node.size	integer	(1, 50)
random forest	sample.fraction	numeric	(0.1, 1)

5.4 Results and Discussion

In this section, we compare different candidates of RMFS methods proposed in Sect. 2.3 with respect to their predictive performance. Our aim is to obtain machine learning models, that generalize well across data sites. As an aggregate measure, we choose the dominated hypervolume of the data kept out-of-bag in the openbox D_{ob}^{og}, curator D_{cu}^{og} and lockbox D_{lb} respectively as shown in Eq. (1). We consider the average performance on the curators for calculating the hypervolume. Lockbox data measures how our methods generalize to subpopulations not considered at all during the training and model selection process. Using hypervolume results in a comprehensive overview of them.

Results on the GEO Datasets. As shown in Fig. 6, we compare the mean dominated hypervolume from Eq. (1) of 3 machine learning algorithms (corresponding to the 3 panels in the plot) and several RFMS methods. We aggregate

[2] https://github.com/mlr-org/mlr/blob/3edac9f65ed5c157a3d868fe8d2908eaa2a09ebd/R/RLearner_classif_glmnet.R\#L7.

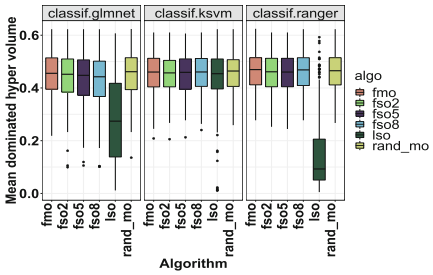


Fig. 6. Dominated hypervolume on GEO datasets

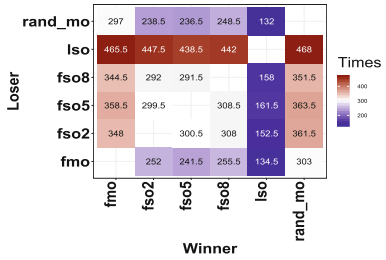


Fig. 7. Comparison of wins and losses on GEO dataset

over 10 replications and 20 combinations of possible openbox-lockbox combinations.

From Fig. 6, we can observe that *lso* performs the worst among other candidates, showing that in the RFMS scenario, solely tuning hyper-parameters on the local openbox data site will usually not lead to a model that generalizes well across data sites, which is in accordance with intuition. The other candidates methods including **fmo** and several **fso** variants, that predicting on the data of the curator and using this performance as a feedback performs better, showing that the feedback could help in arriving at models which generalize better. However, the considered Bayesian Optimization approaches do not overrate the multi-objective random search **rand_mo**, nor do we observe any effect of changing α in the performance of **fso**. In order to make a more precise comparison, we compare the pairwise wins and losses of all the RFMS methods in terms of dominated hypervolume. For each experiment, we build a $0-1$ matrix to compare the win and loss of each algorithm pair (when method A is compared against method B, we take 0 for loss, 1 for win, and 0.5 for tie) and aggregate the matrix across all 600 experiments. Results are shown in Fig. 7, where the horizontal axis corresponds to winners and the vertical axis correspond to losers. The elements in the matrix correspond to how many times the winner has won against the loser. It is easily observable that both bi-criteria methods (**fmo** and **rand_mo**) are slightly better than other candidates, as they win more than half of the experiments.

Results on the Semi-simulated RFMS Scenario. To avoid single dataset bias, we also analyze how the same algorithms compare under our semi-simulated RFMS scenario described in Sect. 4.3 over data of various sources.

Dimension Reduction and Clustering (DRC): We first simulate the RFMS scenario with DRC explained in Sect. 4.3, which could result in a situation that data from different data sites are differently distributed, where we keep 10 percent variance in the PCA step.

644 X. Sun et al.

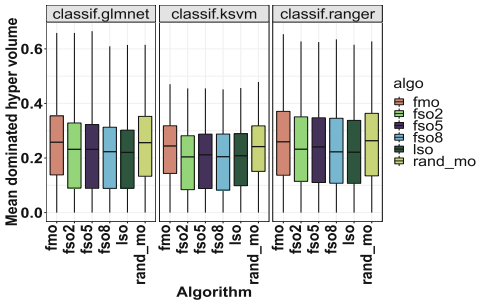


Fig. 8. Aggregated mean dominated hypervolume under DRC scenarios obtained over OpenML datasets

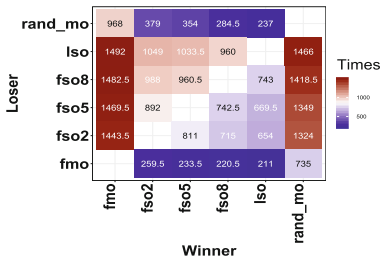


Fig. 9. Aggregated wins and losses on the DRC scenarios obtained over OpenML datasets

Figure 8 shows the dominated hypervolume by aggregating across all the datasets in Table 2. Compared to Fig. 6, it is more obvious here that the multi-objective methods work better than the single objective Bayesian optimization methods. In Fig. 9, we have the Winner-vs-Loser plot for the aggregated results on the OpenML datasets listed in Table 2, where the multi-objective candidates outperform the rest by a large margin. Furthermore, **fmo** wins **rand_mo** by a considerable margin, giving confidence that Bayesian Optimization make a difference compared to random search.

Stratified Random Split (SRS): To answer the question if a different data splitting technique affects the comparison, we use the stratified technique described in Sect. 4.3 which corresponds to the situation that data being more evenly distributed across data sites. Figure 10 shows the hypervolume plot, from which we can still observe the pattern that the multi-objective candidates perform better in terms of hypervolume, while compared to Fig. 8, all methods show increased performance under this evenly distributed data scenario across data sites, possibly due to the bonus of evenly distributed data scenario. In Fig. 11, we compare the wins and losses for each pair of candidates, where in this case, the **fmo** wins **rand_mo** by a larger margin, maybe because the generate a simpler RFMS scenario for the Bayesian Optimization.

6 Summary

We introduce a novel learning scenario, Restrictive Federated Model Selection (RFMS), which could play an important role in clinical research, where privacy sensitive immobile high dimensional data is differently distributed among various data sites, in which case federated learning is not applicable due to a lack of access to data from all data sites to be used for training. RFMS is a model selection process in this scenario, with the aim to obtain a model that generalizes comparably well across data sites with potential different distributions. Compared to Federated Learning, RFMS can be carried out in an asynchronous

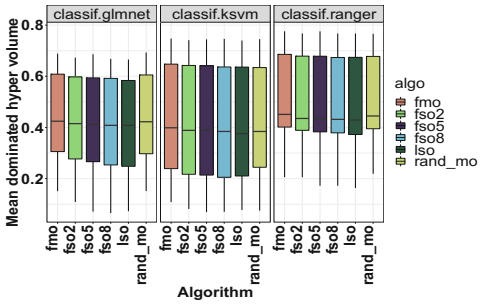


Fig. 10. Aggregated mean dominated hyper-volume under SRS scenario obtained over OpenML datasets

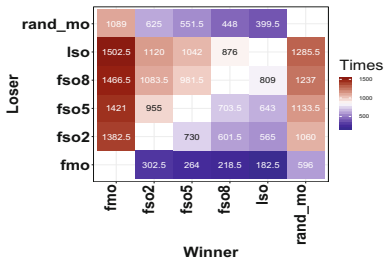


Fig. 11. Aggregated wins and losses over SRS scenario obtained over OpenML datasets

fashion, which is not communication hungry compared to standard federated learning and much easier to be deployed. Additionally, the amount of information that needs to be transferred for each query is comparatively small which takes less efforts to be deployed.

As an initial investigation, we compare various methods for model selection and hyper-parameter tuning using Bayesian Optimization. Empirical results from various data sources indicate that Federated Multi-objective Bayesian Optimization compares favorably against other single objective candidates as well as multi-objective random search, in terms of better generalization across data sites.

Acknowledgment. This work was supported by Deutsche Forschungsgemeinschaft (DFG), Project RA 870/7-1 and BI 1902/1-1. The authors thank Janek Thomas, Philip Probst and Martin Binder for helpful suggestions.

References

1. Bergstra, J.S., Bardenet, R., Bengio, Y., Kégl, B.: Algorithms for hyper-parameter optimization. In: Advances in Neural Information Processing Systems, pp. 2546–2554 (2011)
2. Beume, N., Rudolph, G.: Faster s-metric calculation by considering dominated hypervolume as klee's measure problem. Universitätsbibliothek Dortmund (2006)
3. Bhowmick, A., Duchi, J., Freudiger, J., Kapoor, G., Rogers, R.: Protection against reconstruction and its applications in private federated learning. arXiv:1812.00984 (2018)
4. Bischl, B., Lang, M., Kotthoff, L., Schiffner, J., Richter, J., Studerus, E., Casalicchio, G., Jones, Z.M.: mlr: Machine learning in R. J. Mach. Learn. Res. **17**(1), 5938–5942 (2016)
5. Bischl, B., Richter, J., Bossek, J., Horn, D., Thomas, J., Lang, M.: mlrMBO: a modular framework for model-based optimization of expensive black-box functions. arXiv preprint arXiv:1703.03373 (2017)
6. Cawley, G.C., Talbot, N.L.: On over-fitting in model selection and subsequent selection bias in performance evaluation. J. Mach. Learn. Res. **11**, 2079–2107 (2010)

7. Dwork, C., Feldman, V., Hardt, M., Pitassi, T., Reingold, O., Roth, A.: Guilt-free data reuse. Commun. ACM **60**(4), 86–93 (2017)
8. Edgar, R., Domrachev, M., Lash, A.E.: Gene Expression Omnibus: NCBI gene expression and hybridization array data repository. Nucl. Acids Res. **30**(1), 207–210 (2002)
9. Elder, S.: Bayesian adaptive data analysis guarantees from subgaussianity. arXiv preprint arXiv:1611.00065 (2016)
10. Finn, C., Abbeel, P., Levine, S.: Model-agnostic meta-learning for fast adaptation of deep networks. arXiv preprint arXiv:1703.03400 (2017)
11. Friedman, J., Hastie, T., Tibshirani, R.: glmnet: Lasso and elastic-net regularized generalized linear models. R Packag. Version **1**(4) (2009)
12. Geyer, R.C., Klein, T., Nabi, M.: Differentially private federated learning: A client level perspective. arXiv preprint arXiv:1712.07557 (2017)
13. Gossmann, A., Pezeshk, A., Sahiner, B.: Test data reuse for evaluation of adaptive machine learning algorithms: over-fitting to a fixed 'test' dataset and a potential solution. In: Medical Imaging 2018: Image Perception, Observer Performance, and Technology Assessment, vol. 10577, p. 105770K. International Society for Optics and Photonics (2018)
14. Guyon, I., Saffari, A., Dror, G., Cawley, G.: Model selection: beyond the Bayesian/frequentist divide. J. Mach. Learn. Res. **11**, 61–87 (2010)
15. Horn, D., Dagge, M., Sun, X., Bischl, B.: First investigations on noisy model-based multi-objective optimization. In: International Conference on Evolutionary Multi-Criterion Optimization, pp. 298–313. Springer (2017)
16. Jones, D.R., Schonlau, M., Welch, W.J.: Efficient global optimization of expensive black-box functions. J. Glob. Optim. **13**(4), 455–492 (1998)
17. Karatzoglou, A., Smola, A., Hornik, K., Zeileis, A.: kernlab-an S4 package for kernel methods in R. J. Stat. Softw. **11**(9), 1–20 (2004)
18. Knowles, J.: ParEGO: a hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. IEEE Trans. Evol. Comput. **10**, 50–66 (2006)
19. Konecnỳ, J., McMahan, H.B., Ramage, D., Richtárik, P.: Federated optimization: distributed machine learning for on-device intelligence. arXiv preprint arXiv:1610.02527 (2016)
20. Konečnỳ, J., McMahan, H.B., Yu, F.X., Richtárik, P., Suresh, A.T., Bacon, D.: Federated learning: strategies for improving communication efficiency. arXiv preprint arXiv:1610.05492 (2016)
21. Kushwaha, N., Sun, X., Singh, B., Vyas, O.: A lesson learned from pmf based approach for semantic recommender system. J. Intell. Inf. Syst. **50**(3), 441–453 (2018)
22. Maaten, L.v.d., Hinton, G.: Visualizing data using t-SNE. J. Mach. Learn. Res. **9**, 2579–2605 (2008)
23. McCall, M.N., Bolstad, B.M., Irizarry, R.A.: Frozen robust multiarray analysis (fRMA). Biostatistics **11**(2), 242–253 (2010)
24. McMahan, H.B., Moore, E., Ramage, D., Hampson, S., y Arcas, B.A.: Communication-efficient learning of deep networks from decentralized data. In: AISTATS (2017)
25. McMahan, H.B., Moore, E., Ramage, D., Hampson, S., et al.: Communication-efficient learning of deep networks from decentralized data. arXiv preprint arXiv:1602.05629 (2016)
26. Melis, L.: Building and evaluating privacy-preserving data processing systems. Ph.D. thesis, UCL (University College London) (2018)

27. Melis, L., Song, C., De Cristofaro, E., Shmatikov, V.: Inference attacks against collaborative learning. arXiv preprint arXiv:1805.04049 (2018)
28. Probst, P., Bischl, B., Boulesteix, A.L.: Tunability: Importance of hyperparameters of machine learning algorithms. arXiv preprint arXiv:1802.09596 (2018)
29. Rabanser, S., Günnemann, S., Lipton, Z.C.: Failing loudly: an empirical study of methods for detecting dataset shift. arXiv preprint arXiv:1810.11953 (2018)
30. Rendle, S.: Learning recommender systems with adaptive regularization. In: Proceedings of the Fifth ACM International Conference on Web Search and Data Mining, pp. 133–142. ACM (2012)
31. van Rijn, J.N., Hutter, F.: Hyperparameter importance across datasets. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 2367–2376. ACM (2018)
32. Shokri, R., Stronati, M., Song, C., Shmatikov, V.: Membership inference attacks against machine learning models. In: 2017 IEEE Symposium on Security and Privacy (SP), pp. 3–18 (2017). https://doi.org/10.1109/SP.2017.41
33. Snoek, J., Larochelle, H., Adams, R.P.: Practical Bayesian optimization of machine learning algorithms. In: Advances in Neural Information Processing Systems, pp. 2951–2959 (2012)
34. Snoek, J., Rippel, O., Swersky, K., Kiros, R., Satish, N., Sundaram, N., Patwary, M., Prabhat, M., Adams, R.: Scalable Bayesian optimization using deep neural networks. In: International Conference on Machine Learning, pp. 2171–2180 (2015)
35. Strang, B., van der Putten, P., van Rijn, J.N., Hutter, F.: Don't rule out simple models prematurely: a large scale benchmark comparing linear and non-linear classifiers in OpenML. In: International Symposium on Intelligent Data Analysis, pp. 303–315. Springer (2018)
36. Tennenholtz, G., Zahavy, T., Mannor, S.: Train on validation: squeezing the data lemon. arXiv preprint arXiv:1802.05846 (2018)
37. Truex, S., Baracaldo, N., Anwar, A., Steinke, T., Ludwig, H., Zhang, R.: A hybrid approach to privacy-preserving federated learning. arXiv preprint arXiv:1812.03224 (2018)
38. Van Veldhuizen, D.A., Lamont, G.B.: Evolutionary computation and convergence to a pareto front. In: Late Breaking Papers at the Genetic Programming 1998 Conference, pp. 221–228 (1998)
39. Vanschoren, J., van Rijn, J.N., Bischl, B., Torgo, L.: OpenML: networked science in machine learning. ACM SIGKDD Explor. Newsl. **15**(2), 49–60 (2014)
40. Wright, M.N., Ziegler, A.: Ranger: a fast implementation of random forests for high dimensional data in C++ and R. arXiv preprint arXiv:1508.04409 (2015)
41. Zeng, X., Luo, G.: Progressive sampling-based bayesian optimization for efficient and automatic machine learning model selection. Health Inf. Sci. Syst. **5**(1), 2 (2017)
42. Zhang, K., Schölkopf, B., Muandet, K., Wang, Z.: Domain adaptation under target and conditional shift. In: International Conference on Machine Learning, pp. 819–827 (2013)
43. Zhao, Y., Li, M., Lai, L., Suda, N., Civin, D., Chandra, V.: Federated learning with non-iid data. CoRR abs/1806.00582(2018)

Appendix C

Hierarchical Variational Auto-Encoding for Unsupervised Domain Generalization

Contributing Article Sun, Xudong; Buettner, Florian; Hierarchical Variational Auto-Encoding for Unsupervised Domain Generalization, https://arxiv.org/pdf/2101.09436.pdf, 2021. (Accepted at ICLR 2021 RobustML Workshop)

Author Contributions Xudong Sun designed and implemented the algorithm, experiment and wrote the paper. Florian Buettner supervised the study, contributed to method development and wrote the paper.

Hierarchical Variational Auto-Encoding for Unsupervised Domain Generalization

Xudong Sun* Florian Buettner†

Abstract

We address the task of domain generalization, where the goal is to train a predictive model such that it is able to generalize to a new, previously unseen domain. We choose a generative approach within the framework of variational autoencoders and propose an unsupervised algorithm that is able to generalize to new domains without supervision. We show that our method is able to learn representations that disentangle domain-specific information from class-label specific information even in complex settings where domain structure is not observed during training. Our interpretable method outperforms previously proposed generative algorithms for domain generalization and achieves competitive performance compared to state-of-the-art approaches, which rely on observing domain-specific information during training, on the standard domain generalization benchmark dataset PACS. Additionally, we proposed weak domain supervision which can further increase the performance of our algorithm in the PACS dataset.

1 Background and Motivation

One big challenge of deploying a neural network model in real world use-cases is domain shift. In many real world applications, data seen by a deployed model is drawn from a distribution that is different from the training distribution and often unknown at train time. Domain Generalization aims at training a model from a set of domains (i.e. related distributions) such that the model is able to generalize to a new, unseen domain at test time.

Domain generalization is relevant for a variety of tasks, ranging from personalized medicine, where each patient corresponds to a domain, to predictive maintenance in the context of industrial AI. In the latter use-case, domains can represent different factories where an industrial asset (e.g. a tool machine or a turbine) is operated, or different workers operating the asset. In addition to these discrete domains, domain shift can manifest itself in a continuous manner, where for example the data distribution seen by an industrial asset can change due to wear and tear or due to maintenance procedures. Similarly, domain sub-structures are not always observable during training due to data privacy concerns (in particular when patient data is used). In these latter scenarios, it is difficult to train standard domain generalization algorithms since they are based on the notion of clearly separable domains that are observable during model training.

In many of these use cases, interpretability and human oversight of machine learning models is key. Generative models allow for learning disentangled representations that correspond to specific and interpretable factors of variation, thereby facilitating transparent predictions.

We propose a new generative model that solves domain generalization problems in an interpretable manner without requiring domain labels during training. We build on previous work using autoencoder-based models for domain generalization [Kingma and Welling, 2013, Ilse et al., 2019] and propose a Hierarchical Domain Unsupervised Variational Auto-encoding that we refer to as HDUVA. Our major contributions include:

- We present an unsupervised algorithm for domain generalization that is able to learn in setting with incomplete or hierarchical domain information. Our algorithm only need to use extended ELBO as model selection criteria, instead of relying on the validation set for early stopping.
- Our method is able to learn representations that disentangle domain-specific information from class-label specific information without domain supervision even in complex settings.
- Our algorithm generates interpretable domain predictions that reveal connections between domains.
- We constructed several hierarchical and sequential domain generalization benchmark datasets with doubly colored mnist for the domain generalization community.

2 Related work

In this section, we provide a taxonomy of existing solutions in domain generalization. In general, domain generalisation approaches can be divided into the following main categories, that we describe below.

*LMU Munich, smilesun.east@gmail.com, work partially done during intern at Siemens AG.

†Siemens AG, buettner.florian@siemens.com

Invariant Feature Learning While observations from different domains follow different distributions, Invariant Feature Learning approaches try to map the observations from different domains into a common feature space, where domain information is minimized [Xie et al., 2017, Akuzawa et al., 2018]. The method works in a mini-max game fashion in that there is a domain classifier trying to classify domains from the common feature space, while a feature extractor tries to fool this domain classifier and help the target label classifier to classify class label correctly. Li et al. [2017] presented a related approach and used tensor decomposition to learn a low rank embedding for a set of domain specific models as well as a base model. We classify this method into invariant feature learning because the base model is domain-invariant.

Image Processing Based Method Carlucci et al. [2019] divided the image into small patches and generated permutations of those small patches. They then used a deep classifier to predict the predefined permutation index so that the model learned the global structure of an image instead of local textures. Wang et al. [2019] used a gray level co-occurence matrix to extract superficial statistics. They presented two methods to encourage the model to ignore the superficial statistics and thereby learn robust representations. This group of methods has been developed for image classification tasks, and it is not clear how it can be extended to other data types.

Adversarial Training Based Data Augmentation Volpi and Murino [2019] optimized a procedure to search for worst case adversarial examples to augment the training domain. Volpi et al. [2018] used Wasserstein distance to infer adversarial images that were close to the current training domain, and trained an ensemble of models with different search radius in terms of Wasserstein distance.

Meta Learning Based Method Meta learning based domain generalization method (MLDG) uses model agnostic training to tackle domain generalization as a zero-shot problem, by creating virtual train and test domains and letting the meta-optimizer choose a model with good performance on both virtual train and virtual test domains [Li et al., 2018]. Balaji et al. [2018] improved upon MLDG by concatenating a fixed feature network with task specific networks. They parameterized a learnable regularizer with a neural network and trained with a meta-train and a meta-test set .

Auto-Encoder Based Method DIVA [Ilse et al., 2019] builds on variational auto-encoders and splits the latent representation into three latent variables capturing different sources of variation, namely class specific information (z_y), domain specific information (z_d) and residual variance (z_x). Disentanglement is encouraged via conditional priors, where the domain-specific latent variable z_d is condition on an observed, one-hot-encoded domain d. As auxiliary components, DIVA adds a domain classifier based on z_d, as well as a target class label classifier based on z_y. Hou et al. [2018] encoded images from different domains in a common content latent code and domain-specific latent code, while the two types of encoders share layers. Corresponding discriminators are used to predict whether the input is drawn from a prior distribution or generated from encoder.

Causality based Method Recently, Mahajan et al. [2020] proposed MatchDG with that approximates base object similarity by using a contrastive loss formulation adapted for multiple domains. The algorithm then match inputs that are similar under the invariant representation.

Comparing these families of approaches, we can see that only probabilistic auto-encoder based models inherit advantageous properties like semi-supervised learning, density estimation and variance decomposition naturally. While autoencoder-based approaches such as DIVA have a better interpretability than all other approaches, a main drawback is that explicit domain labels are required during training.This can be problematic in a number of settings. In particular, a one-hot encoding of domains does not reflect scenarios where a continuous domain shift can occur. In this case, without knowledge of the causal factor that causes the domain shift, it is not clear how such continuous shifts can be one-hot encoded in a meaningful manner. In addition,

- Domains can have a hierarchical structure reflected by related sub-domains (e.g. country > factory > machine). One-hot encodings as used in existing autoencoder-based approaches are not able to model such hierarchical domain structures.

- In some applications, domains are not necessarily well-separated, but significant overlap between domains can occur (e.g. a cartoon might look more similar to a pop-art painting than a photography). One-hot encoding such overlapping domains encourages separated representations, which may harm model performance.

- A one-hot encoding of domains mapping to the prior distribution of z_d may limit the generalization power of neural networks, especially when we deal with continuous domain shift.

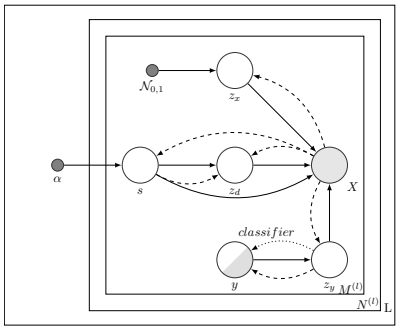


Figure 1: **HDUVA: Hierarchical Domain Unsupervised Variational Auto-encoding**

3 Methods and Technical Solution

3.1 Problem statement and notation Domain generalization aims to generalize models to unseen domains without knowledge about the target distribution during training. A domain d consists of a joint distribution $p(x, y)$ on $\mathcal{X} \times \mathcal{Y}$, with $\mathcal{X}$ being the input space and $\mathcal{Y}$ being the output space [Muandet et al., 2013]. For our modelling approach, we employ the framework of variational autoencoders (VAEs) [Kingma and Welling, 2013]. We use z to represent the latent representation of a VAE and use three independent latent representations to disentangle variability in inputs X related to domain-specific sources, label-specific sources and residual variation. We use probabilistic graphical models to illustrate the conditional dependecies of random variables, observables and hyperparameters in Figure 1. In the graphical model of Figure 1, solid circles represent observations and white circles represent latent variables. We use half-shaded circles to represent a variable can either be observed or act as latent variable, which is typical in semi-supervised learning. Small solid circles in Figure 1 represent fixed hyper-parameters. Subscripts represent components of a variable, while we use superscript to index samples and domains. We use solid arrows to represent generative path, and dashed arrows to represent variational inference part. Plates represent repetitions of random variables. We use θ to represent learnable parameters of priors/decoders and ϕ to represent learnable parameters of variational posterior distributions/encoders.

3.2 HDUVA overview To overcome the limitations of current autoencoder-based ethods, we propose a hierarchical probabilistic graphical model called Hierarchical Domain Unsupervised Variational Auto-encoding(HDUVA). Our model is based on three latent variables are used to model distinct sources of variation that are denoted as z_y, z_d and z_x. z_y represents class specific information, z_d represents domain specific information and z_x models residual variance of the input. We introduce an additional hierarchical level and use a continuous latent representation s to model (potentially unobserved) domain structure. This means that we can encourage disentanglement of the latent variables through conditional priors without the need of conditioning on a one-hot-encoded, observed domain label.

More specifically, we first place a Dirichlet prior on s such that it can be interpreted as a soft, topic-like, version of the standard one-hot encoded domain d. We then use z_d to capture domain-specific variation by conditioning its prior on s. Note that in our model this domain s is not an observable but instead a latent variable to be inferred from data. For clarity, we refer to an observed domain as nominal domain. Borrowing from topic models in NLP [Srivastava and Sutton, 2017], we refer to s as **topic**. We illustrate HDUVA in form of a graphical model in Figure 1, where we form a hierarchical path Klushyn et al. [2019] from topic s to z_d to observation x. We use K to denote the dimension of the domain representation or topic vector s, i.e. $dim(s) = K$. We use k to index each component of s, i.e. $s^{(l)} = [s_1^{(l)}, s_2^{(l)}, \cdots, s_K^{(l)}]$, with l indexing a domain. Note that in our case, K can be either greater, smaller or equal to the number of domains L, while in supervised approaches, the one-hot encoded domain label is always the size of L. This is beneficial not only in setting with unobserved domain observation, but also for problems with a large number of domains which lie on a lower-dimensional manifold (e.g. thousands assets in an predictive maintenance task). In this case, when choosing the topic dimension K to be smaller than the number of training domains, our algorithm can be interpreted as an eigen-domain decomposition algorithm. We use stochastic gradient descent to train our model. Accordingly, in Figure 1, the batch size is denoted by $M^{(l)}$ for the lth domain, with a total of $N^{(l)}$ batches for domain l. We use i to index a batch and j to index a sample. For simplicity, i and j are omitted whenever convenient and not causing confusion. We also provide details on extensions for weak supervision to our model in the supplement.

Taken together, we present a novel approach for probabilistic domain generalization without the need for observed domain labels.

3.3 Model implementation In this section, we first describe the generative model with prior distributions,

followed by a discussion on model inference.

3.3.1 Prior Distributions for z_x, z_y and z_s We chose a standard isotropic Gaussian prior with zero mean and unit variance for z_x and conditional priors for for z_y and z_d. More specifically, we chose a normal prior for z_y that is conditioned on the target class label y:

$$p_{\theta_y}(z_y^{(l,i)}|y^{(l,i)}) = \mathcal{N}\left(\cdot|\mu_{\theta_y}(y^{(l,i)}), \sigma_{\theta_y}(y^{(l,i)})\right) \tag{3.1}$$

with μ_{θ_y} and σ_{θ_y} being learnable parameterizations of the mean and standard deviation in form of neural networks. Similarly, we choose a normal prior for z_d and condition it on s:

$$p_{\theta_d}(z_d^{(l,i)}|s^{(l,i)}) = \mathcal{N}\left(\cdot|\mu_{\theta_d}(s^{(l,i)}), \sigma_{\theta_d}(s^{(l,i)})\right) \tag{3.2}$$

where again μ_{θ_d} and σ_{θ_d} parameterize mean and variance of z_d.

3.3.2 Prior Distribution for s We would like for s to display topic-like characteristics, facilitating interpretable domain representations. Consequently, we use a Dirichlet prior on s, which is a natural prior for topic modeling [Srivastava and Sutton, 2017, Joo et al., 2020, Zhao et al., 2019].

Let α be the Dirichlet concentration parameter $\alpha = [\alpha_1, \alpha_2, \cdots, \alpha_K]$, then the prior distribution of s can be written as:

$$p(s^{(l,i)}|\alpha^l) = Dir(s^{(l,i)}|\alpha^l_{1:K}) = \frac{\prod_k (s_k^{(l,i)})^{\alpha_k^l - 1}}{\mathcal{Z}(\alpha^l_{1:K})} \tag{3.3}$$

where we use $\mathcal{Z}(\alpha_{1:K})$ to represent the partition function.

We do not learn the distribution parameter α, but instead, leave it as a hyper-parameter. By default, we set α to be a vector of ones, which corresponds to a uniform distribution of topics. We refer to this prior setting as flat prior. If more prior knowledge about the relation between training domains is available, an informative prior can be used instead.

3.3.3 Inference for HDUVA Since exact inference is intractable in such an autoencoder, we perform variational inference and introduce three separate encoders as follows:

$$\begin{aligned} &q_\phi(s^{(l,i)}, z_d^{(l,i)}, z_x^{(l,i)}, z_y^{(l,i)}|x^{(l,i)}) \\ =&q_{\phi_s}(s^{(l,i)}|x^{(l,i)}) q_{\phi_d}(z_d^{(l,i)}|s^{(l,i)}, x^{(l,i)}) q_\phi(z_x^{(l,i)}, z_y^{(l,i)}|x^{(l,i)}) \end{aligned} \tag{3.4}$$

For the approximate posterior distributions of z_x and z_y, we assume fully factorized Gaussians with parameters given as a function of their input:

$$q_\phi(z_x^{(l,i)}, z_y^{(l,i)}|x^{(l,i)}) = q_{\phi_x}(z_x^{(l,i)}|x^{(l,i)}) q_{\phi_y}(z_y^{(l,i)}|x^{(l,i)}) \tag{3.5}$$

Encoders q_{ϕ_s} , q_{ϕ_d}, q_{ϕ_y}, and q_{ϕ_x} are parameterized by ϕ_s, ϕ_d, ϕ_y, and ϕ_x using separate neural networks to model respective means and variances as function of x.

For the form of the approximate posterior distribution of the topic s we chose a Dirichlet distribution:

$$q_{\phi_s}(s^{(l,i)}|x^{(l,i)}) = Dir\left(s^{(l,i)}|\phi_s(x^{(l,i)})\right) \tag{3.6}$$

where ϕ_s parameterizes the concentration parameter based on x, using a neural network.

3.3.4 ELBO for HDUVA Given the priors and factorization described above, we can optimize the model parameters by maximizing the evidence lower bound (ELBO). We can write the ELBO for a given input-output tupel (x, y) as:

$$\begin{aligned} &ELBO(x,y) = E_{q(z_d,s|x),q(z_x|x),q(z_y|x)} \log p_\theta(x|s, z_d, z_x, z_y) \\ &- \beta_x KL(q_{\phi_x}(z_x|x)||p_{\theta_x}(z_x)) - \beta_y KL(q_{\phi_y}(z_y|x)||p_{\theta_y} \\ &(z_y|y)) - \beta_d E_{q_{\phi_s}(s|x),q_{\phi_d}(z_d|x,s)} \log \frac{q_{\phi_d}(z_d|x,s)}{p_{\theta_d}(z_d|s)} \\ &- \beta_s E_{q_{\phi_s}(s|x)} KL(q_{\phi_s}(s|x)||p_{\theta_s}(s|\alpha)) \end{aligned} \tag{3.7}$$

where we use β to represent the multiplier in the Beta-VAE setting [Higgins et al., 2016], further encouraging disentanglement of the latent representations.

Finally, we add an auxiliary classifier $q_\omega(y|z)$, which is parameterized by ω, to encourage separation of classes y in z_y. The HDUVA objective then becomes:

$$\mathcal{F}(x,y) = ELBO(x,y) + \gamma_y E_{q_{\phi_y}(z_y|x)}[\log q_\omega(y|z_y)] \tag{3.8}$$

The whole process is described in Algorithm 1. The objective function in Equation 3.8 which we coin extended ELBO can also be used as a model selection criteria, thus our method does not need validation set at all, as we empirically evaluated in the experimental section in section 4.

4 Empirical Evaluation

We conduct experiments, trying to answer the following questions:

- Could HDUVA mitigate the limitations of standard supervised approaches for domain generalization in terms of domain-substructure or overlap between nominal domains? We conduct experiments in

Algorithm 1 HDUVA

while not converged or maximum epochs not reached **do**
  warm up β defined in Equation 3.7, as in Sønderby et al. [2016]
  fetch mini-batch {x, y} ={$x^{(l,i)}, y^{(l,i)}$}
  compute parameters for $q_{\phi_x}(z_x|x)$, $q_{\phi_y}(z_y|x)$, $q_{\phi_s}(s|x)$, $q_{\phi_d}(z_d|s,x)$
  sample latent variable z_x^q, z_y^q, s^q, z_d^q and compute $[\log q_\omega(y|z_y)]$.
  compute prior distribution for z_d using s
  compute $p_\theta(x|z_x, z_y, z_d, s)$ using sampled s, z_x^q, z_y^q, z_d^q
  compute KL divergence for z_d, z_x and z_y, s.
  aggregate loss according to Equation 3.8 and update model
end while

Section 4.1, Section 4.2 and Section 4.4 to address these issues.

- In complex scenarios with domain substructure, can HDUVA still robustly disentangle domain-specific variation from class-label specific variation? See details in Section 4.1.

- We visualize topics from overlapping nominal domains to illustrate why HDUVA improves upon supervised approaches in Section 4.4.

- How does HDUVA perform under standard domain generalization benchmarks where information on clearly separated domain is available, compared with other state-of-the-art algorithms? See Section 4.5.

4.1 Hierarchical Domains To simulate domains with sub-structures (hierarchical domains), we create sub-domains within nominal domains. All sub-domains within one nominal domain share the same domain label. We adapt color-mnist [Metz et al., 2016, Rezende and Viola, 2018] with the modification that both its foreground and background are colored as sub-domain, as shown in Figure 2. We constructed 3 nominal domains with sub-structures as indicated in Figure 2. For baseline algorithms, we use a one-hot encoded nominal domain label as explicit domain label, since these methods require a domain label during training. For HDUVA, we do not use domain label and we only use extended ELBO in Equation 3.8 as model selection criteria, further experimental details can be found in supplement C.

(a) 1st domain (b) 2nd domain (c) 3rd domain

Figure 2: **Random combination of Color-Mnist as Hierarchical Domains**. Mnist has both its foreground and background colored, each color combination represent one sub-domain. Each nominal domains include 2 sub-domains.

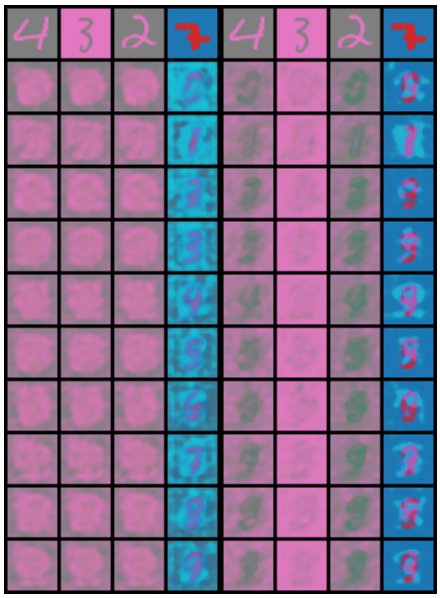
We are interested in evaluating how our unsupervised approach and supervised generative domain generalization algorithms like DIVA Ilse et al. [2019] for domain generalization would behave under this sub-domain scenario, in terms of out-of-domain prediction accuracy and disentanglement performance. We perform a leave-one-domain-out evaluation [Li et al., 2017], where each test domain is repeated 10 times with 10 different random seeds. We report the out of domain test accuracy in Table 1. Table 1 shows that HDUVA outperforms DIVA in terms of out of domain performance on all three test domains, while retaining a very small variance compared to DIVA.

To explain such a performance difference, we further evaluate how robustly DIVA and HDUVA are able to disentangle different sources of variation under this scenario with incomplete sub-domain information.

We sample seed images from different sub-domains as shown in the first row of Figure 3. We then generate new images by scanning the class label from 0 to 9 by sampling from the conditional prior distribution of z_y (i.e. $p_{\theta_y}(z_y|y)$, eq. 3.1). We keep the domain representation the same as in the seed image, set the noise component z_x to zero and then use the decoder network $p_\theta(x|z_d, z_x, z_y)$ to generate an image based on the three latent representations. If the models are able to disentangle domain-specific variation from class-label specific variation in z_y and z_d, we expect that the generated images have the same domain information as the seed image (foreground and background color) while generating different class labels (numbers from 0 to 9). In Figure 3 we compare DIVA and HDUVA's generative performance. Due to the sub-structure inside the nominal domains, DIVA could only reconstruct a blur of colors for the first 3 columns in Figure 3a, while

Table 1: **Out of Domain Accuracy on Color-Mnist Composed Subdomain inside Nominal Domains**

Color-Mnist (Figure 2)	Test Domain 1	Test Domain 2	Test Domain 3
HDUVA	**0.93** ± 0.02	**0.69** ± 0.12	**0.55** ± 0.03
DIVA [Ilse et al., 2019]	0.88 ± 0.05	0.56 ± 0.19	0.50 ± 0.08



(a) DIVA (b) HDUVA

Figure 3: **Comparison of Conditional Image Generation under Incomplete Domain Knowledge**. The domain composition is shown in Figure 2.

HDUVA could generate different numbers for 2 of the three seed images. For the last seed image, both DIVA and HDUVA could conditionally generate numbers, but DIVA did not retain the domain information (since the background color, which is dark blue in the seed image, is light blue in the generated images). This indicates that DIVA is not able to disentangle the different sources of variation and domain information is captured by z_y as well. In contrast, HDUVA was able to separate domain information from class-label information.

4.2 Generalization under domain-drift scenarios In some occasions, the boundaries between different domains can be ambiguous. For example, consider continuous domain drift in industry applications, some physical parameters of the same type of machine in different factories might change continuously and between two factories there can be overlap.

To simulate such a behavior, we consider a domain-drift scenario with Color-Mnist in Figure 4. By dividing a smooth color palette into 7 sub-domains (with each color corresponding to a sub-domain), we simulate a near continuous domain shift. We use this scenario to evaluate how robust our algorithm is in domain drift scenarios.

Following leave-one-domain out setting as in other experiments, we report the out-of-domain classification accuracy in Table 2, illustrating that our unsupervised approach is better able to account for continuous domain drift scenarios than standard supervised approaches that artificially categorize the gradually shifting into distinct nominal domains. For HDUVA, we only use extended ELBO in Equation 3.8 as model selection criteria, further experimental details can be found in supplement C.

4.3 Domain Generalization to Medical Image Classification Trustworthy prediction is essential for biomedical data where domain generalization poses a great challenge [Gossmann et al., 2019, 2020]. For example, medical imaging datasets usually come from a multitude of patients and devices where both the patient and devices can form domains. In this study, as suggested by Ilse et al. [2019], we consider hospital as domains, which consist of patients as sub-domains. This correspond to hierarchical domains and has practical implications. Since there can be thousands of patients, and having thousands of domain labels can be impractical and many patients can share common features, e.g. coming from nearby areas, but it can also be true that two hospitals can have similar patients. To simulate such a setting, we construct virtual hospitals by using the Malaria dataset as described in Table 4. The Malaria dataset [Rajaraman et al., 2018] consist of thin blood smear slide images of segmented cells from Malaria patients. We group patients by their IDs for form hospitals. Table 4 shows the out of do-

(a) 1st domain (b) 2nd domain (c) 3rd domain

Figure 4: **Sequential Color-Mnist (VLAG Palette)**. Background color taking 7 hue values spanning the `VLAG` hue ranges sequentially, with fixed saturation and lightning, foreground color takes equally spaced hue value in the complete hue circle with fixed saturation and lightning. Background and foreground colors are zipped. The first 3 color schemes representing 3 sub-domains compose the first nominal domain in Figure 4a, the 2nd nominal domain in Figure 4b takes the middle 3 color schemes with one color scheme overlap with the 1st and one color scheme overlap with 3rd nominal domain in Figure 4c. The 2nd nominal domain serves as a bridge between the other two nominal domains. Out of domain test accuracy is reported in Table 2.

main classification accuracy across different algorithms. Our approach is able to implicitly learn the unobserved domain substructure of the data, resulting is substantially better accuracy on unseen test domains (i.e. a new hospital) compared to state-of-the-art approaches DIVA and MatchDG. The latter require explicit domain labels during training and fail to perform well in scenarios with domain substructure. Our approach also performs substantially better than a standard baseline where information across all domains is pooled together. Additionally, we only use the ELBO in Equation 3.8 as our model selection criteria, without using the validation set.

4.4 Domain Embedding Here, we investigate the ability of our approach to generate meaningful domain embeddings. To this end, we adapt the standard rotated MNIST benchmark [Ilse et al., 2019] by introducing an overlap between three nominal domains: for the first nominal domain, we use 1000 samples of MNIST and rotate them by 15, 30 and 45 degrees respectively. Thus, the first domain contains 3000 instances and each rotation angle constitutes one sub-domain. For the second domain nominal domain, we rotate the same

Table 2: **Out of Domain Accuracy for Color-Mnist (VLAG Palette) in Figure 4**. Each sub-domain in Figure 4 contains a random sample of 1000 mnist images. Random seed is shared for the different sub-domains of a nominal domain but different across nominal domains. Each repetition is with different starting random seed, 10 repetitions are done. The sub-domains are combined to form one nominal domain and 50 percent is used for training, the rest for validation. Comparison algorithms are DIVA [Ilse et al., 2019] and Match-DG [Mahajan et al., 2020], while Deep-All is used as baseline by pooling all training domain s together. The 2nd domain is a bridge domain that connect the 1st and 3rd domain, so it is not used as test domain at all.

Color-Mnist (Figure 4)	**Test Domain 1**	**Test Domain 3**
DIVA	0.63 ± 0.05	0.68 ± 0.03
HDUVA	**0.69** ± 0.05	**0.71** ± 0.03
Deep-All	0.60 ± 0.05	0.68 ± 0.04
Match-DG	0.67 ± 0.06	0.70 ± 0.03

Table 3: **Out of Domain Test Accuracy for Sequential Color-Mnist (Red Diverging Palette) from Figure 5**. Each sub-domain in Figure 5 contains a random sample of 1000 mnist images. Random seed is shared for the different sub-domains of a nominal domain but different across nominal domains. Each repetition is with different starting random seed, 10 repetitions are done. The sub-domains are combined to form one nominal domain and 50 percent is used for training, the rest for validation. Comparison algorithms are DIVA [Ilse et al., 2019] and Match-DG [Mahajan et al., 2020], while Deep-All by pooling all training domains together is used as baseline. The 2nd domain is a bridge domain that connect the 1st and 3rd domain, so it is not used as test domain at all.

Color-Mnist (Figure 5)	**Test Domain 1**	**Test Domain 3**
DIVA	0.53 ± 0.05	0.63 ± 0.05
HDUVA	**0.56** ± 0.05	**0.68** ± 0.05
Deep-All	0.53 ± 0.06	0.61 ± 0.06
Match-DG	0.44 ± 0.04	0.67 ± 0.10

subset of MNIST, by 30, 45 and 60 degrees respectively. In this way, each nominal domains has two rotation degrees of overlap corresponding to 2000 instances that

(a) 1st domain (b) 2nd domain (c) 3rd domain

Figure 5: **Sequential Color-Mnist (Red Diverging Palette)**. Background color taking 7 hue values spanning in the area between 0 and 350 hue degrees (red spectrum) sequentially, with fixed saturation and lightning, foreground color takes equally spaced hue value in the complete hue circle with fixed saturation and lightning. Background and foreground colors are zipped. The first 3 color schemes representing 3 sub-domains compose the first nominal domain in Fig. 5a, the 2nd nominal domain take the middle 3 color schemes with one color scheme overlap with the 1st and another color scheme overlap with the 3rd nominal domain in Fig. 5b. The 2nd nominal domain serves as a bridge between the other two nominal domains. Out of domain test accuracy is reported in Table 3.

have the same rotation. We use these 2 nominal domains for training, and simulate a sequential domain shift for testing with rotation angles of 0, 22 and 75 degrees. We sampled images from both nominal training domains as well the continuously shifted test domain and plot their topic distributions in Figure 6. We expect the topics of the training domains to overlap substantially, due to the shared rotation angles. We further expect for the topics of the test domain to span the entire range of topics from both training domains. Figure 6 illustrates that HDUVA indeed assigns similar domain topics to many instances from both training domains, while samples from the test domain span the entire range of topics.

4.5 State of the art Domain Generalization benchmark We finally compare HDUVA to state-of-the-art domain generalization algorithms for a standard domain generalization task, where domain information is available on largely different domains. Table 5 shows algorithm performance on the PACS dataset [Li et al., 2017] which is a popular domain generalization benchmark.We use AlexNet [Krizhevsky et al., 2012, 2017] as the neural network architecture for $q_{\phi_y}(z_y|x)$

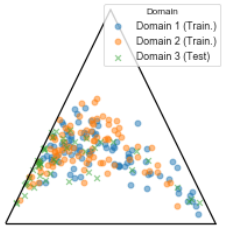


Figure 6: Topic plot of overlapped domains

and $q_\omega(y|z_y)$] in our model in Equation 3.8. For fair comparison, the rest of the algorithms also use AlexNet as classifier.

Table 5 shows that the performance of HDUVA is comparable to state-of-the-art performances on the PACS dataset. Notably, HDUVA without using domain label ties DIVA, and with weak domain supervision variant introduced in Appendix B which we coin WHDUVA, the performance improves over DIVA. With the contrastive pretrain phase of Mahajan et al. [2020] as the initialization for the AlexNet of HDUVA, the performance over the sketch test domain further improves by 3 percent. Deep-All by pooling all training domain together remain a strong baseline where we outperform Deep-All in 3 out of 4 test domains and ties the other one. While overall performance of methods such as JIGSAW is consistently better than HDUVA, it is based on complex image manipulations. In contrast, HDUVA is an interpretable model that can be used for different data modalities and a larger number of tasks including domain prediction and sample generation. Importantly, HDUVA achieves competitive performance without using domain labels during training and without using validation set (we conduct model selection using extended ELBO in Equation 3.8, see supplement C). This enables domain generalization for a much wider range of use-cases than standard algorithms.

5 Conclusion

We proposed an Hierarchical Domain Invariant Variational Autoencoder, with the following improvements:

- Our approach does not require observed domain labels during training, facilitating domain generalization for a much wider range of applications. Additionally, our approach does not need validation set for model selection but only use extended ELBO for model selection.
- In the presence of domain-substructure, our al-

Table 4: **Malaria Virtual Hospital from Malaria Dataset**. Patients with ID starting with C1, C6, C8, C9 are grouped to form 4 virtual hospitals as 4 nominal domains. Virtual hospital C6 has 10 patients with 1061 infected cell images (in total 1748 images). Virtual hospital C8 has 10 patients with 957 infected cell images (in total 1638 images). Virtual hospital C9 has 10 patients with 1284 infected cell images (in total 1964 images). Virtual hospital C1 has 90 patients with 8023 infected cell images (in total 14190 images). Each time, we combine the C6, C8, C9 virtual hospital domain as 3 training domains and sample 20 percent of the images for training. 20 random repetitions are done. We report result on the test domain corresponding to virtual hospital C1. Comparison algorithms are DIVA [Ilse et al., 2019] and Match-DG [Mahajan et al., 2020], while Deep-All is used as baseline by pooling all training domains together.
Data source: [Rajaraman et al., 2018]

Malaria Cell Classification	**Test Accuracy**
DIVA	0.83 ± 0.06
HDUVA	**0.87** ± 0.05
Deep-All	0.84 ± 0.05
MatchDG	0.85 ± 0.09

gorithm is able to robustly disentangle domain-specific variation from class-label specific variation.

- Our algorithm is able to model domain overlap via interpretable topics and generalize to settings with continuous domain shift.
- Our algorithm has a competitive performance even in standard domain generalization tasks, where observed domain information is available on clearly separated domains.
- We proposed evaluation dataset for benchmarking hierarchical and sequential domain shift.

Supplemental Materials

In the supplementary material, to facilitate easy reference, we use consecutive Figure and Table numbering following the main article. In section A we explain an alternative inference algorithm inspired by Ladder-VAE [Sønderby et al., 2016] for our proposed model. In section B, we introduce weak domain supervision methods for both inference algorithms. In section C, we list further details on experimental settings.

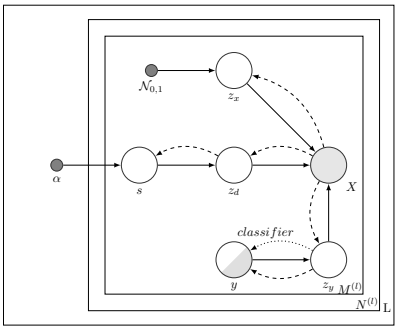


Figure 7: LHDUVA: Ladder Hierarchical Domain Unsupervised Variational Auto-encoding

A Alternative Inference Method for HDUVA

We propose an alternative inference algorithm for our model. The graphical model for the Ladder-VAE version of our model is shown in Figure 7 which we coind LHDUVA. The corresponding variational posterior and ELBO is explained below. We summarize this alternative algorithm in Algorithm 2.

A.1 Inference for LHDUVA In Figure 7, we factorize the approximate posterior as follows:

$$\begin{aligned} &q_\phi(s^{(l,i)}, z_d^{(l,i)}, z_x^{(l,i)}, z_y^{(l,i)}|x^{(l,i)}) \\ =&q_\phi(s|z_d^{(l,i)})q_\phi(z_d^{(l,i)}, z_x^{(l,i)}, z_y^{(l,i)}|x^{(l,i)}) \end{aligned} \tag{A.1}$$

For the approximate posterior distributions of z_x, z_d and z_y, we follow Ilse et al. [2019] and assume fully factorized Gaussians with parameters given as a function of their input:

$$\begin{aligned} &q_\phi(z_d^{(l,i)}, z_x^{(l,i)}, z_y^{(l,i)}|x^{(l,i)}) \\ =&q_{\phi_d}(z_d^{(l,i)}|x^{(l,i)})q_{\phi_x}(z_x^{(l,i)}|x^{(l,i)})q_{\phi_y}(z_y^{(l,i)}|x^{(l,i)}) \end{aligned} \tag{A.2}$$

Encoders q_{ϕ_y}, q_{ϕ_d} and q_{ϕ_x} are parameterized by ϕ_y, ϕ_d and ϕ_x using separate neural networks to model respective means and variances as function of x.

For the form of the approximate posterior distribution of the topic s we chose a Dirichlet distribution:

$$q_{\phi_s}(s^{(l,i)}|z_{d_j}^{(l,i)}) = Dir\left(s^{(l,i)}|\phi_s(z_{d_j}^{(l,i)})\right) \tag{A.3}$$

where ϕ_s parameterizes the concentration parameter based on z_d, using a neural network.

A.2 ELBO for LHDUVA Given the priors and factorization described above, we can optimize the

Table 5: **Domain Generalization in PACS Dataset with AlexNet Classifier**

Methods	Art Painting	Cartoon	Photo	Sketch	Ave.
Factorization [Li et al., 2017]	0.63	0.67	0.90	0.58	0.69
MLDG [Li et al., 2018]	0.66	0.67	0.88	0.59	0.70
SourceCombo [Mancini et al., 2018]	0.64	0.67	0.90	0.60	0.70
MetaReg [Balaji et al., 2018]	**0.70**	0.70	**0.91**	0.59	0.73
GLCM [Wang et al., 2019]	0.67	0.70	0.88	0.56	0.70
Jigsaw [Carlucci et al., 2019]	0.68	**0.72**	0.89	**0.65**	**0.74**
AFLAC [Akuzawa et al., 2018]	0.61	0.64	0.83	0.59	0.67
MatchDG [Mahajan et al., 2020]	0.67 ±0.01	0.69 ± 0.01	0.89 ± 0.01	0.63 ± 0.02	0.72
DIVA [Ilse et al., 2019]	0.64 ± 0.02	0.66 ± 0.003	0.87 ± 0.01	0.58 ± 0.03	0.69
Deep-All	0.64 ± 0.01	0.67 ± 0.02	0.85 ± 0.02	0.56 ± 0.02	0.68
HDUVA	0.65 ± 0.01	0.66 ± 0.01	0.87 ± 0.01	0.58 ± 0.01	0.69
WHDUVA*	0.64 ± 0.01	0.67 ± 0.02	0.88 ± 0.01	0.60 ± 0.02	0.70
LHDUVA**	0.65 ± 0.003	0.69 ± 0.02	0.87 ± 0.004	0.55 ± 0.002	0.69
HDUVA-CTR***	0.65 ± 0.01	0.66 ± 0.02	0.88 ± 0.01	0.63 ± 0.003	0.71

* WHDUVA: weak domain-supervision added to HDUVA as explained in Appendix B.
**LHDUVA: Ladder Hierarchical Domain Unsupervised Variational Auto-encodingexplained in Appendix A.
***HDUVA-CTR: Use the contrastive learning phase (pretrain phase) of Mahajan et al. [2020] as initialization for AlexNet of HDUVA.
Part of the table is adapted from `https://domaingeneralization.github.io/`.

model parameters by maximizing the evidence lower bound (ELBO). We can write the ELBO for a given input-output tuple (x, y) as:

$$\begin{aligned} &ELBO(x,y) = E_{q(z_d|x),q(z_x|x),q(z_y|x)} \log p_\theta(x|z_d, z_x, z_y) \\ &- \beta_x KL(q_{\phi_x}(z_x|x)||p_{\theta_x}(z_x)) - \beta_y KL(q_{\phi_y}(z_y|x)||p_{\theta_y} \\ &(z_y|y)) - \beta_d E_{q_{\phi_s}(s|x,z_d),q_{\phi_d}(z_d|x)} \log \frac{q_{\phi_d}(z_d|x)}{p_{\theta_d}(z_d|s)} \\ &- \beta_s E_{q_{\phi_d}(z_d|x)} KL(q_{\phi_s}(s|z_d)||p_{\theta_s}(s|\alpha)) \end{aligned} \tag{A.4}$$

where we use β to represent the multiplier in the Beta-VAE setting [Higgins et al., 2016], further encouraging disentanglement of the latent representations.

We add an auxiliary classifier $q_\omega(y|z)$, which is parameterized by ω, to encourage separation of classes y in z_y. The LHDUVA objective then becomes:

$$\mathcal{F}(x,y) = ELBO(x,y) + \gamma_y E_{q_{\phi_y}(z_y|x)}[\log q_\omega(y|z_y)] \tag{A.5}$$

To efficiently perform inference with the dependent stochastic variables z_d and s, we follow Sønderby et al. [2016] and adapt the ELBO using the Ladder VAE approach as detailed in the next section.

A.2.1 Dealing with Dependent Stochastic Variables The joint posterior $q(z_d, s|x)$ can be written as:

$$\begin{aligned} q(z_d, s|x) &= \frac{q(z_d, s, x)}{q(x)} = \frac{q(z_d, s, x)}{q(z_d, x)} \frac{q(z_d, x)}{q(x)} \\ &= q(s|z_d, x) q(z_d|x) = q(s|z_d) q(z_d|x) \end{aligned} \tag{A.6}$$

where conditional independence of s from x is assumed. As pointed out by Chen et al. [2016], Tomczak and Welling [2018], this can lead to inactive stochastic units. We follow Sønderby et al. [2016] and recursively correct the generative distribution by a data dependent approximate likelihood. Additionally, we implement a deterministic warm-up period of β following Sønderby et al. [2016], Ilse et al. [2019], in order to prevent the posterior of the latent representation from aligning too quickly to its prior distribution.

B Weak Supervision on domains

In many scenarios only incomplete domain information is available. For example, due to privacy concerns, data from from different customers within a region may be pooled so that information on the nominal domain at customer-level is lost and only higher-level domain information is available. In other settings, substantial heterogeneity may exist in a domain and various unobserved sub-domains may be present. We introduce two techniques for weak supervision on domains, allowing the model to infer such lower-level domains or sub-domain information in the form of a topic s.

B.1 Topic Distribution Aggregation To indicate that a group of samples "weakly" belong to one domain, we aggregate the concentration parameter of the posterior distribution of s for all samples in a minibatch (note that all samples in a minibatch have the same nominal domain):

$$\phi_s^{agg}(z_{d_{1:M}}^{(l,i)}) = 1/M \sum_{j=1:M} \left(\phi_s(z_{d_j}^{(l,i)})\right) \tag{B.7}$$

We then use the aggregated concentration parameter to sample a topic from a Dirichlet distribution:

$$q^{agg}(s^{(l,i)}|z_{d_{1:M}}^{(l,i)}) = Dir\left(\cdot|\phi_s^{agg}(z_{d_{1:M}}^{(l,i)})\right) \tag{B.8}$$

The conditional prior of $z_d^{(l,i)}$ (equation 3.2) then shares this same topic for all samples in the ith mini-batch. We interpret this topic-sharing across samples in a mini-batch as a form of regularized weak supervision. In one-hot encoded approaches, all samples from the same nominal domain would share the same topic. In contrast, sharing a topic in the conditional prior of the latent representation across samples in a mini-batch provides a weak supervision, whilst allowing for an efficient optimisation via SGD. Note that concentration parameters for a mini-batch are only aggregated during training, at test time sample-specific posterior concentration parameters are used.

B.2 Weak domain distribution supervision with MMD DIVA encourages separation of nominal domains in the latent space z_d by fitting an explicit domain classifier which might limit model performance in the case of incomplete domain information. To mitigate these limitations but still weakly encourage separation between different nominal domains, we constrain the HDUVA objective based on the Maximum-Mean-Discrepancy (MMD) [Gretton et al., 2012] between pairwise domains.

Denoting C_{mmd}^d as the minimal distance computed by MMD as an inequality constraint, we can write the constraint optimization of equation A.5 as follows:

$$\begin{aligned} &\underset{\theta,\phi,\omega}{argmax} \sum_{l,i} \mathcal{F}(x^{(l,i)}, y^{(l,i)}) \\ &s.t.\ MMD(q_{z_d}^{(l,i)}(\cdot)|q_{z_d}^{(l^{'},i)}(\cdot)) \geq C_{mmd}^{(l,l^{'})} \end{aligned} \tag{B.9}$$

B.3 Practical considerations In practice, we transform the constrained optimization in Equation B.9 with a Langrange Multiplier. This leads to the final loss in Equation B.10, where $\gamma_d^{(l)}$ denotes the Lagrange multiplier for C_{mmd}^d (c.f. Equation B.9):

$$\begin{aligned} \mathcal{L} = &\sum_{l,i} -\mathcal{F}^{(agg,ladder)}(x^{(l,i)}, y^{(l,i)}) \\ &- \gamma_d^{(l)} \sum_{i,l,l^{'}} MMD(q_{z_d}^{(l,i)}(\cdot)|q_{z_d}^{(l^{'},i)(\cdot)}) \end{aligned} \tag{B.10}$$

Superscript *agg* and *ladder* in Equation B.10 refer to batch-wise aggregation of the concentration parameter and the ladder approach described above.

Algorithm 2 LHDUVA

1: **while** not converged or maximum epochs not reached **do**
2: warm up β defined in Equation A.4, as in [Sønderby et al., 2016]
3: fetch mini-batch {x, y} ={$x^{(l,i)}, y^{(l,i)}$}
4: compute parameters for $q_{\phi_x}(z_x|x)$, $q_{\phi_y}(z_y|x)$, $q_{\phi_d}(z_d|x)$
5: sample latent variable z_x^q, z_y^q and compute $[\log q_\omega(y|z_y)]$ in equation A.5
6: sample z_d^q, infer concentration parameter $\phi_s(z_d)$ and aggregate according to Equation B.7
7: sample topic s from aggregated $\phi_s^{agg}(z_{d_{1:M}})$ according to Equation B.8.
8: compute prior distribution for z_d using s
9: adapt posterior of $q_{\phi_d}(z_d)$ with ladder-vae method [Sønderby et al., 2016]
10: sample z_d^q from adapted $q_{\phi_d}(z_d)$
11: compute $p_\theta(x|z_x, z_y, z_d)$ using sampled z_x^q, z_y^q, z_d^q
12: compute KL divergence for z_d, z_x and z_y, s in Equation A.4
13: compute pair wise MMD of the nominal domains
14: aggregate loss according to B.10 and update model
15: **end while**

C Other experiment details

For comparing algorithms, we implemented DIVA [Ilse et al., 2019] and MatchDG [Mahajan et al., 2020], and use the same hyper-parameters suggested by the original paper. For HDUVA, we match the hyper-parameters in [Ilse et al., 2019], where we take the latent dimension for each latent code is taken to be 64, i.e. $z_x = z_y = z_d = 64$. The classifier is taken to be a one layer neural network with Relu activation. For all experiments, γ_y in equation 3.8 is taken to be $1e5$, while the β values are taken to be 1, warm-up of KL divergence loss in Equation 3.7 is taken to be 100 epochs. We use topic dimension of 3 for HDUVA.

For the malaria experiment, we run with maximum 1000 epochs, with early stopping tolerance of 100 epochs. For HDUVA, we use ELBO directly as model selection criteria, for the rest of the algorithms, we use validation accuracy as model selection criteria. That means, we do not use the validation set at all.

The mnist related experiments are run with maximum 500 epochs with early stopping tolerance of 100 epochs. For HDUVA, we use ELBO directly as model selection criteria, for the rest of the algorithms, we use validation accuracy as model selection criteria. That means, we do not use the validation set at all. We use a learning rate of 1e-4 for DIVA and HDUVA, a learning rate of 1e-5 (better than 1e-4) for Deep-All and the suggested learning rate for MatchDG. For experiments regarding MNIST, including MNIST rotation overlap 4.4, colored mnist combination 4.1, and domain overlapped color-mnist in 4.2, we use random sub-samples (each contains 1000 instances) pre-sampled from `https://github.com/AMLab-Amsterdam/DIVA/tree/master/paper_experiments/rotated_mnist/dataset` with commit hash tag `ab590b4c95b5f667e7b5a7730a797356d124`.

For the PACS experiment, we run with maximum 500 epochs, with early stopping criteria of 5 epochs to save computation resources. For HDUVA, we use ELBO directly as model selection criteria, for the rest of the algorithms, we use validation accuracy as model selection criteria. That means, we do not use the validation set at all. We use a learning rate of 1e-5 for HDUVA, DIVA, Deep-All and use default learning rate of MatchDG.

References

Kei Akuzawa, Yusuke Iwasawa, and Yutaka Matsuo. Domain generalization via invariant representation under domain-class dependency. 2018.

Yogesh Balaji, Swami Sankaranarayanan, and Rama Chellappa. Metareg: Towards domain generalization using meta-regularization. In *Advances in Neural Information Processing Systems*, pages 998–1008, 2018.

Fabio M Carlucci, Antonio D'Innocente, Silvia Bucci, Barbara Caputo, and Tatiana Tommasi. Domain generalization by solving jigsaw puzzles. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2229–2238, 2019.

Xi Chen, Diederik P Kingma, Tim Salimans, Yan Duan, Prafulla Dhariwal, John Schulman, Ilya Sutskever, and Pieter Abbeel. Variational lossy autoencoder. *arXiv preprint arXiv:1611.02731*, 2016.

Alexej Gossmann, Kenny H Cha, and Xudong Sun. Variational inference based assessment of mammographic lesion classification algorithms under distribution shift. In *NeurIPS 2019 workshop: Medical Imaging meets NeurIPS 2019*, 2019.

Alexej Gossmann, Kenny H Cha, and Xudong Sun. Performance deterioration of deep neural networks for lesion classification in mammography due to distribution shift: an analysis based on artificially created distribution shift. In *Medical Imaging 2020: Computer-Aided Diagnosis*, volume 11314, page 1131404. International Society for Optics and Photonics, 2020.

Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *The Journal of Machine Learning Research*, 13(1):723–773, 2012.

Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. 2016.

Haodi Hou, Jing Huo, and Yang Gao. Cross-domain adversarial auto-encoder. *arXiv preprint arXiv:1804.06078*, 2018.

Maximilian Ilse, Jakub M Tomczak, Christos Louizos, and Max Welling. Diva: Domain invariant variational autoencoders. *arXiv preprint arXiv:1905.10427*, 2019.

Weonyoung Joo, Wonsung Lee, Sungrae Park, and Il-Chul Moon. Dirichlet variational autoencoder. *Pattern Recognition*, 107:107514, 2020.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Alexej Klushyn, Nutan Chen, Richard Kurle, Botond Cseke, and Patrick van der Smagt. Learning hierarchical priors in vaes. *arXiv preprint arXiv:1905.04982*, 2019.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60 (6):84–90, 2017.

Da Li, Yongxin Yang, Yi-Zhe Song, and Timothy M Hospedales. Deeper, broader and artier domain generalization. In *Proceedings of the IEEE international conference on computer vision*, pages 5542–5550, 2017.

Da Li, Yongxin Yang, Yi-Zhe Song, and Timothy M Hospedales. Learning to generalize: Meta-learning for domain generalization. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Divyat Mahajan, Shruti Tople, and Amit Sharma. Domain generalization using causal matching. *arXiv preprint arXiv:2006.07500*, 2020.

Massimiliano Mancini, Samuel Rota Bulo, Barbara Caputo, and Elisa Ricci. Best sources forward: domain generalization through source-specific nets. In *2018 25th IEEE International Conference on Image Processing (ICIP)*, pages 1353–1357. IEEE, 2018.

Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. Unrolled generative adversarial networks. *arXiv preprint arXiv:1611.02163*, 2016.

Krikamol Muandet, David Balduzzi, and Bernhard Schölkopf. Domain generalization via invariant feature representation. In *International Conference on Machine Learning*, pages 10–18, 2013.

Sivaramakrishnan Rajaraman, Sameer K Antani, Mahdieh Poostchi, Kamolrat Silamut, Md A Hossain, Richard J Maude, Stefan Jaeger, and George R Thoma. Pre-trained convolutional neural networks as feature extractors toward improved malaria parasite detection in thin blood smear images. *PeerJ*, 6:e4568, 2018.

Danilo Jimenez Rezende and Fabio Viola. Taming vaes. *arXiv preprint arXiv:1810.00597*, 2018.

Casper Kaae Sønderby, Tapani Raiko, Lars Maaløe, Søren Kaae Sønderby, and Ole Winther. Ladder variational autoencoders. In *Advances in neural information processing systems*, pages 3738–3746, 2016.

Akash Srivastava and Charles Sutton. Autoencoding variational inference for topic models. *arXiv preprint arXiv:1703.01488*, 2017.

Jakub Tomczak and Max Welling. Vae with a vampprior. In *International Conference on Artificial Intelligence and Statistics*, pages 1214–1223, 2018.

Riccardo Volpi and Vittorio Murino. Addressing model vulnerability to distributional shifts over image transformation sets. 2019.

Riccardo Volpi, Hongseok Namkoong, Ozan Sener, John C Duchi, Vittorio Murino, and Silvio Savarese. Generalizing to unseen domains via adversarial data augmentation. In *Advances in neural information processing systems*, pages 5334–5344, 2018.

Haohan Wang, Zexue He, Zachary C Lipton, and Eric P Xing. Learning robust representations by projecting superficial statistics out. *arXiv preprint arXiv:1903.06256*, 2019.

Qizhe Xie, Zihang Dai, Yulun Du, Eduard Hovy, and Graham Neubig. Controllable invariance through adversarial feature learning. In *Advances in Neural Information Processing Systems*, pages 585–596, 2017.

Tingting Zhao, Zifeng Wang, Aria Masoomi, and Jennifer G. Dy. Adaptive nonparametric variational autoencoder. *arXiv*, 2019. ISSN 23318422. arXiv: 1906.03288v2.

Appendix D

Maximum Entropy-Regularized Multi-Goal Reinforcement Learning

Contributing Article Zhao, Rui; Sun, Xudong; Tresp, Volker; Maximum Entropy-Regularized Multi-Goal Reinforcement Learning, Proceedings of the 36th International Conference on Machine Learning, 7553–7562, 2019.

Author Contributions Xudong Sun derived the entropy improvement theorem independently and the lower bound theorem with the help of Rui Zhao. The initial idea, algorithm implementation, code development, experiments and results analysis were done by Rui Zhao. The first version of the paper was written by Rui Zhao, both authors discussed on the paper back and forth and elaborated on the final manuscript.

Maximum Entropy-Regularized Multi-Goal Reinforcement Learning

Rui Zhao [1 2] **Xudong Sun** [1] **Volker Tresp** [1 2]

Abstract

In Multi-Goal Reinforcement Learning, an agent learns to achieve multiple goals with a goal-conditioned policy. During learning, the agent first collects the trajectories into a replay buffer, and later these trajectories are selected randomly for replay. However, the achieved goals in the replay buffer are often biased towards the behavior policies. From a Bayesian perspective, when there is no prior knowledge about the target goal distribution, the agent should learn uniformly from diverse achieved goals. Therefore, we first propose a novel multi-goal RL objective based on weighted entropy. This objective encourages the agent to maximize the expected return, as well as to achieve more diverse goals. Secondly, we developed a maximum entropy-based prioritization framework to optimize the proposed objective. For evaluation of this framework, we combine it with Deep Deterministic Policy Gradient, both with or without Hindsight Experience Replay. On a set of multi-goal robotic tasks of OpenAI Gym, we compare our method with other baselines and show promising improvements in both performance and sample-efficiency.

1. Introduction

Reinforcement Learning (RL) (Sutton & Barto, 1998) combined with Deep Learning (DL) (Goodfellow et al., 2016) has led to great successes in various tasks, such as playing video games (Mnih et al., 2015), challenging the World Go Champion (Silver et al., 2016), and learning autonomously to accomplish different robotic tasks (Ng et al., 2006; Peters & Schaal, 2008; Levine et al., 2016; Chebotar et al., 2017; Andrychowicz et al., 2017).

[1]Faculty of Mathematics, Informatics and Statistics, Ludwig Maximilian University of Munich, Munich, Bavaria, Germany [2]Siemens AG, Munich, Bavaria, Germany. Correspondence to: Rui Zhao <zhaorui.in.germany@gmail.com>.

Proceedings of the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019.

One of the biggest challenges in RL is to make the agent learn efficiently in applications with sparse rewards. To tackle this challenge, Lillicrap et al. (2015) developed the Deep Deterministic Policy Gradient (DDPG), which enables the agent to learn continuous control, such as manipulation and locomotion. Schaul et al. (2015a) proposed Universal Value Function Approximators (UVFAs), which generalize not just over states, but also over goals, and extend value functions to multiple goals. Furthermore, to make the agent learn faster in sparse reward settings, Andrychowicz et al. (2017) introduced Hindsight Experience Replay (HER), which encourages the agent to learn from the goal-states it has achieved. The combined use of DDPG and HER allows the agent to learn to accomplish more complex robot manipulation tasks. However, there is still a huge gap between the learning efficiency of humans and RL agents. In most cases, an RL agent needs millions of samples before it is able to solve the tasks, while humans only need a few samples (Mnih et al., 2015).

In previous works, the concept of maximum entropy has been used to encourage exploration during training (Williams & Peng, 1991; Mnih et al., 2015; Wu & Tian, 2016). Recently, Haarnoja et al. (2017) introduced Soft-Q Learning, which learns a deep energy-based policy by evaluating the maximum entropy of actions for each state. Soft-Q Learning encourages the agent to learn all the policies that lead to the optimum (Levine, 2018). Furthermore, Soft Actor-Critic (Haarnoja et al., 2018c) demonstrated a better performance while showing compositional ability and robustness of the maximum entropy policy in locomotion (Haarnoja et al., 2018a) and robot manipulation tasks (Haarnoja et al., 2018b). The agent aims to maximize the expected reward while also maximizing the entropy to succeed at the task while acting as randomly as possible. Based on maximum entropy policies, Eysenbach et al. (2018) showed that the agent is able to develop diverse skills solely by maximizing an information theoretic objective without any reward function. For multi-goal and multi-task learning (Caruana, 1997), the diversity of training sets helps the agent transfer skills to unseen goals and tasks (Pan et al., 2010). The variability of training samples mitigates overfitting and helps the model to better generalize (Goodfellow

This paper is based on our 2018 NeurIPS Deep RL workshop paper (Zhao & Tresp, 2019).

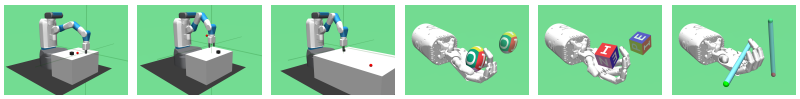


Figure 1. Robot arm Fetch and Shadow Dexterous hand environment: `FetchPush`, `FetchPickAndPlace`, `FetchSlide`, `HandManipulateEgg`, `HandManipulateBlock`, and `HandManipulatePen`.

et al., 2016). In our approach, we combine maximum entropy with multi-goal RL to help the agent to achieve unseen goals by learning uniformly from diverse achieved goals during training.

We observe that during experience replay the uniformly sampled trajectories are biased towards the behavior policies, with respect to the achieved goal-states. Consider training a robot arm to reach a certain point in a space. At the beginning, the agent samples trajectories using a random policy. The sampled trajectories are centered around the initial position of the robot arm. Therefore, the distribution of achieved goals, i.e., positions of the robot arm, is similar to a Gaussian distribution around the initial position, which is non-uniform. Sampling from such a distribution is biased towards the current policies. From a Bayesian point of view (Murphy, 2012), the agent should learn uniformly from these achieved goals, when there is no prior knowledge of the target goal distribution.

To correct this bias, we propose a new objective which combines maximum entropy and the multi-goal RL objective. This new objective uses entropy as a regularizer to encourage the agent to traverse diverse goal-states. Furthermore, we derive a safe lower bound for optimization. To optimize this surrogate objective, we implement maximum entropy-based prioritization as a simple yet effective solution.

2. Preliminary

2.1. Settings

Environments: We consider multi-goal reinforcement learning tasks, like the robotic simulation scenarios provided by OpenAI Gym (Plappert et al., 2018), where six challenging tasks are used for evaluation, including push, slide, pick & place with the robot arm, as well as hand manipulation of the block, egg, and pen, as shown in Figure 1. Accordingly, we define the following terminologies for this specific kind of multi-goal scenarios.

Goals: The goals g are the desired positions and the orientations of the object. Specifically, we use g^e, with e standing for environment, to denote the real goal which serves as the input from the environment, in order to distinguish it from the achieved goal used in Hindsight settings (Andrychowicz et al., 2017). Note that in this paper we consider the case where the goals can be represented by states, which leads us to the concept of achieved goal-state g^s, with details explained below.

States, Goal-States and Achieved Goals: The state s consists of two sub-vectors, the achieved goal-state s^g, which represents the position and orientation of the object being manipulated, and the context state s^c, i.e. $s = (s^g \| s^c)$, where $\|$ denotes concatenation.

In our case, we define $g^s = s^g$ to represent an achieved goal that has the same dimension as the real goal g^e from the environment. The context state s^c contains the rest information about the state, including the linear and angular velocities of all robot joints and of the object. The real goals g^e can be substituted by the achieved goals g^s to facilitate learning. This goal relabeling technique was proposed by Andrychowicz et al. (2017) as Hindsight Experience Replay.

Achieved Goal Trajectory: A trajectory consisting solely of goal-states is represented as $\boldsymbol{\tau}^g$. We use $\boldsymbol{\tau}^g$ to denote all the achieved goals in the trajectory $\boldsymbol{\tau}$, i.e., $\boldsymbol{\tau}^g = (g^s_0, ..., g^s_T)$.

Rewards: We consider sparse rewards r. There is a tolerated range between the desired goal-states and the achieved goal-states. If the object is not in the tolerated range of the real goal, the agent receives a reward signal -1 for each transition; otherwise, the agent receives a reward signal 0.

Goal-Conditioned Policy: In multi-goal settings, the agent receives the environmental goal g^e and the state input $s = (s^g \| s^c)$. We want to train a goal-conditioned policy to effectively generalize its behavior to different environmental goals g^e.

2.2. Reinforcement Learning

We consider an agent interacting with an environment. We assume the environment is fully observable, including a set of state $\mathcal{S}$, a set of action $\mathcal{A}$, a distribution of initial states $p(s_0)$, transition probabilities $p(s_{t+1} \mid s_t, a_t)$, a reward function r: $\mathcal{S} \times \mathcal{A} \to \mathbb{R}$, and a discount factor $\gamma \in [0, 1]$.

Deep Deterministic Policy Gradient: For continuous control tasks, the Deep Deterministic Policy Gradient (DDPG) shows promising performance, which is essentially an off-policy actor-critic method (Lillicrap et al., 2015).

Universal Value Function Approximators: For multi-

goal continuous control tasks, DDPG can be extended by Universal Value Function Approximators (UVFA) (Schaul et al., 2015a). UVFA essentially generalizes the Q-function to multiple goal-states, where the Q-value depends not only on the state-action pairs, but also on the goals.

Hindsight Experience Replay: For robotic tasks, if the goal is challenging and the reward is sparse, the agent could perform badly for a long time before learning anything. Hindsight Experience Replay (HER) encourages the agent to learn from whatever goal-states it has achieved. Andrychowicz et al. (2017) show that HER makes training possible in challenging robotic tasks via goal relabeling, i.e., randomly substituting real goals with achieved goals.

2.3. Weighted Entropy

Guiaşu (1971) proposed weighted entropy, which is an extension of Shannon entropy. The definition of weighted entropy is given as

$$\mathcal{H}_p^w = -\sum_{k=1}^{K} w_k p_k \log p_k, \tag{1}$$

where w_k is the weight of the elementary event and p_k is the probability of the elementary event.

3. Method

In this section, we formally describe our method, including the mathematical derivation of the Maximum Entropy-Regularized Multi-Goal RL objective and the Maximum Entropy-based Prioritization framework.

3.1. Multi-Goal RL

In this paper, we consider multi-goal RL as goal-conditioned policy learning (Schaul et al., 2015a; Andrychowicz et al., 2017; Rauber et al., 2017; Plappert et al., 2018). We denote random variables by upper case letters and the values of random variables by corresponding lower case letters. For example, let $\mathrm{Val}(X)$ denote the set of valid values to a random variable X, and let $p(x)$ denote the probability function of random variable X.

Consider that an agent receives a goal $g^e \in \mathrm{Val}(G^e)$ at the beginning of the episode. The agent interacts with the environment for T timesteps. At each timestep t, the agent observes a state $s_t \in \mathrm{Val}(S_t)$ and performs an action $a_t \in \mathrm{Val}(A_t)$. The agent also receives a reward conditioned on the input goal $r(s_t, g^e) \in \mathbb{R}$.

We use $\boldsymbol{\tau} = s_1, a_1, s_2, a_2, \ldots, s_{T-1}, a_{T-1}, s_T$ to denote a trajectory, where $\boldsymbol{\tau} \in \mathrm{Val}(\boldsymbol{\mathcal{T}})$. We assume that the probability $p(\boldsymbol{\tau} \mid g^e, \boldsymbol{\theta})$ of trajectory $\boldsymbol{\tau}$, given goal g^e and a policy parameterized by $\boldsymbol{\theta} \in \mathrm{Val}(\boldsymbol{\Theta})$, is given as

$$p(\boldsymbol{\tau} \mid g^e, \boldsymbol{\theta}) = p(s_1) \prod_{t=1}^{T-1} p(a_t \mid s_t, g^e, \boldsymbol{\theta}) p(s_{t+1} \mid s_t, a_t).$$

The transition probability $p(s_{t+1} \mid s_t, a_t)$ states that the probability of a state transition given an action is independent of the goal, and we denote it with $S_{t+1} \perp\!\!\!\perp G^e \mid S_t, A_t$. For every $\boldsymbol{\tau}, g^e$, and $\boldsymbol{\theta}$, we also assume that $p(\boldsymbol{\tau} \mid g^e, \boldsymbol{\theta})$ is non-zero. The expected return of a policy parameterized by $\boldsymbol{\theta}$ is given as

$$\begin{aligned} \eta(\boldsymbol{\theta}) &= \mathbb{E}\left[\sum_{t=1}^{T} r(S_t, G^e) \mid \boldsymbol{\theta}\right] \\ &= \sum_{g^e} p(g^e) \sum_{\boldsymbol{\tau}} p(\boldsymbol{\tau} \mid g^e, \boldsymbol{\theta}) \sum_{t=1}^{T} r(s_t, g^e). \end{aligned} \tag{2}$$

Off-policy RL methods use experience replay (Lin, 1992; Mnih et al., 2015) to leverage bias over variance and potentially improve sample-efficiency. In the off-policy case, the objective, Equation (2), is given as

$$\eta^{\mathcal{R}}(\boldsymbol{\theta}) = \sum_{\boldsymbol{\tau},\, g^e} p_{\mathcal{R}}(\boldsymbol{\tau}, g^e \mid \boldsymbol{\theta}) \sum_{t=1}^{T} r(s_t, g^e), \tag{3}$$

where $\mathcal{R}$ denotes the replay buffer. Normally, the trajectories $\boldsymbol{\tau}$ are randomly sampled from the buffer. However, we observe that the trajectories in the replay buffer are often imbalanced with respect to the achieved goals $\boldsymbol{\tau}^g$. Thus, we propose Maximum Entropy-Regularized Multi-Goal RL to improve performance.

3.2. Maximum Entropy-Regularized Multi-Goal RL

In multi-goal RL, we want to encourage the agent to traverse diverse goal-state trajectories, and at the same time, maximize the expected return. This is like maximizing the empowerment (Mohamed & Rezende, 2015) of an agent attempting to achieve multiple goals. We propose the reward-weighted entropy objective for multi-goal RL, which is given as

$$\begin{aligned} \eta^{\mathcal{H}}(\boldsymbol{\theta}) &= \mathcal{H}_p^w(\boldsymbol{\mathcal{T}}^g) \\ &= \mathbb{E}_p\left[\log \frac{1}{p(\boldsymbol{\tau}^g)} \sum_{t=1}^{T} r(S_t, G^e) \mid \boldsymbol{\theta}\right]. \end{aligned} \tag{4}$$

For simplicity, we use $p(\boldsymbol{\tau}^g)$ to represent $\sum_{g^e} p_{\mathcal{R}}(\boldsymbol{\tau}^g, g^e \mid \boldsymbol{\theta})$, which is the occurrence probability of the goal-state trajectory $\boldsymbol{\tau}^g$. The expectation is calculated based on $p(\boldsymbol{\tau}^g)$ as well, so the proposed objective is the weighted entropy (Guiaşu, 1971; Kelbert et al., 2017) of $\boldsymbol{\tau}^g$, which we denote as $\mathcal{H}_p^w(\boldsymbol{\mathcal{T}}^g)$, where the weight w is the accumulated reward $\sum_{t=1}^{T} r(s_t, g^e)$ in our case.

The objective function, Equation (4), has two interpretations. The first interpretation is to maximize the weighted expected return, where the rare trajectories have larger weights. Note that when all trajectories occur uniformly, this weighting mechanism has no effect. The second interpretation is to maximize a reward-weighted entropy, where the more rewarded trajectories have higher weights. This objective encourages the agent to learn how to achieve diverse goal-states, as well as to maximize the expected return.

In Equation (4), the weight, $\log\left(1/p(\boldsymbol{\tau}^g)\right)$, is unbounded, which makes the training of the universal function approximator unstable. Therefore, we propose a safe surrogate objective, $\eta^{\mathcal{L}}$, which is essentially a lower bound of the original objective.

3.3. Surrogate Objective

To construct the safe surrogate objective, we sample the trajectories from the replay buffer with a proposal distribution, $q(\boldsymbol{\tau}^g) = \frac{1}{Z}p(\boldsymbol{\tau}^g)\left(1 - p(\boldsymbol{\tau}^g)\right)$. $p(\boldsymbol{\tau}^g)$ represents the distribution of the goal trajectories in the replay buffer. The surrogate objective is given in Theorem 1, which is proved to be a lower bound of the original objective, Equation (4).

Theorem 1. *The surrogate $\eta^{\mathcal{L}}(\boldsymbol{\theta})$ is a lower bound of the objective function $\eta^{\mathcal{H}}(\boldsymbol{\theta})$, i.e., $\eta^{\mathcal{L}}(\boldsymbol{\theta}) < \eta^{\mathcal{H}}(\boldsymbol{\theta})$, where*

$$\begin{aligned}\eta^{\mathcal{H}}(\boldsymbol{\theta}) &= \mathcal{H}_p^w(\boldsymbol{\mathcal{T}}^g) \\ &= \mathbb{E}_p\left[\log\frac{1}{p(\boldsymbol{\tau}^g)}\sum_{t=1}^{T} r(S_t, G^e) \mid \boldsymbol{\theta}\right]\end{aligned} \tag{5}$$

$$\eta^{\mathcal{L}}(\boldsymbol{\theta}) = Z \cdot \mathbb{E}_q\left[\sum_{t=1}^{T} r(S_t, G^e) \mid \boldsymbol{\theta}\right] \tag{6}$$

$$q(\boldsymbol{\tau}^g) = \frac{1}{Z}p(\boldsymbol{\tau}^g)\left(1 - p(\boldsymbol{\tau}^g)\right) \tag{7}$$

Z is the normalization factor for $q(\boldsymbol{\tau}^g)$. $\mathcal{H}_p^w(\boldsymbol{\mathcal{T}}^g)$ is the weighted entropy (Guiaşu, 1971; Kelbert et al., 2017), where the weight is the accumulated reward $\sum_{t=1}^{T} r(S_t, G^e)$, in our case.

Proof. See Appendix. □

3.4. Prioritized Sampling

To optimize the surrogate objective, Equation (6), we cast the optimization process into a prioritized sampling framework. At each iteration, we first construct the proposal distribution $q(\boldsymbol{\tau}^g)$, which has a higher entropy than $p(\boldsymbol{\tau}^g)$. This ensures that the agent learns from a more diverse goal-state distribution. In Theorem 2, we prove that the entropy with respect to $q(\boldsymbol{\tau}^g)$ is higher than the entropy with respect to $p(\boldsymbol{\tau}^g)$.

Theorem 2. *Let the probability density function of goals in the replay buffer be*

$$p(\boldsymbol{\tau}^g), \textit{where } p(\boldsymbol{\tau}_i^g) \in (0, 1) \textit{ and } \sum_{i=1}^{N} p(\boldsymbol{\tau}_i^g) = 1. \tag{8}$$

Let the proposal probability density function be defined as

$$q(\boldsymbol{\tau}_i^g) = \frac{1}{Z}p(\boldsymbol{\tau}_i^g)\left(1 - p(\boldsymbol{\tau}_i^g)\right), \textit{ where } \sum_{i=1}^{N} q(\boldsymbol{\tau}_i^g) = 1. \tag{9}$$

Then, the proposal goal distribution has an equal or higher entropy

$$\mathcal{H}_q(\boldsymbol{\mathcal{T}}^g) - \mathcal{H}_p(\boldsymbol{\mathcal{T}}^g) \geq 0. \tag{10}$$

Proof. See Appendix. □

3.5. Estimation of Distribution

To optimize the surrogate objective with prioritized sampling, we need to know the probability distribution of a goal-state trajectory $p(\boldsymbol{\tau}^g)$. We use a Latent Variable Model (LVM) (Murphy, 2012) to model the underlying distribution of $p(\boldsymbol{\tau}^g)$, since LVM is suitable for modeling complex distributions.

Specifically, we use $p(\boldsymbol{\tau}^g \mid z_k)$ to denote the latent-variable-conditioned goal-state trajectory distribution, which we assume to be Gaussians. z_k is the k-th latent variable, where $k \in \{1, ..., K\}$ and K is the number of the latent variables. The resulting model is a Mixture of Gaussians(MoG), mathematically,

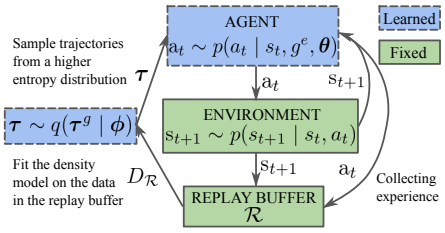
$$p(\boldsymbol{\tau}^g \mid \boldsymbol{\phi}) = \frac{1}{Z}\sum_{i=k}^{K} c_k \mathcal{N}(\boldsymbol{\tau}^g | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k), \tag{11}$$

where each Gaussian, $\mathcal{N}(\boldsymbol{\tau}^g | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$, has its own mean $\boldsymbol{\mu}_k$ and covariance $\boldsymbol{\Sigma}_k$, c_k represents the mixing coefficients, and Z is the partition function. The model parameter $\boldsymbol{\phi}$ includes all mean $\boldsymbol{\mu}_i$, covariance $\boldsymbol{\Sigma}_i$, and mixing coefficients c_k.

In prioritized sampling, we use the complementary predictive density of a goal-state trajectory $\boldsymbol{\tau}^g$ as the priority, which is given as

$$\bar{p}(\boldsymbol{\tau}^g \mid \boldsymbol{\phi}) \propto 1 - p(\boldsymbol{\tau}^g \mid \boldsymbol{\phi}). \tag{12}$$

The complementary density describes the likelihood that a goal-state trajectory $\boldsymbol{\tau}^g$ occurs in the replay buffer. A high complementary density corresponds to a rare occurrence of the goal trajectory. We want to over-sample these rare goal-state trajectories during replay to increase the entropy



Algorithm 1 Maximum Entropy-based Prioritization (MEP)

while *not converged* **do**
  Sample goal $g^e \sim p(g^e)$ and initial state $s_0 \sim p(s_0)$
  for *steps_per_epoch* **do**
    for *steps_per_episode* **do**
      Sample action $a_t \sim p(a_t \mid s_t, g^e, \boldsymbol{\theta})$ from behavior policy.
      Step environment: $s_{t+1} \sim p(s_{t+1} \mid s_t, a_t)$.
      Update replay buffer $\mathcal{R}$.
      Construct prioritized sampling distribution:
      $q(\boldsymbol{\tau}^g) \propto (1 - p(\boldsymbol{\tau}^g \mid \boldsymbol{\phi}))p(\boldsymbol{\tau}^g)$ with higher $\mathcal{H}_q(\boldsymbol{\mathcal{T}}^g)$.
      Sample trajectories $\boldsymbol{\tau} \sim q(\boldsymbol{\tau}^g \mid \boldsymbol{\phi})$
      Update policy $(\boldsymbol{\theta})$ to max. $\mathbb{E}_q\left[r(S, G)\right]$ via DDPG, HER.
    Update density model $(\boldsymbol{\phi})$.

Figure 2. **MEP Algorithm**: We update the density model to construct a higher entropy distribution of achieved goals and update the agent with the more diversified training distribution.

of the training distribution. Therefore, we use the complementary density to construct the proposal distribution as a joint distribution

$$\begin{aligned} q(\boldsymbol{\tau}^g) &\propto \bar{p}(\boldsymbol{\tau}^g \mid \boldsymbol{\phi}) p(\boldsymbol{\tau}^g) \\ &\propto (1 - p(\boldsymbol{\tau}^g \mid \boldsymbol{\phi})) p(\boldsymbol{\tau}^g) \\ &\approx p(\boldsymbol{\tau}^g) - p(\boldsymbol{\tau}^g)^2. \end{aligned} \tag{13}$$

3.6. Maximum Entropy-Based Prioritization

With prioritized sampling, the agent learns to maximize the return of a more diverse goal distribution. When the agent replays the samples, it first ranks all the trajectories with respect to their proposal distribution $q(\boldsymbol{\tau}^g)$, and then uses the ranking number directly as the probability for sampling. This means that rare goals have high ranking numbers and, equivalently, have higher priorities to be replayed. Here, we use the ranking instead of the density. The reason is that the rank-based variant is more robust since it is neither affected by outliers nor by density magnitudes. Furthermore, its heavy-tail property also guarantees that samples will be diverse (Schaul et al., 2015b). Mathematically, the probability of a trajectory to be replayed after the prioritization is:

$$q(\boldsymbol{\tau}_i^g) = \frac{\mathrm{rank}(q(\boldsymbol{\tau}_i^g))}{\sum_{n=1}^{N} \mathrm{rank}(q(\boldsymbol{\tau}_n^g))}, \tag{14}$$

where N is the total number of trajectories in the replay buffer and $\mathrm{rank}(\cdot)$ is the ranking function.

We summarize the complete training algorithm in Algorithm 1 and in Figure 2. In short, we propose Maximum Entropy-Regularized Multi-Goal RL (Section 3.2) to enable RL agents to learn more efficiently in multi-goal tasks (Section 3.1). We integrate a goal entropy term into the normal expected return objective. To maximize the objective, Equation (4), we derive a surrogate objective in Theorem 1, i.e., a lower bound of the original objective. We use prioritized sampling based on a higher entropy proposal distribution at each iteration and utilize off-policy RL methods to maximize the expected return. This framework is implemented as Maximum Entropy-based Prioritization (MEP).

4. Experiments

We test the proposed method on a variety of simulated robotic tasks, see Section 2.1, and compare it to strong baselines, including DDPG and HER. To the best of our knowledge, the most similar method to MEP is Prioritized Experience Replay (PER) (Schaul et al., 2015b). In the experiments, we first compare the performance improvement of MEP and PER. Afterwards, we compare the time-complexity of the two methods. We show that MEP improves performance with much less computational time than PER. Furthermore, the motivations of PER and MEP are different. The former uses TD-errors, while the latter is based on an entropy-regularized objective function.

In this section, we investigate the following questions:

1. Does incorporating goal entropy via MEP bring benefits to off-policy RL algorithms, such as DDPG or DDPG+HER?

2. Does MEP improve sample-efficiency of state-of-the-art RL approaches in robotic manipulation tasks?

3. How does MEP influence the entropy of the achieved goal distribution during training?

Our code is available online at `https://github.com/ruizhaogit/mep.git`. The implementation uses OpenAI Baselines (Dhariwal et al., 2017) with a backend of TensorFlow (Abadi et al., 2016).

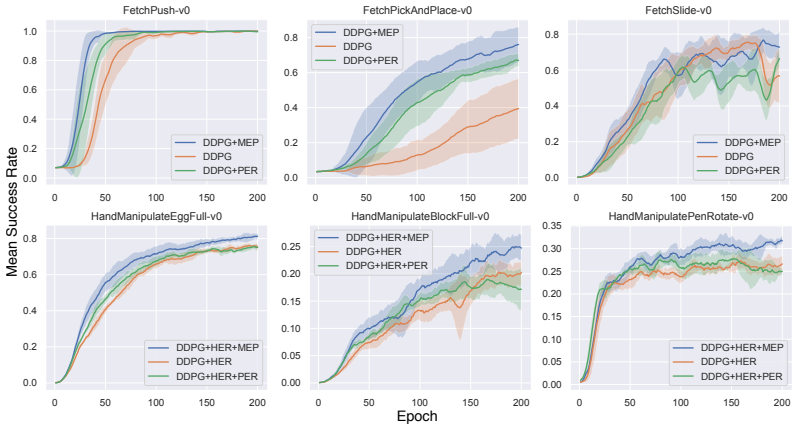


Figure 3. Mean success rate with standard deviation in all six robot environments

Table 1. Mean success rate (%) and training time (hour) for all six environments

	Push		Pick & Place		Slide	
Method	success	time	success	time	success	time
DDPG	99.90%	5.52h	39.34%	5.61h	75.67%	5.47h
DDPG+PER	99.94%	30.66h	67.19%	25.73h	66.33%	25.85h
DDPG+MEP	**99.96%**	6.76h	**76.02%**	6.92h	**76.77%**	6.66h
	Egg		Block		Pen	
Method	success	time	success	time	success	time
DDPG+HER	76.19%	7.33h	20.32%	8.47h	27.28%	7.55h
DDPG+HER+PER	75.46%	79.86h	18.95%	80.72h	27.74%	81.17h
DDPG+HER+MEP	**81.30%**	17.00h	**25.00%**	19.88h	**31.88%**	25.36h

4.1. Performance

To test the performance difference among methods including DDPG, DDPG+PER, and DDPG+MEP, we run the experiment in the three robot arm environments. We use the DDPG as the baseline here because the robot arm environment is relatively simple. In the more challenging robot hand environments, we use DDPG+HER as the baseline method and test the performance among DDPG+HER, DDPG+HER+PER, and DDPG+HER+MEP. To combine PER with HER, we calculate the TD-error of each transition based on the randomly selected achieved goals. Then we prioritize the transitions with higher TD-errors for replay.

Now, we compare the mean success rates. Each experiment is carried out with 5 random seeds and the shaded area represents the standard deviation. The learning curve with respect to training epochs is shown in Figure 3. For all experiments, we use 19 CPUs and train the agent for 200 epochs. After training, we use the best-learned policy for evaluation and test it in the environment. The testing results are the mean success rates. A comparison of the performances along with the training time is shown in Table 1.

From Figure 3, we can see that MEP converges faster in all six tasks than both the baseline and PER. The agent trained with MEP also shows a better performance at the end of the training, as shown in Table 1. In Table 1, we can also see that the training time of MEP lies in between the baseline and PER. It is known that PER can become very time-consuming (Schaul et al., 2015b), especially when the memory size N is very large. The reason is that PER uses TD-errors for prioritization. After each update of the model, the agent needs to update the priorities of the transitions in the replay buffer, which is $O(\log N)$. In our experiments, we use the efficient implementation based on the "sum-tree" data

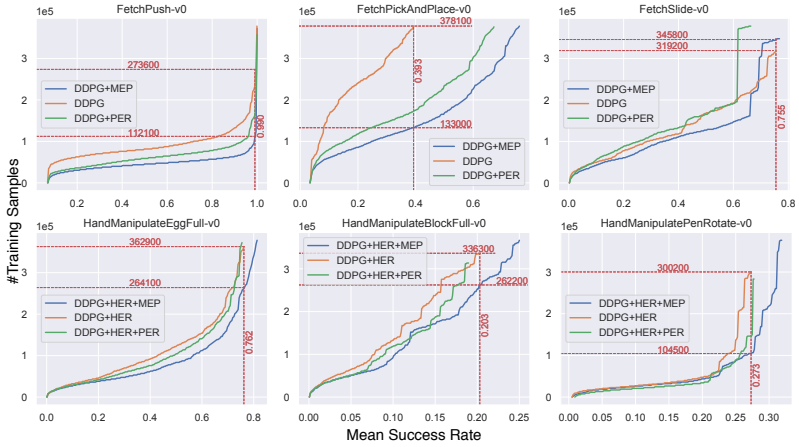


Figure 4. Number of training samples needed with respect to mean success rate for all six environments (the lower the better)

structure, which can be relatively efficiently updated and sampled from (Schaul et al., 2015b). To be more specific, MEP consumes much less computational time than PER. For example in the robot arm environments, on average, DDPG+MEP consumes about 1.2 times the training time of DDPG. In comparison, DDPG+PER consumes about 5 times the training time as DDPG. In this case, MEP is 4 times faster than PER. MEP is faster because it only updates the trajectory density once per epoch and can easily be combined with any multi-goal RL methods, such as DDPG and HER.

Table 1 shows that baseline methods with MEP result in better performance in all six tasks. The improvement increases by up to 39.34 percentage points compared to the baseline methods. The average improvement over the six tasks is 9.15 percentage points. We can see that MEP is a simple, yet effective method and it improves state-of-the-art methods.

4.2. Sample-Efficiency

To compare sample-efficiency of the baseline and MEP, we compare the number of training samples needed for a certain mean success rate. The comparison is shown in Figure 4. From Figure 4, in the `FetchPush-v0` environment, we can see that for the same 99% mean success rate, the baseline DDPG needs 273,600 samples for training, while DDPG+MEP only needs 112,100 samples. In this case, DDPG+MEP is more than twice (2.44) as sample-efficient as DDPG. Similarly, in the other five environments, MEP improves sample-efficiency by factors around one to three. In conclusion, for all six environments, MEP is able to improve sample-efficiency by an average factor of two (1.95) over the baseline's sample-efficiency.

4.3. Goal Entropy

To verify that the overall MEP procedure works as expected, we calculated the entropy value of the achieved goal distribution $\mathcal{H}_p(\mathcal{T}^g)$ with respect to the epoch of training. The experimental results are averaged over 5 different random seeds. Figure 5 shows the mean entropy values with its standard deviation in three different environments. From Figure 5, we can see that the implemented MEP algorithm indeed increases the entropy of the goal distribution. This affirms the consistency of the stated theory with the implemented MEP framework.

5. Related Work

Maximum entropy was used in RL by Williams & Peng (1991) as an additional term in the loss function to encourage exploration and avoid local minimums (Mnih et al., 2016; Wu & Tian, 2016; Nachum et al., 2016; Asadi & Littman, 2016). A similar idea has also been utilized in the deep learning community, where entropy loss was used as a regularization technique to penalize over-confident output distributions (Pereyra et al., 2017). In RL, the entropy loss adds more cost to actions that dominate quickly. A higher entropy loss favors more exploration (Mnih et al., 2016). Neu et al. (2017) gave a unified view on entropy-regularized Markov Decision Processes (MDP) and discussed the convergence properties of entropy-regularized RL, including TRPO (Schulman et al., 2015) and A3C (Mnih et al., 2016).

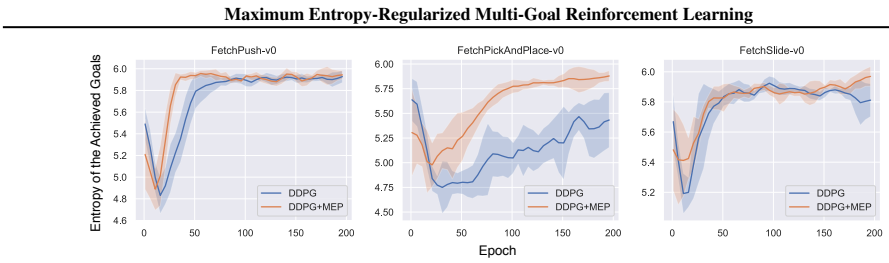


Figure 5. Entropy values of the achieved goal distribution $\mathcal{H}_p(\boldsymbol{\mathcal{T}}^g)$ during training

More recently, Haarnoja et al. (2017) and Levine (2018) proposed deep energy-based policies with state conditioned entropy-based regularization, which is known as Soft-Q Learning. They showed that maximum entropy policies emerge as the solution when optimal control is cast as probabilistic inference. Concurrently, Schulman et al. (2017) showed the connection and the equivalence between Soft-Q Learning and policy gradients. Maximum entropy policies are shown to be robust and lead to better initializations for RL agents (Haarnoja et al., 2018a;b). Based on maximum entropy polices, Eysenbach et al. (2018) developed an information theoretic objective, which enables the agent to automatically discover different sets of skills.

Unlike aforementioned works (Williams & Peng, 1991; Mnih et al., 2016; Haarnoja et al., 2017), the information theoretic objective (Eysenbach et al., 2018) uses state, not actions, to calculate the entropy for distinguishing different skills. Our work is similar to this previous work (Eysenbach et al., 2018) in the sense that we also use the states, instead of actions, to calculate the entropy term and encourage the trained agent to cover a variety of goal-states. Our method generalizes to multi-goal and multi-task RL (Kaelbling, 1993; Sutton et al., 1999; Bakker & Schmidhuber, 2004; Sutton et al., 2011; Szepesvari et al., 2014; Schaul et al., 2015a; Pinto & Gupta, 2017; Plappert et al., 2018).

The entropy term that we used in the multi-goal RL objective is maximized over goal-states. We use maximum goal entropy as a regularization for multi-goal RL, which encourages the agent to learn uniformly with respect to goals instead of experienced transitions. This corrects the bias introduced by the agent's behavior policies. For example, the more easily achievable goals are generally dominant in the replay buffer. The goal entropy-regularized objective allows the agent to learn to achieve the unknown real goals, as well as various virtual goals.

We implemented the maximum entropy regularization via prioritized sampling based on achieved goal-states. We believe that the most similar framework is prioritized experience replay (Schaul et al., 2015b). Prioritized experience replay was introduced by Schaul et al. (2015b) as an improvement to the experience replay in DQN (Mnih et al., 2015). It prioritizes the transitions with higher TD-error in the replay buffer to speed up training. The prioritized experience replay is motivated by TD-errors. However, the motivation of our method comes from information theory–maximum entropy. Compared to prioritized experience replay, our method performs superior empirically and consumes much less computational time.

The intuition behind our method is to assign priority to those under-represented goals, which are relatively more valuable to learn from (see Appendix). Essentially, our method samples goals from an entropy-regularized distribution, rather than from a true replay buffer distribution, which is biased towards the behavior policies. Similar to recent work on goal sampling methods (Forestier et al., 2017; Péré et al., 2018; Florensa et al., 2018; Zhao & Tresp, 2018; Nair et al., 2018; Warde-Farley et al., 2018), our aim is to model a goal-conditioned MDP. In the future, we want to further explore the role of goal entropy in multi-goal RL.

6. Conclusion

This paper makes three contributions. First, we propose the idea of Maximum Entropy-Regularized Multi-Goal RL, which is essentially a reward-weighted entropy objective. Secondly, we derive a safe surrogate objective, i.e., a lower bound of the original objective, to achieve stable optimization. Thirdly, we implement a novel Maximum Entropy-based Prioritization framework for optimizing the surrogate objective. Overall, our approach encourages the agent to achieve a diverse set of goals while maximizing the expected return.

We evaluated our approach in multi-goal robotic simulations. The experimental results showed that our approach improves performance and sample-efficiency of the agent while keeping computational time under control. More precisely, the results showed that our method improves performance by 9 percentage points and sample-efficiency by a factor of two compared to state-of-the-art methods.

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pp. 265–283, 2016.

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, O. P., and Zaremba, W. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pp. 5048–5058, 2017.

Asadi, K. and Littman, M. L. An alternative softmax operator for reinforcement learning. *arXiv preprint arXiv:1612.05628*, 2016.

Bakker, B. and Schmidhuber, J. Hierarchical reinforcement learning based on subgoal discovery and subpolicy specialization. In *Proc. of the 8-th Conf. on Intelligent Autonomous Systems*, pp. 438–445, 2004.

Caruana, R. Multitask learning. *Machine learning*, 28(1): 41–75, 1997.

Chebotar, Y., Kalakrishnan, M., Yahya, A., Li, A., Schaal, S., and Levine, S. Path integral guided policy search. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pp. 3381–3388. IEEE, 2017.

Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., Wu, Y., and Zhokhov, P. Openai baselines. https://github.com/openai/baselines, 2017.

Eysenbach, B., Gupta, A., Ibarz, J., and Levine, S. Diversity is all you need: Learning skills without a reward function. *arXiv preprint arXiv:1802.06070*, 2018.

Florensa, C., Held, D., Geng, X., and Abbeel, P. Automatic goal generation for reinforcement learning agents. In *International Conference on Machine Learning*, pp. 1514–1523, 2018.

Forestier, S., Mollard, Y., and Oudeyer, P.-Y. Intrinsically motivated goal exploration processes with automatic curriculum learning. *arXiv preprint arXiv:1708.02190*, 2017.

Goodfellow, I., Bengio, Y., Courville, A., and Bengio, Y. *Deep learning*, volume 1. MIT press Cambridge, 2016.

Guiaşu, S. Weighted entropy. *Reports on Mathematical Physics*, 2(3):165–179, 1971.

Haarnoja, T., Tang, H., Abbeel, P., and Levine, S. Reinforcement learning with deep energy-based policies. *arXiv preprint arXiv:1702.08165*, 2017.

Haarnoja, T., Hartikainen, K., Abbeel, P., and Levine, S. Latent space policies for hierarchical reinforcement learning. *arXiv preprint arXiv:1804.02808*, 2018a.

Haarnoja, T., Pong, V., Zhou, A., Dalal, M., Abbeel, P., and Levine, S. Composable deep reinforcement learning for robotic manipulation. *arXiv preprint arXiv:1803.06773*, 2018b.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018c.

Kaelbling, L. P. Hierarchical learning in stochastic domains: Preliminary results. In *Proceedings of the tenth international conference on machine learning*, volume 951, pp. 167–173, 1993.

Kelbert, M., Stuhl, I., and Suhov, Y. Weighted entropy: basic inequalities. *Modern Stochastics: Theory and Applications*, 4(3):233–252, 2017. doi: 10.15559/17-VMSTA85. URL www.i-journals.org/vmsta.

Levine, S. Reinforcement learning and control as probabilistic inference: Tutorial and review. *arXiv preprint arXiv:1805.00909*, 2018.

Levine, S., Finn, C., Darrell, T., and Abbeel, P. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Lin, L.-J. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529, 2015.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pp. 1928–1937, 2016.

Mohamed, S. and Rezende, D. J. Variational information maximisation for intrinsically motivated reinforcement learning. In *Advances in neural information processing systems*, pp. 2125–2133, 2015.

Murphy, K. P. Machine learning: A probabilistic perspective. adaptive computation and machine learning, 2012.

Nachum, O., Norouzi, M., and Schuurmans, D. Improving policy gradient by exploring under-appreciated rewards. *arXiv preprint arXiv:1611.09321*, 2016.

Nair, A. V., Pong, V., Dalal, M., Bahl, S., Lin, S., and Levine, S. Visual reinforcement learning with imagined goals. In *Advances in Neural Information Processing Systems*, pp. 9191–9200, 2018.

Neu, G., Jonsson, A., and Gómez, V. A unified view of entropy-regularized markov decision processes. *arXiv preprint arXiv:1705.07798*, 2017.

Ng, A. Y., Coates, A., Diel, M., Ganapathi, V., Schulte, J., Tse, B., Berger, E., and Liang, E. Autonomous inverted helicopter flight via reinforcement learning. In *Experimental Robotics IX*, pp. 363–372. Springer, 2006.

Pan, S. J., Yang, Q., et al. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

Péré, A., Forestier, S., Sigaud, O., and Oudeyer, P.-Y. Unsupervised learning of goal spaces for intrinsically motivated goal exploration. *arXiv preprint arXiv:1803.00781*, 2018.

Pereyra, G., Tucker, G., Chorowski, J., Kaiser, Ł., and Hinton, G. Regularizing neural networks by penalizing confident output distributions. *arXiv preprint arXiv:1701.06548*, 2017.

Peters, J. and Schaal, S. Reinforcement learning of motor skills with policy gradients. *Neural networks*, 21(4): 682–697, 2008.

Pinto, L. and Gupta, A. Learning to push by grasping: Using multiple tasks for effective learning. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pp. 2161–2168. IEEE, 2017.

Plappert, M., Andrychowicz, M., Ray, A., McGrew, B., Baker, B., Powell, G., Schneider, J., Tobin, J., Chociej, M., Welinder, P., et al. Multi-goal reinforcement learning: Challenging robotics environments and request for research. *arXiv preprint arXiv:1802.09464*, 2018.

Rauber, P., Mutz, F., and Schmidhuber, J. Hindsight policy gradients. *arXiv preprint arXiv:1711.06006*, 2017.

Schaul, T., Horgan, D., Gregor, K., and Silver, D. Universal value function approximators. In *International Conference on Machine Learning*, pp. 1312–1320, 2015a.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015b.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. Trust region policy optimization. In *International Conference on Machine Learning*, pp. 1889–1897, 2015.

Schulman, J., Chen, X., and Abbeel, P. Equivalence between policy gradients and soft q-learning. *arXiv preprint arXiv:1704.06440*, 2017.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

Sutton, R. S., Precup, D., and Singh, S. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2): 181–211, 1999.

Sutton, R. S., Modayil, J., Delp, M., Degris, T., Pilarski, P. M., White, A., and Precup, D. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pp. 761–768. International Foundation for Autonomous Agents and Multiagent Systems, 2011.

Szepesvari, C., Sutton, R. S., Modayil, J., Bhatnagar, S., et al. Universal option models. In *Advances in Neural Information Processing Systems*, pp. 990–998, 2014.

Warde-Farley, D., Van de Wiele, T., Kulkarni, T., Ionescu, C., Hansen, S., and Mnih, V. Unsupervised control through non-parametric discriminative rewards. *arXiv preprint arXiv:1811.11359*, 2018.

Williams, R. J. and Peng, J. Function optimization using connectionist reinforcement learning algorithms. *Connection Science*, 3(3):241–268, 1991.

Wu, Y. and Tian, Y. Training agent for first-person shooter game with actor-critic curriculum learning. 2016.

Zhao, R. and Tresp, V. Energy-based hindsight experience prioritization. In *Proceedings of the 2nd Conference on Robot Learning*, pp. 113–122, 2018.

Zhao, R. and Tresp, V. Curiosity-driven experience prioritization via density estimation. *arXiv preprint arXiv:1902.08039*, 2019.

Maximum Entropy-Regularized Multi-Goal Reinforcement Learning (Appendix)

Rui Zhao [1 2] **Xudong Sun** [1] **Volker Tresp** [1 2]

A. Proof of Theorem 1

Theorem 1. *The surrogate $\eta^{\mathcal{L}}(\boldsymbol{\theta})$ is a lower bound of the objective function $\eta^{\mathcal{H}}(\boldsymbol{\theta})$, i.e., $\eta^{\mathcal{L}}(\boldsymbol{\theta}) < \eta^{\mathcal{H}}(\boldsymbol{\theta})$, where*

$$\eta^{\mathcal{H}}(\boldsymbol{\theta}) = \mathcal{H}_p^w(\boldsymbol{\mathcal{T}}^g) = \mathbb{E}_p\left[\log \frac{1}{p(\boldsymbol{\tau}^g)} \sum_{t=1}^{T} r(S_t, G^e) \mid \boldsymbol{\theta}\right] \tag{1}$$

$$\eta^{\mathcal{L}}(\boldsymbol{\theta}) = Z \cdot \mathbb{E}_q\left[\sum_{t=1}^{T} r(S_t, G^e) \mid \boldsymbol{\theta}\right] \tag{2}$$

$$q(\boldsymbol{\tau}^g) = \frac{1}{Z} p(\boldsymbol{\tau}^g)\left(1 - p(\boldsymbol{\tau}^g)\right) \tag{3}$$

Z is the normalization factor for $q(\boldsymbol{\tau}^g)$. $\mathcal{H}_p^w(\boldsymbol{\mathcal{T}}^g)$ is the weighted entropy (Guiaşu, 1971; Kelbert et al., 2017), where the weight is the accumulated reward $\sum_{t=1}^{T} r(S_t, G^e)$ in our case.

Proof.

$$\eta^{\mathcal{L}}(\boldsymbol{\theta}) = Z \cdot \mathbb{E}_q\left[\sum_{t=1}^{T} r(S_t, G^e) \mid \boldsymbol{\theta}\right] \tag{4}$$

$$= \sum_{\boldsymbol{\tau}^g} Z \cdot q(\boldsymbol{\tau}^g) \sum_{t=1}^{T} r(s_t, g^e) \tag{5}$$

$$= \sum_{\boldsymbol{\tau}^g} \frac{Z}{Z} p(\boldsymbol{\tau}^g)(1 - p(\boldsymbol{\tau}^g)) \sum_{t=1}^{T} r(s_t, g^e) \tag{6}$$

$$< \sum_{\boldsymbol{\tau}^g} -p(\boldsymbol{\tau}^g) \log p(\boldsymbol{\tau}^g) \sum_{t=1}^{T} r(s_t, g^e) \tag{7}$$

$$= \mathbb{E}_p\left[\log \frac{1}{p(\boldsymbol{\tau}^g)} \sum_{t=1}^{T} r(S_t, G^e) \mid \boldsymbol{\theta}\right] \tag{8}$$

$$= \mathcal{H}_p^w(\boldsymbol{\mathcal{T}}^g) \tag{9}$$

$$= \eta^{\mathcal{H}}(\boldsymbol{\theta}) \tag{10}$$

In the inequality, we use the property $\log x < x - 1$. □

[1]Faculty of Mathematics, Informatics and Statistics, Ludwig Maximilian University of Munich, Munich, Bavaria, Germany [2]Siemens AG, Munich, Bavaria, Germany. Correspondence to: Rui Zhao <zhaorui.in.germany@gmail.com>.

Proceedings of the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019.

B. Proof of Theorem 2

Theorem 2. *Let the probability density function of goals in the replay buffer be*

$$p(\boldsymbol{\tau}^g), \textit{where } p(\boldsymbol{\tau}_i^g) \in (0,1) \textit{ and } \sum_{i=1}^{N} p(\boldsymbol{\tau}_i^g) = 1. \tag{11}$$

Let the proposal probability density function be defined as

$$q(\boldsymbol{\tau}_i^g) = \frac{1}{Z} p(\boldsymbol{\tau}_i^g)\left(1 - p(\boldsymbol{\tau}_i^g)\right), \textit{ where } \sum_{i=1}^{N} q(\boldsymbol{\tau}_i^g) = 1. \tag{12}$$

Then, the proposal goal distribution has an equal or higher entropy

$$\mathcal{H}_q(\boldsymbol{\mathcal{T}}^g) - \mathcal{H}_p(\boldsymbol{\mathcal{T}}^g) \geq 0. \tag{13}$$

Proof. For clarity, we define the notations in this proof as $p_i = p(\boldsymbol{\tau}_i^g)$ and $q_i = q(\boldsymbol{\tau}_i^g)$.

Note that the definition of Entropy is

$$\mathcal{H}_p = \sum_i -p_i \log(p_i), \tag{14}$$

where the ith summand is $p_i \log(p_i)$, which is a concave function. Since the goal distribution has a finite support I, we have the real-valued vector $(p_1, \ldots, p_N)$ and $(\frac{1}{Z}q_1, \ldots, \frac{1}{Z}q_N)$.

We use Karamata's inequality (Kadelburg et al., 2005), which states that if the vector $(p_1, \ldots, p_N)$ majorizes $(\frac{1}{Z}q_1, \ldots, \frac{1}{Z}q_N)$ then the summation of the concave transformation of the first vector is smaller than the concave transformation of the second vector.

In our case, the concave transformation is the weighted information at the ith position -$p_i \log(p_i)$, where the weight is the probability p_i (entropy is the expectation of information). Therefore, the proof of the theorem is also a proof of the majorizing property of p over q (Petrov).

We denote the proposal goal distribution as

$$q_i = f(p_i) = \frac{1}{Z} p_i (1 - p_i). \tag{15}$$

Note that in our case, the partition function Z is a constant.

Majorizing has three requirements (Marshall et al., 1979).

The first requirement is that both vectors must sum up to one. This requirement is already met because

$$\sum_i p_i = \sum_i q_i = 1. \tag{16}$$

The second requirement is that monotonicity exits. Without loss of generality, we assume the probabilities are sorted:

$$p_1 \geq p_2 \geq \ldots \geq p_N \tag{17}$$

Thus, if $i > j$ then

$$f(p_i) - f(p_j) = \frac{1}{Z} p_i (1 - p_i) - \frac{1}{Z} p_j (1 - p_j) \tag{18}$$

$$= \frac{1}{Z}[(p_i - p_j) - (p_i + p_j)(p_i - p_j)] \tag{19}$$

$$= \frac{1}{Z}(p_i - p_j)(1 - p_i - p_j) \tag{20}$$

$$\geq 0. \tag{21}$$

which means that if the original goal probabilities are sorted, the transformed goal probabilities are also sorted,

$$f(p_1) \geq f(p_2) \geq \ldots \geq f(p_N). \tag{22}$$

The third requirement is that for an arbitrary cutoff index k, there is

$$p_1 + \ldots p_k < q_1 + \ldots + q_k. \tag{23}$$

To prove this, we have

$$\begin{aligned} p_1 + \ldots + p_k &= \frac{p_1 + \ldots + p_k}{1} && (24)\\ &= \frac{p_1 + \ldots + p_k}{p_1 + \ldots + p_N} && (25)\\ &\geq f(p_1) + ... + f(p_k) && (26)\\ &= \frac{1}{Z}[p_1(1-p_1) + ... + p_k(1-p_k)] && (27)\\ &= \frac{1}{Z}[p_1 + \ldots + p_k - (p_1^2 + \ldots + p_k^2)] && (28) \end{aligned}$$

Note that, we multiply $Z * 1$ to each side of

$$Z = p_1(1-p_1) + \ldots + p_N(1-p_N). \tag{29}$$

Then we have

$$(p_1 + \ldots + p_k)Z * 1 \geq p_1 + \ldots + p_k - (p_1^2 + \ldots + p_k^2) * 1. \tag{30}$$

Now, we substitute the expression of Z and then have

$$(p_1 + \ldots + p_k)[p_1(1-p_1) + \ldots + p_N(1-p_N)] \geq [p_1 + \ldots + p_k - (p_1^2 + \ldots + p_k^2)] * 1. \tag{31}$$

We express 1 as a series of terms $\sum_i p_i$, we have

$$(p_1 + \ldots + p_k)[p_1(1-p_1) + \ldots + p_N(1-p_N)] \geq [p_1 + \ldots + p_k - (p_1^2 + \ldots + p_k^2)] * [(p_1 + \ldots + p_k) + (p_{k+1} + \ldots p_N)]. \tag{32}$$

We use the distributive law to the right side and have

$$\begin{aligned} &(p_1 + \ldots + p_k)[p_1(1-p_1) + \ldots + p_N(1-p_N)]\\ \geq &[p_1 + \ldots + p_k] * [(p_1 + \ldots + p_k) + (p_{k+1} + \ldots p_N)] - [(p_1^2 + \ldots + p_k^2)] * [(p_1 + \ldots + p_k) + (p_{k+1} + \ldots p_N)]. \end{aligned} \tag{33}$$

We move the first term on the right side to the left and use the distributive law then have

$$(p_1 + \ldots + p_k)[-1 * (p_1^2 + \ldots + p_N^2))] \geq -[(p_1^2 + \ldots + p_k^2)] * [(p_1 + \ldots + p_k) + (p_{k+1} + \ldots p_N)]. \tag{34}$$

We use the distributive law again on the right side and move the first term to the left and use the distributive law then have

$$(p_1 + \ldots + p_k)[-1 * (p_{k+1}^2 + \ldots + p_N^2))] \geq -[(p_1^2 + \ldots + p_k^2)] * [(p_{k+1} + \ldots p_N)]. \tag{35}$$

We remove the minus sign then have

$$(p_1 + \ldots + p_k)[(p_{k+1}^2 + \ldots + p_N^2))] \leq [(p_1^2 + \ldots + p_k^2)] * [(p_{k+1} + \ldots p_N)]. \tag{36}$$

To prove the inequality above, it suffices to show that the inequality holds true for each associated term of the multiplication on each side of the inequality.

Suppose that

$$i \leq k < j \tag{37}$$

then we have

$$p_i > p_j. \tag{38}$$

As mentioned above, the probabilities are sorted in descending order. We have

$$p_i p_j^2 - p_i^2 p_j = p_i p_j (p_j - p_i) < 0 \tag{39}$$

then

$$p_i p_j^2 < p_i^2 p_j. \tag{40}$$

Therefore, we have proved that the inequality holds true for an arbitrary associated term, which also applies when they are added up. □

C. Insights

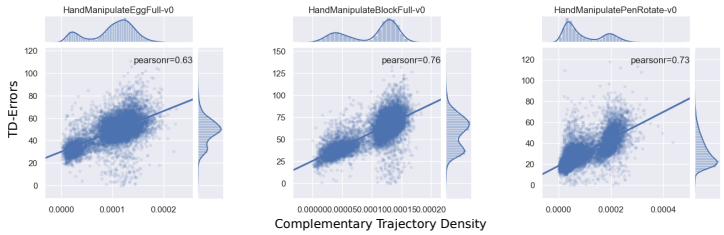


Figure 1. Pearson correlation between the complementary density $\bar{p}(\boldsymbol{\tau}^g)$ and TD-errors in the middle of training

To further understand why maximum entropy in goal space facilitates learning, we look into the TD-errors during training. We investigate the correlation between the complementary predictive density $\bar{p}(\boldsymbol{\tau}^g \mid \boldsymbol{\phi})$ and the TD-errors of the trajectory. The Pearson correlation coefficients, i.e., Pearson's r (Benesty et al., 2009), between the density $\bar{p}(\boldsymbol{\tau}^g \mid \boldsymbol{\phi})$ and the TD-errors of the trajectory are 0.63, 0.76, and 0.73, for the hand manipulation of egg, block, and pen tasks, respectively. The plot of the Pearson correlation is shown in Figure 1. The value of Pearson's r is between 1 and -1, where 1 is total positive linear correlation, 0 is no linear correlation, and -1 is total negative linear correlation. We can see that the complementary predictive density is correlated with the TD-errors of the trajectory with an average Pearson's r of 0.7. This proves that the agent learns faster from a more diverse goal distribution. Under-represented goals often have higher TD-errors, and thus are relatively more valuable to learn from. Therefore, it is helpful to maximize the goal entropy and prioritize the under-represented goals during training.

References

Benesty, J., Chen, J., Huang, Y., and Cohen, I. Pearson correlation coefficient. In *Noise reduction in speech processing*, pp. 1–4. Springer, 2009.

Guiaşu, S. Weighted entropy. *Reports on Mathematical Physics*, 2(3):165–179, 1971.

Kadelburg, Z., Dukic, D., Lukic, M., and Matic, I. Inequalities of karamata, schur and muirhead, and some applications. *The Teaching of Mathematics*, 8(1):31–45, 2005.

Kelbert, M., Stuhl, I., and Suhov, Y. Weighted entropy: basic inequalities. *Modern Stochastics: Theory and Applications*, 4 (3):233–252, 2017. doi: 10.15559/17-VMSTA85. URL www.i-journals.org/vmsta.

Marshall, A. W., Olkin, I., and Arnold, B. C. *Inequalities: theory of majorization and its applications*, volume 143. Springer, 1979.

Petrov, F. Shannon entropy of $p(x)(1 - p(x))$ is no less than entropy of $p(x)$. MathOverflow. URL https://mathoverflow.net/q/320726. URL:https://mathoverflow.net/q/320726 (version: 2019-01-12).

Appendix E

Tutorial and Survey on Probabilistic Graphical Model and Variational Inference in Deep Reinforcement Learning

Contributing Article Sun, Xudong; Bischl, Bernd; Tutorial and Survey on Probabilistic Graphical Model and Variational Inference in Deep Reinforcement Learning, 2019 IEEE Symposium Series on Computational Intelligence (SSCI), 1908.09381, 2019.

Tutorial and Survey on Probabilistic Graphical Model and Variational Inference in Deep Reinforcement Learning

Xudong Sun
Department of Statistics
Ludwig Maximillian University of Munich
Munich, Germany
Email: xudong.sun@stat.uni-muenchen.de

Bernd Bischl
Department of Statistics
Ludwig Maximillian University of Munich
Munich, Germany

***Abstract*—Aiming at a comprehensive and concise tutorial survey, recap of variational inference and reinforcement learning with Probabilistic Graphical Models are given with detailed derivations. Reviews and comparisons on recent advances in deep reinforcement learning are made from various aspects. We offer detailed derivations to a taxonomy of Probabilistic Graphical Model and Variational Inference methods in deep reinforcement learning, which serves as a complementary material on top of the original contributions.**

***Keywords*—Probabilistic Graphical Models; Variational Inference; Deep Reinforcement Learning**

I. Introduction

Despite the recent successes of Reinforcement Learning, powered by Deep Neural Networks, in complicated tasks like games [1] and robot locomotion [2], as well as optimization tasks like Automatic Machine Learning [3]. The field still faces many challenges including expressing high dimensional state and policy, exploration in sparse reward, etc. Probabilistic Graphical Model and Variational Inference offers a great tool to express a wide spectrum of trajectory distributions as well as conducting inference which can serve as a control method. Due to the emerging popularity, we present a comprehensive and concise tutorial survey paper with the following contributions:

- We provide Probabilistic Graphical Models for many basic concepts of Reinforcement Learning, which is rarely covered in literature. We also provide Probabilistic Graphical Models to some recent works on Deep Reinforcement Learning [4], [5] which does not exist in the original contributions.
- We cover a taxonomy of Probabilistic Graphical Model and Variational Inference [6] methods used in Deep Reinforcement Learning and give detailed derivations to many of the critical equations, which is not given in the original contributions. Together with the recap of variational inference and deep reinforcement learning, the paper serves as a self-inclusive tutorial to both beginner and advanced readers.

A. Organization of the paper

In section I-B, we first introduce the fundamentals of Probabilistic Graphical Models and Variational Inference, then we review the basics about reinforcement learning by connecting probabilistic graphical models (PGM) in section II-A, II-B, II-C, as well as an overview about deep reinforcement learning, accompanied with a comparison of different methods in section II-D. In section III-A, we discuss how undirected graph could be used in modeling both the value function and the policy, which works well on high dimensional discrete state and action spaces. In section III-B, we introduce the directed acyclic graph framework on how to treat the policy as posterior on actions, while adding many proofs that does not exist in the original contributions. In section III-C, we introduce works on how to use variational inference to approximate the environment model, while adding graphical models and proofs which does not exist in the original contributions.

B. Prerequisite on Probabilistic Graphical Models and Variational Inference, Terminologies and Conventions

We use capital letter to denote a Random Variable (RV), while using the lower case letter to represent the realization. To avoid symbol collision of using A to represent advantage in many RL literature, we use A^{act} explicitly to represent action. We use $(B \perp\!\!\!\perp C) \mid A$ to represent B is conditionally independent from C, given A, or equivalently $p(B|A, C) = p(B|A)$ or $p(BC|A) = P(B|A)P(C|A)$. Directed Acyclic Graphs (DAG) [7] as a PGM offers an instinctive way of defining factorized joint distributions of RV by assuming the conditional independence [7] through d-separation [7]. Undirected Graph including Markov Random Fields also specifies the conditional independence with local Markov property and global Markov property [8].

Variational Inference (VI) approximates intractable posterior distribution $p(z \mid x) = \frac{1}{\int_{z'} p(z')p(x|z')dz'} p(z)p(x \mid z)$ with latent variable z specified in a probabilistic graphical model, by a variational proposal posterior distribution $q_\phi(z \mid x)$, characterized by variational parameter ϕ. By optimizing the

$$
\begin{aligned}
&\log p(x) \\
=&E_q\left[\log p(x \mid z)\right] - E_q\left[\log \frac{q_\phi(z \mid x)}{p(z)}\right] + E_q\left[\log \frac{q_\phi(z \mid x)}{p(z \mid x)}\right] \\
=&-D_{KL}(q_\phi(z|x)||p(x,z)) - E_q\left[\log p(z|x)\right] + \\
&E_q \log q_\phi(z|x) \\
=&-F(\phi,\theta) + H_q(p) - H(q) \\
=&ELBO(\phi,\theta) + D_{KL}(q_\phi(z \mid x)||p(z \mid x)) \qquad (1)
\end{aligned}
$$

Evidence Lower Bound (ELBO) [6], VI assigns the values to observed and latent variables at the same time. VI is widely used in Deep Learning Community like variational resampling [9]. VI is also used in approximating the posterior on the weights distribution of neural networks for Thompson Sampling to tackle the exploration-exploitation trade off in bandit problems [10], as well as approximating on the activations distribution like Variational AutoEncoder [11].

As a contribution of this paper, we summarize the relationship of evidence $\log p(x)$, KL divergence D_{KL}, cross entropy $H_q(p)$, entropy $H(q)$, free energy $F(\phi,\theta)$ and $ELBO(\phi,\theta)$ in Equation (1).

II. Reinforcement Learning and Deep Reinforcement Learning

A. Basics about Reinforcement Learning with graphical model

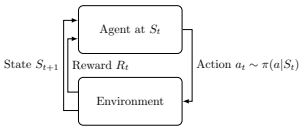


Fig. 1. Concept of Reinforcement Learning

1) RL Concepts, Terminology and Convention: As shown in Figure 1, Reinforcement Learning (RL) involves optimizing the behavior of an agent via interaction with the environment. At time t, the agent lives on state S_t, By executing an action a_t according to a policy [12] $\pi(a|S_t)$, the agent jumps to another state S_{t+1}, while receiving a reward R_t. Let discount factor γ decides how much the immediate reward is favored compared to longer term return, with which one could also allow tractability in infinite horizon reinforcement learning [12], as well as reducing variance in Monte Carlo setting [13]. The goal is to maximize the accumulated rewards, $G = \sum_{t=0}^{T} \gamma^t R_t$ which is usually termed return in RL literature.

For simplicity, we interchangeably use two conventions whenever convenient: Suppose an episode last from $t = 0 : T$, with $T \to \infty$ correspond to continuous non-episodic reinforcement learning. We use another convention of $t \in \{0, \cdots, \infty\}$ by assuming when episode ends, the agent stays at a self absorbing state with a null action, while receiving null reward.

By unrolling Figure 1, we get a sequence of state, action and reward tuples $\{(S_t, A_t^{act}, R_t)\}$ in an episode, which is coined trajectory τ [14]. Figure 2 illustrates part of a trajectory in one rollout. The state space $\mathcal{S}$ and action space $\mathcal{A}$, which can be either discrete or continuous and multi-dimensional, are each represented with one continuous dimension in Figure 2 and plotted in an orthogonal way with different colors, while we use the thickness of the plate to represent the reward space $\mathcal{R}$.

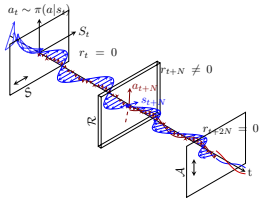


Fig. 2. Illustration of State, Action and Reward Trajectory

2) DAGs for (Partially Observed) Markov Decision Process: Reinforcement Learning is a stochastic decision process, which usually comes with three folds of uncertainty. That is, under a particular stochastic policy characterized by $\pi(a|s) = p(a|s)$, within a particular environment characterized by state transition probability $p(s_{t+1}|s_t, a)$ and reward distribution function $p(r_t|s_t, a_t)$, a learning agent could observe different trajectories with different unrolling realizations. This is usually modeled as a Markov Decision Process [12], with its graphical model shown in Figure 3, where we could define a joint probability distribution over the trajectory of state , action and reward RVs. In Figure 3, we use dashed arrows connecting state and action to represent the policy, upon fixed policy π, we have the trajectory likelihood in Equation (2)

$$p(\tau) = p(s_0) \prod_{t=0}^{T} p(s_{t+1}|s_t, a_t) p(r_t|s_t, a_t) \pi(a_t|s_t) \qquad (2)$$

Upon observation of a state s_t in Figure 3, the action at the time step in question is conditionally independent with the state and action history $\mathcal{E}_t = \{S_0, A_0^{act}, \cdots, S_{t-1}\}$, which could be denoted as $(A_t^{act} \perp\!\!\!\perp \mathcal{E}_t) \mid S_t$. A more realistic model,

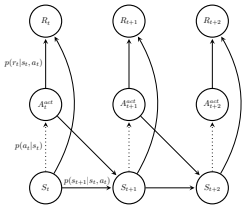


Fig. 3. Directed Acyclic Graph For Markov Decision Process

however, is the Partially Observable Markov Decision process [15], with its DAG representation shown in Figure 4, where the agent could only observe the state partially by observing O_t through a non invertible function of the next state S_{t+1} and the action a_t, as indicated the Figure by $p(o_t|s_{t+1}, a_t)$, while

the distributions on other edges are omitted since they are the same as in Figure 3. Under the graph specification of Figure 4, the observable O_t is no longer Markov, but depends on the whole history, however, the latent state S_t is still Markov. For POMDP, belief state b_t is defined at time t, which is associated with a probability distribution $b_t(s_t)$ over the hidden state S_t, with $\sum_{\mathcal{S}} b(S_t) = 1$, where state S takes value in latent state space $\mathcal{S}$ [15]. The latent state distribution associated with

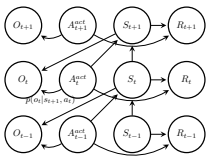


Fig. 4. Probabilistic Graphical Model for POMDP

belief state can be updated in a Bayesian way in Equation (6).

B. Value Function, Bellman Equation, Policy Iteration

Define state value function of state $s \in \mathcal{S}$ in Equation (7), where the corresponding Bellman Equation is derived in Equation (8).

$$V^{\pi}(s) = E_{\pi,\varepsilon}[\sum_{i=0}^{\infty} \gamma^i R_{t+i}(S_{t+i}, A_{t+i}^{act}) \mid \forall S_t = s] \tag{7}$$

$$\begin{aligned}
&= E_{\pi,\varepsilon}[R_t(S_t, A_t^{act}) + \gamma \sum_{i=1}^{\infty} \gamma^{(i-1)} R_{t+i}(S_{t+i}, A_{t+i}^{act})] \\
&= E_{\pi,\varepsilon}[R_t(S_t, A_t^{act}) + \gamma \sum_{i'=0}^{\infty} \gamma^{i'} R_{t+1+i'}(S_{t+1+i'}, A_{t+1+i'}^{act})] \\
&= E_{\pi,\varepsilon}[R_t(S_t, A_t^{act}) + \gamma V^{\pi}(S_{t+1})]
\end{aligned} \tag{8}$$

where $S_{t+i} \sim p(s_{t+i+1}|s_{t+i}, a_{t+i})$ takes value from $\mathcal{S}$, $A_{t+i}^{act} \sim \pi(a|S_{t+i+1})$ taking value from $\mathcal{A}$, and we have used the π and ε in the subscript of the expectation E operation to represent the probability distribution of the policy and the environment (including transition probability and reward probability) respectively. State action value function [12] is defined in Equation (9), where in Equation (10), its relationship to the state value function is stated.

$$\begin{aligned}
&Q^{\pi}(s,a)\,(\forall S_t = s, A_t^{act} = a) \\
&= E_{\pi,\varepsilon}[R_t(S_t = s, A_t^{act} = a) + \sum_{i=1}^{\infty} \gamma^i R_{t+i}(S_{t+i}, A_{t+i}^{act})] \quad (9) \\
&= E_{\pi,\varepsilon}[R_t(S_t = s, A_t^{act} = a) + \gamma V^{\pi}(S_{t+1})] \quad (10)
\end{aligned}$$

Combining Equation (8) and Equation (9), we have

$$V(s) = \sum_a \pi(a|s) Q(s,a) \tag{11}$$

Define optimal policy [12] to be

$$\begin{aligned}
\pi^* &= arg\max_{\pi} V^{\pi}(s), \forall s \in S \\
&= arg\max_{\pi} E_{\pi}[R_t + \gamma V^{\pi}(S_{t+1})]
\end{aligned} \tag{12}$$

Taking the optimal policy π^* into the Bellman Equation in Equation (8), we have

$$V^{\pi^*}(s) = E_{\pi^*,\varepsilon}\left[R_t(s, A_t^{act}) + \gamma V^{\pi^*}(S_{t+1})\right] \tag{13}$$

Taking the optimal policy π^* into Equation (9), we have

$$Q^{\pi^*}(s,a) = E_{\pi^*,\varepsilon}[R_t(s,a) + \sum_{i=1}^{\infty} \gamma^i R_{t+i}(S_{t+i}, A_{t+i}^{act})] \tag{14}$$

Based on Equation (14) and Equation (13), we get

$$V^{\pi^*}(s) = \max_a Q^{\pi^*}(s,a) \tag{15}$$

and

$$Q^{\pi^*}(s,a) = E_{\varepsilon,\pi^*}\left[R_t(s,a) + \gamma \max_{\bar{a}} Q^{\pi^*}(S_{t+1}, \bar{a})\right] \tag{16}$$

For learning the optimal policy and value function, General Policy Iteration [12] can be conducted, as shown in Figure 5, where a contracting process [12] is drawn. Starting from initial policy π_0, the corresponding value function V^{π_0} could be estimated, which could result in improved policy π_1 by greedy maximization over actions. The contracting process is supposed to converge to the optimal policy π^*.

As theoretically fundamentals of learning algorithms, Dynamic programming and Monte Carlo learning serve as two extremities of complete knowledge of environment and complete model free [12], while time difference learning [12] is more ubiquitously used, like a bridge connecting the two extremities. Time difference learning is based on the Bellman update error $\delta_t = Q(s_t, a_t) - \left(R_t(s,a) + \gamma \max_{\bar{a}} Q(s_{t+1}, a)\right)$.

C. Policy Gradient and Actor Critic

Reinforcement Learning could be viewed as a functional optimization process. We could define an objective function over a policy $\pi_\theta(a|s)$, as a functional, characterized by parameter θ, which could correspond to the neural network weights, for example.

$$\begin{aligned}
&b_{t+1}(s_{t+1}) \\
&= p(s_{t+1} \mid o_t, a_t, b_t) \\
&= \frac{p(s_{t+1}, o_t, a_t, b_t)}{p(o_t, a_t, b_t)} \frac{p(s_{t+1}, a_t, b_t)}{p(s_{t+1}, a_t, b_t)} \\
&= p(o_t \mid a_t, s_{t+1}, b_t) \frac{p(s_{t+1} \mid a_t, b_t)}{p(o_t \mid a_t, b_t)} \quad (3) \\
&= p(o_t \mid s_{t+1}, a_t) \frac{\sum_{s_t} p(s_t, s_{t+1} \mid a_t, b_t)}{p(o_t \mid a_t, b_t)} \quad (4) \\
&= p(o_t \mid s_{t+1}, a_t) \frac{\sum_{s_t} p(s_{t+1} \mid s_t, a_t, b_t) p(s_t \mid a_t, b_t)}{p(o_t \mid a_t, b_t)} \quad (5) \\
&= p(o_t \mid s_{t+1}, a_t) \frac{\sum_{s_t} p(s_{t+1} \mid s_t, a_t) p(s_t \mid a_t, b_t)}{p(o_t \mid a_t, b_t)} \quad (6)
\end{aligned}$$

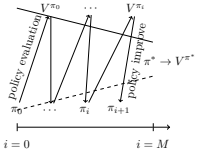


Fig. 5. General Policy Iteration

$$\eta(s) = \sum_{k=0} \gamma^k P^{\pi}(s_0 \to s, k+1) \tag{17}$$

$$= h(s) + \sum_{\bar{s},a} \gamma \eta(\bar{s}) \pi_\theta(a|\bar{s}) P(s|\bar{s}, a) \tag{18}$$

Suppose all episodes start from an auxiliary initial state s_0, which with probability $h(s)$, jumps to different state $s \in \mathcal{S}$ without reward. $h(s)$ characterizes the initial state distribution which only depends on the environment. Let $\eta(s)$ represent the expected number of steps spent on state s, which can be calculated by summing up the γ discounted probability $P^{\pi}(s_0 \to s, k+1)$ of entering state s with $k+1$ steps from auxiliary state s_0, as stated in Equation (17), which can be thought of as the expectation of γ^k conditional on state s. In Equation (18), the quantity is calculated by either directly starting from state s, which correspond to $k = 0$ in Equation (17), or entering state s from state $\bar{s}$ with one step, corresponding to $k + 1 \geq 2$ in Equation (17).

For an arbitrary state $s \in \mathcal{S}$, using $s^{'}$ and $s^{''}$ to represent subsequent states as dummy index, the terms in square brackets in Equation (22) are simply Equation (21) with a and $s^{'}$ replaced by $a^{'}$ and $s^{''}$. Since $\nabla_\theta V^{\pi(\theta)}(s^{\infty}) = 0$, Equation (22) could be written as Equation (23), where s_k represent the state of k steps after s and $P^{\pi}(s \to s_k, k)$ already includes integration of intermediate state $s_{k-1}, \ldots s_1$ before reaching state s_k.

$$\nabla_\theta V^{\pi(\theta)}(s) = \sum_a \nabla_\theta \pi_\theta(a|s) Q^{\pi(\theta)}(s,a) + \sum_{k=1} \sum_{s_k} \sum_{a_k} \gamma^k P^{\pi}(s \to s_k, k) \nabla_\theta \pi_\theta(a_k|s_k) Q^{\pi(\theta)}(s_k, a_k) \tag{23}$$

Let objective function with respect to policy be defined to be the value function starting from auxiliary state s_0 as in Equation (24).

$$J(\pi_\theta) = V^{\pi}(s_0) = E_{\pi,\varepsilon} \sum_{t=0}^{\infty} \gamma^t R_t(S_0 = s) \tag{24}$$

The optimal policy could be obtained by gradient accent optimization, leading to the policy gradient algorithm [12],

$$\nabla_\theta V^{\pi(\theta)}(s) = \nabla_\theta \left[\sum_a Q^{\pi(\theta)}(s,a) \pi_\theta(a|s) \right] \tag{19}$$

$$= \sum_a \left[\nabla_\theta Q^{\pi(\theta)}(s,a) \pi_\theta(a|s) + \nabla_\theta \pi_\theta(a|s) Q^{\pi(\theta)}(s,a) \right]$$

$$= \sum_a \nabla_\theta \left[\sum_{s^{'},R} P(s^{'}, R|s, a) \left(R + \gamma V^{\pi(\theta)}(s^{'}) \right) \right] \pi_\theta(a|s) + \sum_a \nabla_\theta \pi_\theta(a|s) Q^{\pi(\theta)}(s,a) \tag{20}$$

$$= \sum_a \sum_{s^{'}} \gamma P(s^{'}|s,a) \nabla_\theta V^{\pi(\theta)}(s^{'}) \pi_\theta(a|s) + \sum_a \nabla_\theta \pi_\theta(a|s) Q^{\pi(\theta)}(s,a) \tag{21}$$

$$= \sum_a \nabla_\theta \pi_\theta(a|s) Q^{\pi(\theta)}(s,a) + \sum_a \sum_{s^{'}} \gamma P(s^{'}|s,a) \pi_\theta(a|s) \left[\sum_{a^{'}} \sum_{s^{''}} \gamma P(s^{''}|s^{'}, a^{'}) \nabla_\theta V^{\pi(\theta)}(s^{''}) \pi_\theta(a^{'}|s^{'}) + \sum_{a^{'}} \nabla_\theta \pi_\theta(a^{'}|s^{'}) Q^{\pi(\theta)}(s^{'}, a^{'}) \right] \tag{22}$$

as in Equation (29).

$$\nabla_\theta J(\pi_\theta) = \nabla_\theta V^{\pi}(s_0)$$

$$= \sum_{k=0} \sum_{s_k} \sum_{a_k} \gamma^k P^{\pi}(s_0 \to s_k, k) \nabla_\theta \pi_\theta(a_k|s_k) Q^{\pi(\theta)}(s_k, a_k)$$

$$= \sum_s \sum_a \eta(s) \nabla_\theta \pi_\theta(a|s) Q^{\pi(\theta)}(s,a) \tag{25}$$

$$= \frac{\sum_s \eta(s)}{\sum_s \eta(s)} \sum_s \sum_a \eta(s) \nabla_\theta \pi_\theta(a|s) Q^{\pi(\theta)}(s,a) \tag{26}$$

$$= \sum_{\bar{s}} \eta(\bar{s}) \sum_s \sum_a \mu(s) \nabla_\theta \pi_\theta(a|s) Q^{\pi(\theta)}(s,a) \tag{27}$$

$$= \sum_{\bar{s}} \eta(\bar{s}) \sum_s \sum_a \mu(s) \frac{\pi_\theta(a|s)}{\pi_\theta(a|s)} \nabla_\theta \pi_\theta(a|s) Q^{\pi(\theta)}(s,a) \tag{28}$$

$$\propto E_\pi \left[\frac{\nabla_\theta \pi_\theta(A|S)}{\pi_\theta(A|S)} \hat{Q}^{\pi(\theta)}(S,A) \right] \tag{29}$$

The policy gradient could be augmented to include zero gradient baseline $b(s)$, with respect to objective function $J(\pi_\theta)$ in Equation (28), as a function of state s, which does not include parameters for policy θ, since $\sum_a \nabla_\theta \pi_\theta(a|s) = 0$. To reduce variance of the gradient, the baseline is usually chosen to be the state value function estimator $\hat{V}_w(s)$ to smooth out the variation of $Q(s,a)$ at each state, while $\hat{V}_w(s)$ is updated in a Monte Carlo way by comparing with $\hat{Q}^{\pi_\theta}(S,A) = G_t$.

The actor-critic algorithm [12] decomposes $G_t - V_w(s_t)$ to be $R_t + \gamma V_w(s_{t+1}) - V_w(s_t)$, so bootstrap is used instead of Monte Carlo.

D. Basics of Deep Reinforcement Learning

Deep Q learning [1] makes a breakthrough in using neural network as the functional approximator for value function on complicated tasks. It solves the transition correlation problem by random sampling from a replay memory. Specifically, the reinforcement learning is transformed in a supervised learning task by fitting on the target $R_t + \gamma max_a Q(s_{t+1}, a)$ from the replay memory with state s_t as input. However, the target can get drifted easily which leads to unstable learning. In [1], a target network is used to provide a stable target for the updating network to be learned before getting updated occasionally. Double Deep Q learning [16], however, solves the problem by having two Q network and update the parameters in a alternating way. We review some state of art deep reinforcement learning algorithms from different aspects:

1) Off Policy methods: Except for Deep Q Learning [1] mentioned above, DDPG [17] extends Deterministic Policy Gradient (DPG) [18] with deep neural network functional approximator, which is an actor-critic algorithm and works well in continuous action spaces.

2) On Policy methods: A3C [19] stands out in the asynchronous methods in deep learning [19] which can be run in parallel on a single multi-core CPU. Trust Region Policy Optimization [2] and Proximal Policy Optimization [20] assimilates the natural policy gradient, which use a local approximation to the expected return. The local approximation could serve as a lower bound for the expected return, which can be optimized safely subject to the KL divergence constraint between two subsequent policies, while in practice, the constraint is relaxed to be a regularization.

3) Goal based Reinforcement Learning: In robot manipulation tasks, the goal could be represented with state in some cases [14]. Universal Value Function Approximator (UVFA) [21] incorporate the goal into the deep neural network, which let the neural network functional approximator also generalize to goal changes in tasks, similar to Recommendation System [22]. Work of this direction include [23], [14], for example.

4) Replay Memory Manipulation based Method: Replay memory is a critical component in Deep Reinforcement Learning, which solves the problem of correlated transition in one episode. Beyond the uniform sampling of replay memory in Deep Q Network [1], Prioritized Experience Replay [24] improves the performance by giving priority to those transitions with bigger TD error, while Hindsight Experience Replay (HER) [23] manipulate the replay memory with changing goals to transition so as to change reward to promote exploration. Maximum entropy regularized multi goal reinforcement learning [14] gives priority to those rarely occurred trajectory in sampling, which has been shown to improve over HER [14].

5) Surrogate policy optimization: Like surrogate model used in Bayesian Optimization [25], lower bound surrogate is also used in Reinforcement Learning. Trust Region Policy Optimization (TRPO) [2] is built on the identity from [26] in Equation (30), where $\eta_{\pi^{new}}(s)$ means the state visitation frequency under policy π^{new} and advantage $A^{\pi^{old}}(a_t, s_t) = Q^{\pi^{old}}(a_t, s_t) - V^{\pi^{old}}(s_t)$. Based on Policy Advantage [26] $A_{\pi^{old},\eta_{old}}(\pi^{new}) = \sum_s \eta_{\pi^{old}}(s) \sum_a \pi^{new}(a|s) A^{\pi^{old}}(a, s)$, a local approximation $L_{\pi^{old}}(\pi^{new})$ to Equation (30) can be defined in Equation (31), based on which, a surrogate function $M(\pi^{new}, \pi^{old})$ is defined in Equation (32) that minorizes $J(\pi^{new})$ at π^{old}, where $D_{KL}^{max}(\pi^{old}, \pi^{new}) = \max_s D_{KL}(\pi^{old}(a|s), \pi^{new}(a|s))$ is the maximum KL divergence, so MM [2] algorithm could be used to improve the policy, leading to the trust region method [2].

$$J(\pi^{new}) = J(\pi^{old}) + \sum_s \eta_{\pi^{new}}(s) \sum_a \pi^{new}(a|s) A^{\pi^{old}}(a, s) \tag{30}$$

$$\begin{aligned} &L_{\pi^{old}}(\pi^{new}) \\ =&J(\pi^{old}) + \sum_s \eta_{\pi^{old}}(s) \sum_a \pi^{new}(a|s) A^{\pi^{old}}(a_t, s_t) \end{aligned} \tag{31}$$

TABLE I
COMPARISON OF DEEP REINFORCEMENT LEARNING METHODS: "S" MEANS STATE AND "A" MEANS ACTION, WHERE "C" MEANS CONTINUOUS, "D" MEANS DISCRETE. "STANDALONE" MEANS WHETHER THE ALGORITHM WORK INDEPENDENTLY OR NEEDS TO BE COMBINED WITH ANOTHER LEARNING ALGORITHM. "VAR" MEANS WHICH PROBABILITY THE VARIATIONAL INFERENCE IS APPROXIMATING, "P" MEANS WHETHER THE METHOD IS ON POLICY OR OFF POLICY. "NA" MEANS NOT APPLICABLE

Algorithm	S	A	standalone	var	p
Deep Q	c	d	y	na	off
A3C	c	c/d	y	na	on
TRPO/PPO	c	c/d	y	na	on
DDPG	c	c	y	na	off
Boltzmann	d	d	y	na	on
VIME	c	c	n	$p_\theta(s_{t+1}\|s_t, a_t)$	na
VAST	c	d	n	$p(s_t\|o_{t-k})$	na
SoftQ	c	c/d	y	$p(a_t\|s_t)$	on

III. TAXONOMY OF PGM AND VI IN DEEP REINFORCEMENT LEARNING

Despite the success of deep reinforcement learning in many talks, the field still faces some critical challenges. One problem is exploration with sparse reward. In complicated real environment, an agent has to explore for a long trajectory before it can get any reward as feedback. Due to lack of enough rewards, traditional Reinforcement Learning methods

$$\begin{aligned} &M(\pi^{new}, \pi^{old}) \\ =&L_{\pi^{old}}(\pi^{new}) - \frac{4 max_{a,s} |A^{\pi^{old}}(s, a)| \gamma}{1 - \gamma^2} D_{KL}^{max}(\pi^{old}, \pi^{new}) \end{aligned} \tag{32}$$

performs poorly, which lead to a lot of recent contributions in the exploration methods. Another challenge is how to represent policy in extremely large state and action spaces. Furthermore, sometimes it is beneficial to have multimodal behavior for a agent when some trajectory might be equivalent to other trajectories and we want to learn all of them.

In this section, we give detailed explanation on how graphical model and variational inference could be used to model and optimize the reinforcement learning process under these challenges and form a taxonomy of these methods.

Together with the deep reinforcement learning methods mentioned in section II-D, we make a comparison of them in Table I.

A. Policy and value function with undirected graphs

We first discuss the application of undirected graphs in deep reinforcement learning, which models joint distribution of variables with cliques [7]. In [27], the authors use Restricted Boltzmann Machine (RBM) [8], which has nice property of tractable factorized posterior distribution over the latent variables conditioned on observed variables. To deal with MDPs of large state and action spaces, they model the state-action value function with the negative free energy of a Restricted Boltzmann Machine. Specifically, the visible states of the Restricted Boltzmann Machine [27] consists of both state s and action a binary variables, as shown in Figure 6, where the hidden nodes consist of L binary variables, while state variables s_i are dark colored to represent it can be observed and actions a_j are light colored to represent it need to be sampled. Together with the auxiliary hidden variables, the undirected graph defines a joint probability distribution over state and action pairs, which defines a stochastic policy network that could sample actions out for on policy learning. Since it is pretty easy to calculate the derivative of the free energy $F(s,a)$ with respect to the coefficient $w_{k,j}$ of the network, one could use temporal difference learning to update the coefficients in the network. Thanks to properties of Boltzmann Machine, the conditional distribution of action over state $p(a|s)$, which could be used as a policy, is still Boltzmann distributed as in Equation (33), governed by the free energy $F(a,s)$, where $Z(s)$ is the partition function [7] and the negative free energy to approximate the state action value function $Q(s,a)$. By adjusting the temperature T, one could also change between different exploration strength.

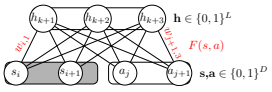


Fig. 6. Restricted Boltzmann Machine Value and Policy

$$p(a|s) = 1/Z(s)e^{-F(s,a)/T} = 1/Z(s)e^{Q(s,a)/T} \tag{33}$$

A few steps of MCMC sampling [7] could be used to sample actions, as an approximation of the policy, which can be fed into a time difference learning method like SARSA [12], to update the state value function $Q(s,a)$'s estimation. Such an on-policy process has been shown to be empirically effective in the large state actions spaces [27].

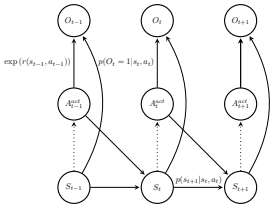


Fig. 7. Optimal Policy as posterior on actions: $p(a_t|s_t, O_{t:T} = 1)$

B. Variational Inference on "optimal" Policies

1) policy as "optimal" posterior: The Boltzmann Machine defined Product of Expert Model in [27] works well for large state and action spaces, but are limited to discrete specifically binary state and action variables. For continuous state and action spaces, in [28], the author proposed deep energy based models with Directed Acyclic Graphs (DAG) [7], which we re-organize in a different form in Figure 7 with annotations added. The difference with respect to Figure 3 is that, in Figure 7, the reward is not explicit expressed in the directed graphical model. Instead, an auxilliary binary Observable O is used to define whether the corresponding action at the current step is optimal or not. The conditional probability of the action being optimal is $p(O_t = 1|s_t, a_t) = \exp(r(s_t, a_t))$, which connects conditional optimality with the amount of award received by encouraging the agent to take highly rewarded actions in an exponential manner. Note that the reward here must be negative to ensure the validity of probability, which does not hurt generality since reward range can be translated [13].

The Graphical Model in Figure 7 in total defines the trajectory likelihood or the evidence in Equation (34):

$$p(\tau) = \left[p(s_1)\prod_t p(s_{t+1}|s_t, a_t)\right] \exp\left(\sum_t r(s_t, a_t)\right) \tag{34}$$

.

By doing so, the author is forcing a form of functional expression on top of the conditional independence structure of the graph by assigning a likelihood. In this way, calculating the optimal policy of actions distributions becomes an inference problem of calculating the posterior $p(a_t|s_t, O_{t:T} = 1)$, which reads as, conditional on optimality from current time step until end of episode, and the current current state to be s_t, the distribution of action a_t, and this posterior corresponds to the optimal policy. Observing the d-separation from Figure 7, $O_{1:t-1}$ is conditionally independent of a_t given s_t, $(O_{1:t-1} \perp\!\!\!\perp A_t^{act}) \mid S_t$, so $p(a_t|s_t, O_{1:t-1}, O_{t:T}) = p(a_t|s_t, O_{t:T})$

2) Message passing for exact inference on the posterior: In this section, we give detailed derivation on conducting exact inference on the policy posterior which is not given

$$
\begin{aligned}
& p(a_t|s_t, O_{t:T}=1) \\
=& \frac{p(a_t, s_t, O_{t:T}=1)}{p(s_t, O_{t:T}=1)} \\
=& \frac{p(O_{t:T}=1|a_t, s_t)p(a_t, s_t)}{p(s_t, O_{t:T}=1)} \\
=& \frac{p(O_{t:T}=1|a_t, s_t)p(a_t|s_t)p(s_t)}{\int_{a_{t'}} p(s_t, a_t', O_{t:T}=1)d\{a_t'\}} \\
=& \frac{p(O_{t:T}=1|a_t, s_t)p(a_t|s_t)p(s_t)}{\int_{a_t'} p(O_{t:T}=1|a_t', s_t)p(a_t'|s_t)p(s_t)d\{a_t'\}} \\
=& \frac{p(O_{t:T}=1|a_t, s_t)p(a_t|s_t)}{\int_{a_t'} p(O_{t:T}=1|a_t', s_t)p(a_t'|s_t)d\{a_t'\}} \\
=& \frac{\beta(a_t, s_t)}{\int_{a_t'} \beta(a_t', s_t)d\{a_t'\}} \\
=& \frac{\beta(a_t, s_t)}{\beta(s_t)}
\end{aligned} \tag{35}
$$

in [13]. Similar to the forward-backward message passing algorithm [7] in Hidden Markov Models [7], the posterior $p(a_t|s_t, O_{t:T}=1)$ could also be calculated by passing messages. We offer a detailed derivation of the decomposition of the posterior $p(a_t|s_t, O_{t:T}=1)$ in Equation (35), which is not available in [13]. In Equation (35), we define message $\beta(a_t, s_t) = p(O_{t:T}=1|a_t, s_t)p(a_t|s_t)$ and message $\beta(s_t) = \int_{a_t'} \beta(a_t, s_t)d\{a_t'\}$. If we consider $p(a_t|s_t)$ as a prior with a trivial form of uniform distribution [13], the only policy related term becomes $p(O_{t:T}=1|a_t, s_t)$.

In contrast to HMM, here, only the backward messages are relevant. Additionally, the backward message $\beta(a_t, s_t)$ here is not a probability distribution as in HMM, instead, is just a probability. In Figure 7, the backward message $\beta(a_t, s_t)$ could be decomposed recursively. Since in [13] the author only give the conclusion without derivation, we give a detailed derivaion of this recursion in Equation (36). The recursion in Equation (36) start from the last time point T of an episode.

3) Connection between Message Passing and Bellman equation: If we define Q function in Equation (37) and V function in Equation (38)

$$Q(s_t, a_t) = \log(\beta(a_t, s_t)) \tag{37}$$

then the corresponding policy could be written as Equation (39).

$$\pi(a_t|s_t) = p(a_t|s_t, O_{t:T}=1) = \exp(Q(s_t, a_t) - V(s_t)) \tag{39}$$

Taking the logrithm of Equation (36), we get Equation (40) which reduces to the risk seeking backup in Equation (41) as mentioned in [13]:

$$Q(s_t, a_t) = r(s_t, a_t) + \log E_{s_{t+1}\sim p(s_{t+1}|s_t, a_t)}[\exp(V(s_{t+1}))] \tag{41}$$

$$
\begin{aligned}
& \beta(s_t, a_t) \\
=& p(O_t=1, O_{t+1:T}=1|s_t, a_t) \\
=& \frac{\int p(O_t=1, O_{t+1:T}=1, s_t, a_t, s_{t+1}, a_{t+1})d\{s_{t+1}, a_{t+1}\}}{p(s_t, a_t)} \\
=& \int p(O_{t+1:T}=1, s_{t+1}, a_{t+1}, O_t=1|s_t, a_t)d\{s_{t+1}, a_{t+1}\} \\
=& \int p(O_{t+1:T}=1, s_{t+1}, a_{t+1}|s_t, a_t)p(O_t=1|s_t, a_t) \\
& d\{s_{t+1}, a_{t+1}\} \qquad ((O_{t+1:T}, S_{t+1}, A_{t+1} \perp\!\!\!\perp O_t) \mid S_t, A_t) \\
=& \int \frac{p(O_{t+1:T}=1, s_{t+1}, a_{t+1})}{p(s_{t+1}, a_{t+1})} \frac{p(s_{t+1}, s_t, a_t)}{p(s_t, a_t)} \\
& p(O_t=1|s_t, a_t)d\{s_{t+1}, a_{t+1}\} \\
=& \int p(O_{t+1:T}=1|s_{t+1}, a_{t+1})p(s_{t+1}|s_t, a_t)p(O_t=1|s_t, a_t) \\
& d\{s_{t+1}, a_{t+1}\} \\
=& \int \beta(s_{t+1})p(s_{t+1}|s_t, a_t)p(O_t=1|s_t, a_t)ds_{t+1}
\end{aligned} \tag{36}
$$

$$
\begin{aligned}
V(s_t) = \log \beta(s_t) &= \log \int \beta(s_t, a_t)da_t \\
&= \log \int exp(Q(s_t, a_t))da_t \approx \max_{a_t} Q(s_t, a_t)
\end{aligned} \tag{38}
$$

The mathematical insight here is that if we define the messages passed on the Directed Acyclic Graph in Figure 7, then message passing correspond to a peculiar version Bellman Equation like backup, which lead to an unwanted risk seeking behavior [13]: when compared to Equation (10), the Q function here is taking a softmax instead of expectation over the next state.

4) Variational approximation to "optimal" policy: Since the exact inference lead to unexpected behavior, approximate inference could be used. The optimization of the policy could be considered as a variational inference problem, and we use the variational policy of the action posterior distribution $q(a_t|s_t)$, which could be represented by a neural network, to compose the proposal variational likelihood of the trajectory as in Equation (42): where the initial state distribution $p(s_1)$ and the environmental dynamics of state transmission is kept

$$
\begin{aligned}
& \log(\beta(s_t, a_t)) \\
&= \log \int \beta(s_{t+1})p(s_{t+1}|s_t, a_t)p(O_t=1|s_t, a_t)ds_{t+1} \\
&= \log \int \exp[r(s_t, a_t) + V(s_{t+1})]p(s_{t+1}|s_t, a_t)ds_{t+1} \\
&= r(s_t, a_t) + \log \int \exp(V(s_{t+1}))p(s_{t+1}|s_t, a_t)ds_{t+1}
\end{aligned} \tag{40}
$$

$$q(\tau) = p(s_1)\prod_t [p(s_{t+1}|s_t, a_t)q(a_t|s_t)] \tag{42}$$

$$\begin{aligned}
&\log(p(O_{1:T})) \\
=&\log \int p(O_{1:T} = 1, s_{1:T}, a_{1:T}) \frac{q(s_{1:T}, a_{1:T})}{q(s_{1:T}, a_{1:T})} ds_{1:T} da_{1:T} \\
=&\log E_{q(s_{1:T}, a_{1:T})} \frac{p(O_{1:T} = 1, s_{1:T}, a_{1:T})}{q(s_{1:T}, a_{1:T})} \\
\geq& E_{q(s_{1:T}, a_{1:T})}[\log p(O_{1:T} = 1, s_{1:T}, a_{1:T}) - \log q(s_{1:T}, a_{1:T})] \qquad (43) \\
=& - D_{KL}(q(\tau)|p(\tau)) \qquad (44) \\
=& E_{q(s_{1:T}, a_{1:T})}[\sum_{t=1:T} [r(s_t, a_t) - \log q(a_t|s_t)]] \\
=& \sum_{t=1:T} E_{s_t, a_t}[r(s_t, a_t) + H(\pi(a_t|s_t))] \qquad (45)
\end{aligned}$$

intact. Using the proposal trajectory as a pivot, we could derive the Evidence Lower Bound (ELBO) of the optimal trajectory as in Equation (43), which correspond to an interesting objective function of reward plus entropy return, as in Equation (45).

5) Examples: In [28], the state action value function is defined in Equation (46). and a soft version of Bellman update similar to Q Learning [12] is carried out, which lead to policy improvement with respect to the corresponding functional objective in Equation (47). Setting policy as Equation (39) lead to policy improvement. We offer a detailed proof for a key formula in Equation (48), which is stated in Equation (19) of [28] without proof. In Equation (48), we use $\pi(\cdot|s)$ to implicitly represent $\pi(a|s)$ to avoid symbol aliasing whenever necessary. For the rest of the proof, we invite the reader to read the appendix of [28]. Algorithms of the this kind of maximum entropy family also include Soft Actor Critic [29].

$$Q^{\pi}_{soft}(s, a) = r_0 + E_{r\sim\pi, s_0=s, a_0=a}[\sum_{t=1}^{\infty} \gamma^t (r_t + \alpha H(\pi(.|s_t)))] \tag{46}$$

$$\begin{aligned}
&J(\pi) \\
=&\sum_t E_{(s_t, a_t)\sim\rho_\pi} \sum_{l=t}^{\infty} \gamma^{l-t} E_{(s_l, a_l)}[r(s_l, a_l) + \\
&\alpha H(\pi(.|s_l))|s_t, a_t]] \\
=&\sum_t E_{(s_t, a_t)\sim\rho_\pi}[Q^{\pi}_{soft}(s_t, a_t) + \alpha H(\pi(.|s_t))] \qquad (47)
\end{aligned}$$

$$\begin{aligned}
&H(\pi(\cdot|s)) + E_{a\sim\pi}[Q^{\pi}_{soft}(s, a)] \\
=& - \int_a \pi(a|s)[\log \pi(a|s) - Q^{\pi}_{soft}(s, a)] da \\
=& - \int_a \pi(a|s)[\log \pi(a|s) - \log[\exp(Q^{\pi}_{soft}(s, a))]] da \\
=& - \int_a \pi(a|s)[\log \pi(a|s) - \log[\frac{\exp(Q^{\pi}_{soft}(s, a))}{\int \exp(Q^{\pi}_{soft}(s, a^{'}))da^{'}} \\
&\int \exp(Q^{\pi}_{soft}(s, a^{'}))da^{'}]] da \\
=& - \int_a \pi(a|s)[\log \pi(a|s) - \log[\tilde{\pi}(a|s)] - \\
&log \int exp(Q^{\pi}_{soft}(s, a^{'}))]da^{'} \\
=& - D_{KL}(\pi(\cdot|s)||\tilde{\pi}(\cdot|s)) + \log \int \exp(Q^{\pi}_{soft}(s, a^{'}))da^{'} \qquad (48)
\end{aligned}$$

C. Variational Inference on the Environment

Another direction of using Variational Inference in Reinforcement Learning is to learn an environmental model, either on the dynamics or the latent state space posterior.

1) Variational inference on transition model: In Variational Information Maximizing Exploration (VIME) [4], where dynamic model $p_\theta(s_{t+1}|s_t, a_t)$ for the agent's interaction with the environment is modeled using Bayesian Neural Network [10]. The R.V. for θ is denoted by Θ, and is treated in a Bayesian way by modeling the weight θ uncertainty of a neural network. We represent this model with the graphical model in Figure 8, which is not given in [4]. The belief uncertainty about the environment is modeled as entropy of the posterior distribution of the neural network weights $H(\Theta|\xi_t)$ based on trajectory observations $\xi_t = \{s_{1:t}, a_{1:t-1}\}$. The method encourages taking exploratory actions by alleviating the average information gain of the agent's belief about the environment after observing a new state s_{t+1}, which is $E_{p(s_{t+1}|\xi_t, a_t)} D_{KL}(p(\theta|\xi_{t+1})||p(\theta|\xi_t))$, and this is equivalent to the entropy minus conditional entropy $H(\Theta|\xi_t, a_t) - H(\Theta|\xi_t, a_t, s_{t+1}) = H(\Theta|\xi_t, a_t) - H(\Theta|\xi_{t+1})$. With the help of Equation (49), as derived following the definition of conditional mutual information, we derive in Equation (50) that the conditional entropy difference is actually the average information gain, which is equal to the conditional mutual information $I(\Theta, S_{t+1}|\xi_t, a_t)$ between environmental parameter Θ and the new state S_{t+1}. Such a derivation is not given in [4]. Based on Equation (50), an intrinsic reward can be augmented from the environmental reward function, thus the method could be incorporated with any existing reinforcement learning algorithms for exploration, TRPO [2], for example. Upon additional observation of action a_t and state s_{t+1} pair on top of trajectory history ξ_t, the posterior on the distribution of the environmental parameter θ, $p(\theta|\xi_t)$, could be updated to be $p(\theta|\xi_{t+1})$ in a Bayesian way as derived

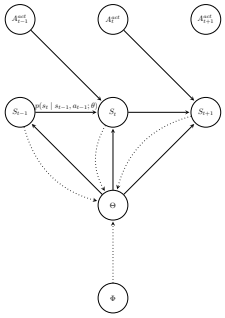


Fig. 8. Probabilistic Graphical Model For VIME

$$
\begin{aligned}
I(X;Y \mid Z) &= \int_{x,y,z} p(z)p(x,y|z)\log\frac{p(x,y|z)}{p(x|z)p(y|z)}dxdydz \\
&= -\int_{x,y,z} p(z)p(x,y|z)\log p(x|z)dxdzdy + \\
&+ \int_{x,y,z} p(x,y,z)\log p(x|y,z)dxdzdy \\
&= H(X \mid Z) - H(X \mid Y,Z) \qquad (49)
\end{aligned}
$$

$$
\begin{aligned}
p(\theta|\xi_{t+1}) =& \frac{p(\theta,\xi_t,a_t,s_{t+1})}{p(\xi_t,a_t,s_{t+1})} \\
=& \frac{p(s_{t+1}|\theta,\xi_t,a_t)p(\theta,\xi_t,a_t)}{p(\xi_t,a_t,s_{t+1})} \\
=& \frac{p(s_{t+1}|\theta,\xi_t,a_t)p(\theta,\xi_t,a_t)}{p(a_t,\xi_t)p(s_{t+1}|a_t,\xi_t)} \\
=& \frac{p(s_{t+1}|\theta,\xi_t,a_t)p(\theta|\xi_t,a_t)}{p(s_{t+1}|a_t,\xi_t)} \\
=& \frac{p(s_{t+1}|\theta,\xi_t,a_t)p(\theta|\xi_t)}{p(s_{t+1}|a_t,\xi_t)} \qquad (51)
\end{aligned}
$$

$$
\begin{aligned}
& p(s_{t+1}|a_t,\xi_t) \\
&= \int_\Theta p(s_{t+1},\theta|a_t,\xi_t)d\theta \\
&= \int_\Theta \frac{p(s_{t+1},\theta,a_t,\xi_t)}{p(a_t,\xi_t)}d\theta \\
&= \int_\Theta \frac{p(s_{t+1}|\theta,a_t,\xi_t)p(\theta,a_t,\xi_t)}{p(a_t,\xi_t)}d\theta \\
&= \int_\Theta p(s_{t+1}|\theta,a_t,\xi_t)p(\theta|\xi_t)d\theta \qquad (52)
\end{aligned}
$$

in Equation (51), which is first proposed in [30]. In Equation (51), the denominator can be written as Equation (52), so that the dynamics of the environment modeled by neural network weights θ, $p(s_{t+1}|\theta,a_t,\xi_t)$, could be used. The last step of Equation (52) makes use of $p(\theta|\xi_t,a_t) = p(\theta|\xi_t)$.

Since the integral in Equation (52) is not tractable, variational treatment over the neural network weights posterior distribution $p(\theta|\xi_t)$ is used, characterized by variational parameter ϕ, as shown in the dotted line in Figure 8. The variational posterior about the model parameter θ, updated at each step, could than be used to calculate the intrinsic reward in Equation (50).

2) Variational Inference on hidden state posterior: In Variational State Tabulation (VaST) [5], the author assume the high dimensional observed state to be represented by Observable O, while the transition happens at the latent state space represented by S, which is finite and discrete. The author assume a factorized form of observation and latent space joint probability, which we explicitly state in Equation (53).

$$
\begin{aligned}
& H(\Theta|\xi_t,a_t) - H(\Theta|\xi_t,a_t,s_{t+1}) = I(\Theta,S_{t+1}|\xi_t,a_t) \\
=& E_{\xi_t,a_t}\int_{\Theta,\mathcal{S}} p(s_{t+1},\theta|\xi_t,a_t)\log[\frac{p(s_{t+1},\theta|\xi_t,a_t)}{p(\theta|\xi_t)p(s_{t+1}|\xi_t,a_t)}]d\theta \\
& ds_{t+1} \\
=& E_{\xi_t,a_t}\int_{\Theta,\mathcal{S}} p(s_{t+1}|\xi_t,a_t)p(\theta|\xi_{t+1})\log[\frac{p(\theta|\xi_{t+1})}{p(\theta|\xi_t)}]d\theta ds_{t+1} \\
=& E_{\xi_t,a_t}E_{p(s_{t+1}|\xi_t,a_t)}D_{KL}(p(\theta|\xi_{t+1})||p(\theta|\xi_t)) \qquad (50)
\end{aligned}
$$

$$
p(O,S) = \pi_{\theta_0}(s_0)\prod_{t=0}^{T} p_{\theta^R}(o_t|s_t)\prod_{t=1}^{T} p_{\theta^T}(s_t|s_{t-1},a_{t-1}) \qquad (53)
$$

Additionally, we characterize Equation (53) with the probabilistic graphical model in Figure 9 which does not exist in [5]. Compared to Figure 7, here the latent state S is in discrete space instead of high dimension, and the observation is a high dimensional image instead of binary variable to indicate optimal action. By assuming a factorized form of the variational posterior in Equation (54),

$$
q(S_{0:T}|O_{0:T}) = \prod_{t=0}^{T} q_\phi(S_t|O_{t-k:t}) \qquad (54)
$$

The author assume the episode length to be T, and default frame prior observation to be blank frames. The Evidence Lower Bound (ELBO) of the observed trajectory of Equation (53) could be easily represented by a Varitional AutoEncoder [31] like architecture, where the encoder q_ϕ, together with the reparametrization trick [31], maps the observed state O into parameters for the Con-crete distribution [32], so backprobagation could be used on deterministic variables to update the weight of the network based on the ELBO, which is decomposed into different parts of the reconstruction losses of the variational autoencoder like architecture. Like VIME [4], VaSt could be combined with other reinforcement learning algorithms. Here prioritized sweeping [12] is carried out on the

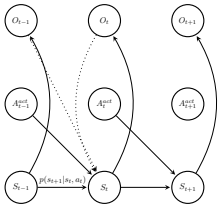


Fig. 9. Graphical Model for Variation State Tabulation

Heviside activation of the encoder output directly, by counting the transition frequency, instead of waiting for the slowly learned environmental transition model $p_{\theta^T}(s_t|s_{t-1}, a_{t-1})$ in Equation (53). A potential problem of doing so is aliasing between latent state s and observed state o. To alleviate this problem, in [5], the author actively relabel the transition history in the replay memory once found the observable has been assigned a different latent discrete state.

IV. Conclusion

As a tutorial survey, we recap Reinforcement Learning with Probabilistic Graphical Models, summarizes recent advances of Deep Reinforcement Learning and offer a taxonomy of Probabilistic Graphical Model and Variational Inference in DRL with detailed derivations which are not included in the original contributions.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[2] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*, 2015, pp. 1889–1897.

[3] X. Sun, J. Lin, and B. Bischl, "Reinbo: Machine learning pipeline search and configuration with bayesian optimization embedded reinforcement learning," 2019.

[4] R. Houthooft, X. Chen, Y. Duan, J. Schulman, F. De Turck, and P. Abbeel, "Vime: Variational information maximizing exploration," in *Advances in Neural Information Processing Systems*, 2016, pp. 1109–1117.

[5] D. Corneil, W. Gerstner, and J. Brea, "Efficient model-based deep reinforcement learning with variational state tabulation," *arXiv preprint arXiv:1802.04325*, 2018.

[6] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe, "Variational inference: A review for statisticians," *Journal of the American Statistical Association*, vol. 112, no. 518, pp. 859–877, 2017.

[7] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.

[8] A. Fischer and C. Igel, "Training restricted boltzmann machines: An introduction," *Pattern Recognition*, vol. 47, pp. 25–39, 01 2014.

[9] X. Sun, A. Gossmann, Y. Wang, and B. Bischl, "Variational resampling based assessment of deep neural networks under distribution shift," 2019.

[10] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight uncertainty in neural networks," *arXiv preprint arXiv:1505.05424*, 2015.

[11] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[12] R. S. Sutton, A. G. Barto *et al.*, *Introduction to reinforcement learning*. MIT press Cambridge, 1998, vol. 2, no. 4.

[13] S. Levine, "Reinforcement learning and control as probabilistic inference: Tutorial and review," *arXiv preprint arXiv:1805.00909*, 2018.

[14] R. Zhao, X. Sun, and V. Tresp, "Maximum entropy-regularized multi-goal reinforcement learning," *arXiv preprint arXiv:1905.08786*, 2019.

[15] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial i ntelligence*, vol. 101, no. 1-2, pp. 99–134, 1998.

[16] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Thirtieth AAAI conference on artificial intelligence*, 2016.

[17] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[18] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," 2014.

[19] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, 2016, pp. 1928–1937.

[20] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[21] T. Schaul, D. Horgan, K. Gregor, and D. Silver, "Universal value function approximators," in *International Conference on Machine Learning*, 2015, pp. 1312–1320.

[22] N. Kushwaha, X. Sun, B. Singh, and O. Vyas, "A lesson learned from pmf based approach for semantic recommender system," *Journal of Intelligent Information Systems*, vol. 50, no. 3, pp. 441–453, 2018.

[23] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. P. Abbeel, and W. Zaremba, "Hindsight experience replay," in *Advances in Neural Information Processing Systems*, 2017, pp. 5048–5058.

[24] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[25] X. Sun, A. Bommert, F. Pfisterer, J. Rahnenführer, M. Lang, and B. Bischl, "High dimensional restrictive federated model selection with multi-objective bayesian optimization over shifted distributions," *arXiv preprint arXiv:1902.08999*, 2019.

[26] S. Kakade and J. Langford, "Approximately optimal approximate reinforcement learning," in *ICML*, vol. 2, 2002, pp. 267–274.

[27] B. Sallans and G. E. Hinton, "Reinforcement learning with factored states and actions," *Journal of Machine Learning Research*, vol. 5, no. Aug, pp. 1063–1088, 2004.

[28] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine, "Reinforcement learning with deep energy-based policies," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 1352–1361.

[29] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," *arXiv preprint arXiv:1801.01290*, 2018.

[30] Y. Sun, F. Gomez, and J. Schmidhuber, "Planning to be surprised: Optimal bayesian exploration in dynamic environments," in *International Conference on Artificial General Intelligence*. Springer, 2011, pp. 41–51.

[31] C. Doersch, "Tutorial on variational autoencoders," *arXiv preprint arXiv:1606.05908*, 2016.

[32] C. J. Maddison, A. Mnih, and Y. W. Teh, "The concrete distribution: A continuous relaxation of discrete random variables," *arXiv preprint arXiv:1611.00712*, 2016.

Appendix F

Machine Learning Pipeline Conditional Hierarchy Search and Configuration with Bayesian Optimization Embedded Reinforcement Learning

Contributing Article Sun, Xudong; Lin, Jiali; Bischl, Bernd; ReinBo: Machine Learning Pipeline Conditional Hierarchy Search and Configuration with Bayesian Optimization Embedded Reinforcement Learning, Joint European Conference on Machine Learning and Knowledge Discovery in Databases, 68-84, 2019, Springer, Cham.

Author Contributions Xudong Sun initiated the idea and designed the first version of the algorithm, with the final version of the algorithm attributed to iterated discussions between Xudong Sun, Jiali Lin and Bernd Bischl. Xudong Sun implemented the reinforcement learning part of the algorithm, the hyper-parameter search space specification and ancestral sampling independently, he also refactored code from Jiali Lin. Jiali Lin independently proposed the idea of using small budgets to interleave bayesian optimization and reinforcement learning. She implemented the reinforcement learning environment for evaluating the pipeline, wrote the benchmark code, conducted the experiment and generated the experimental results. The implementation of the competitor algorithms in the benchmark study was joint work of Xudong Sun and Jiali Lin, many vital points of the algorithm were resolved through intensive discussion between both junior authors. Both junior authors contributed to the manuscript with Xudong Sun taking a leading role. Bernd Bischl reviewed the paper.

ReinBo: Machine Learning Pipeline Conditional Hierarchy Search and Configuration with Bayesian Optimization Embedded Reinforcement Learning

Xudong Sun(✉), Jiali Lin(✉), and Bernd Bischl(✉)

Department of Statistics, Ludwig Maximillian University of Munich, Munich, Germany
Xudong.Sun@stat.uni-muenchen.de, linjialideu@gmail.com, bernd_bischl@gmx.net

Abstract. Machine learning pipeline potentially consists of several stages of operations like data preprocessing, feature engineering and machine learning model training. Each operation has a set of hyper-parameters, which can become irrelevant for the pipeline when the operation is not selected. This gives rise to a hierarchical conditional hyper-parameter space. To optimize this mixed continuous and discrete conditional hierarchical hyper-parameter space, we propose an efficient pipeline search and configuration algorithm which combines the power of Reinforcement Learning and Bayesian Optimization. Empirical results show that our method performs favorably compared to state of the art methods like Auto-sklearn, TPOT, Tree Parzen Window, and Random Search.

Keywords: Bayesian Optimization · Reinforcement Learning · Conditional hierarchy search · AutoML

1 Introduction

Over the past years, Machine Learning (ML) has achieved remarkable success in a wide range of application areas, which has greatly increased the demand for machine learning systems. However, an efficient machine learning algorithm crucially depends on a human expert, who has to carefully design the pipeline of the machine learning system, potentially consisting of data preprocessing, feature filtering, machine learning algorithm selection, as well as identifying a good set of hyper-parameters. As there are a large number of possible alternatives of models as well as hyper-parameters, the need for automated machine learning (AutoML) has been growing, which is supposed to automatically generate a data analysis pipeline with machine learning methods and parameter settings that

X. Sun and J. Lin—Equal contribution.

P. Cellier and K. Driessens (Eds.): ECML PKDD 2019 Workshops, CCIS 1167, pp. 68–84, 2020.
https://doi.org/10.1007/978-3-030-43823-4_7

are optimized for a given data set, in order to make machine learning methods available for non-expert users.

Since hyper-parameters of a machine learning model have a large influence on the performance of the model, hyper-parameter optimization becomes a critical part of an AutoML system. Popular hyper-parameter optimization methods include Sequential Bayesian Optimization, which iterates between fitting surrogate models that predict model performance, and using them to make choices about which configurations to investigate.

However, the composition of the machine learning pipelines also plays a vital role in the performance of AutoML systems. Choosing different data preprocessing or feature engineering techniques as well as choosing different machine learning models for a specific dataset could potentially result in considerable performance differences. The joint optimization of the pipeline search and its associated hyper-parameters configuration could essentially reside under the umbrella of Combined Algorithm Selection and Hyperparameter optimization (CASH) problem [31], where Algorithm corresponds to the pipeline and Configuration corresponds to the hyper-parameters associated with the pipeline. The pipelines and hyper-parameters reside in a conditional hierarchical space, where some hyper-parameters only become valid when the corresponding pipeline is present. For example, Fig. 1 illustrates such a situation when the data preprocessing and feature engineering operations are selected, which correspond to an incomplete pipeline, one out of three machine learning algorithms need to be chosen (indicated by dashed edges) to complete the pipeline, the corresponding hyper-parameters (indicated by solid edges) of an algorithm only become valid when the algorithm is selected.

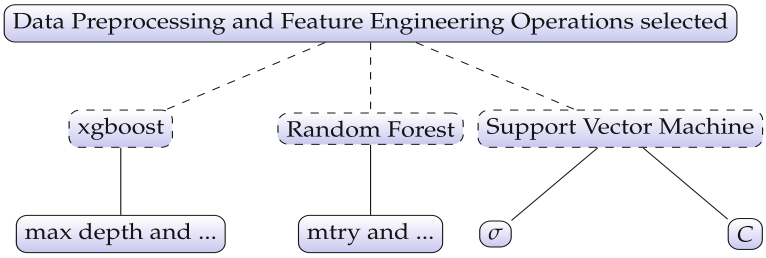


Fig. 1. Example of conditional hierarchical space

To optimize the conditional hyper-parameters space jointly with the pipeline it is attached to, we embed Bayesian Optimization in the Reinforcement Learning process, and dub the method ReinBo, which means Machine Learning Pipeline search and configuration with Reinforcement Learning and Bayesian Optimization. Note that ReinBo can solve not only CASH problems, but also any mixed discrete and continuous conditional hierarchical space optimization, which is left for future work.

Our major contributions are:

- Inspired by Hierarchical Reinforcement Learning [14], we transform the conditional hierarchical hyper-parameter optimization problem into subtasks of pipeline selection and hyper-parameter optimization, which circumvents the conditional constraint and reduces the search dimension.
- To our best knowledge, we are the first to embed Bayesian Optimization (BO) into Reinforcement learning, specifically Q Learning [32] for collaborative joint search of pipelines and hyper-parameters, which is different from using BO for policy optimization [12], and also different from using BO for hyper-parameter fine tuning after an optimal pipeline is selected by a reinforcement learning based AutoML framework [33].
- We provide an open source light weight R language implementation with benchmark codes[1] for the R Machine Learning community which could run efficiently on a personal computer, and takes much less resources (IO, disk space for example) compared to other AutoML softwares.

In the following section, we review related works and discuss the differences to our method. In Sect. 3, we explain our method in detail and also shed light to connections with Hyperband [22]. In Sect. 4, we benchmark our method by comparing it with several state of the art methods.

2 Related Work

In this section, we try to classify the current popular AutoML solutions into a taxonomy and discuss the differences of each individual work with ours.

Sequential Model Based Optimization Family. Auto-sklearn [16] and Auto-Weka [31] both use Sequential Model-based Algorithm Configuration (SMAC) [18] to solve the Combined Algorithm Selection and Hyperparameter optimization (CASH) problem. SMAC [18] transforms the conditional hierarchical hyper-parameter space into a flat structure by instantiating inactive conditional parameters to default values, which allows the random forest to focus on active hyper-parameters [18]. A potential drawback for this method is that the surrogate model needs to learn in a high dimensional hyper-parameter space, which might need a large sample of observations to be sufficiently trained, while in practice, running machine learning algorithm is usually very expensive. Tree Parzen Window (TPE) [7], however, tackles the conditional hierarchical hyper-parameter space using a tree like Parzen Window to construct two density estimators on top of a tree like hyper-parameter set. Expected improvement induced from lower and upper quantile density estimators is used to select new candidate proposals from points generated by Ancestral Sampling.

[1] https://github.com/compstat-lmu/paper_2019_ReinBo.

Tree-Based Genetic Programming. TPOT [25] automatically designs and optimizes machine learning pipelines with a genetic programming [3] algorithm. The machine learning operators are used as genetic programming primitives, which will be combined by tree-based pipelines and the Genetic Programming algorithm is used to evolve tree-based pipelines until the best pipeline is found. Similar methods also include Recipe [27]. However, this family of methods does not scale well [24]. In this paper, we aim for an AutoML system that could give a valuable configured pipeline within limited time.

Monte Carlo Tree Search Alike. ML-Plan [24] is an AutoML system, built upon a Hierarchical Task Network, which uses a Monte Carlo Tree Search like algorithm to search for pipelines and also configure the pipeline with hyper-parameters. Task is expanded based on best-first search, where the score is estimated by a randomized depth first search by randomly trying different subtree possibilities on a Hierarchical Task Network. To ensure exploration, ML-Plan gives equal possibility to the starting node in a Hierarchical Task Network and then uses a best-first strategy for searching at the lower part of the network. Potential drawback for this method is that the hyper-parameter space is discretized, which might essentially lose good candidates in continuous spaces since large continuous hyper-parameter spaces would be essentially hard to discretize.

Reinforcement Learning Based Neural Network Architecture Search. This family of methods are usually not termed as AutoML systems but rather Neural Architecture Search. For instance, MetaQNN [2] uses Q-learning to search convolutional neural network architectures. The learning agent is trained to sequentially choose CNN layers using Q-learning with an ϵ-greedy exploration strategy and the goal is to maximize the cross-validation accuracy. In [35], instead of using Q-learning, the authors use Recurrent Neural Networks as the reinforcement learning policy approximator to generate variable strings to represent various neural architecture forms. The reward-function is designed to be the validation performance of the constructed network. The reinforcement learning policy is trained with gradient descent algorithm, specifically REINFORCE. The architecture elements being searched are very similar to MetaQNN. Inspired from [35], we also assume the machine learning pipeline to be optimized could be represented by a variable length string, but our work is different from [35] in that we use both Deep Q-learning and Tabular Q-learning. More importantly, compared with both [2] and [35], which optimize the neural architecture, the elements of the architecture are mostly factor variables like layer type or discretized elements like filter size, while in this paper, we deal heavily with continuous hyper-parameters (The C and σ for a rbf kernel Support Vector Machine). To jointly optimize the discrete pipeline choice and associated continuous hyper-parameters, we embed Bayesian Optimization inside our reinforcement learning agent.

Other Reinforcement Learning Based Methods. In [33], the authors also combine pipeline search and hyper-parameter optimization in a reinforcement

learning process based on the PEORL [34] framework, however, the hyper-parameter is randomly sampled during the reinforcement learning process, an extra stage is needed to sweep the hyper-parameters using hyper-parameter optimization techniques, while in our work, hyper-parameter optimization is embedded in the reinforcement learning process. Alpha3M [15] combined MCTS and recurrent neural network in a self play [28] fashion, however, it seems that Alpha3M does not perform better than the state of the art AutoML systems. For example, out of all the 6 OpenML datasets they have used to compare with state of the art AutoML systems, Alpha3M only shows a clear improvement on 1 dataset (spectf) against Auto-sklearn [16] and TPOT [25], according to Fig. 4 in [15]. Furthermore, it is not clear to us how the hyper-parameters are set and if Bayesian Optimization is used. The implementation of Alpha3M takes advantage of the GPUs [15] for the fast performance while our method has a light weight implementation which efficiently runs with CPU and does not necessarily need Neural Networks.

3 Method

3.1 Towards ReinBo

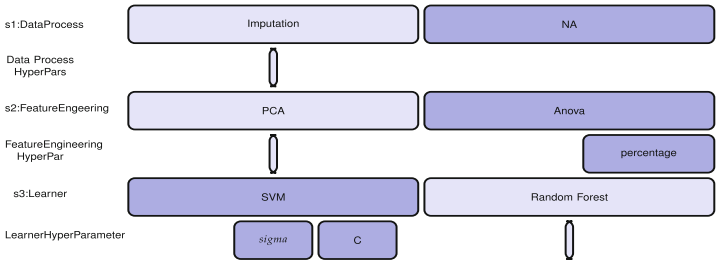


Fig. 2. Illustrative example of selected pipeline and associated hyper-parameters (Color figure online)

As shown in Fig. 2, we assume that a machine learning pipeline potentially consists of 3 stages (s1 through s3 in the figure), which include data preprocessing (imputations, NA and more), feature engineering (Principal Component Analysis for feature transform, Anova for feature filtering and more), and machine learning model selection (learner like SVM, Random Forest). Specifically, we use operation "NA" to indicate that no operation would be done in the stage in question. Figure 2 just serves as a toy but working example for ReinBo, in practice, there are a lot more operations available. A particular operation has associated hyper-parameters (for instance the percentage of selected features in Anova feature filtering). In Fig. 2, dark color filled cells (NA, Anova, SVM)

represent selected operations and their associated active hyper-parameters (percentage, sigma, C), while hyper-parameters for inactive operations are not drawn in the figure.

Observing from Fig. 2, along with Fig. 1, we could think of the pipeline selection and configuration problem as a two-phase process. During the first phase, a planning algorithm guides the agent to choose a path which corresponds to an unconfigured pipeline. This is similar to a multi-armed bandit problem, where each path corresponds to one arm, while difference lies in that the agent can not directly pull a discrete arm but have to pull across several consecutive discrete arm groups (each arm group corresponds to a stage in Fig. 2) and the agent only gets reward after choosing one of arms from the last group. The second phase is similar to contextual bandit with continuous action space (corresponding to hyper-parameters), where the context is which path from the first phase has been selected.

We model the first phase as a reinforcement learning episode, where a particular operation in stage i is treated as action a_i, taken upon corresponding state s_i. State s_i could be represented by actions taken up to the current stage for example. The pipeline search problem is then to find an optimal policy π to decide which operation (action) to take at a particular state. The action value function $Q(s, a)$ at each state tells us how favorable a particular operation is. We use $\mathcal{A}_{s_i}$ to denote the space of legal actions at state s_i. Suppose a roll-out of states trajectory for one composition (episode) is $s_1, \ldots, s_K$, the corresponding space of pipeline could be denoted by $\prod_{i=1}^{K} \mathcal{A}_{s_i}$, where K is the total number of stages and we use $\prod$ to denote the Cartesian Product. For a more general notation, we use $\mathcal{A}(S_i, \Phi_{a_i})$ to denote the space of actions, together with configurable hyper-parameters when the state is S_i at stage i.

We search for potentially better hyper-parameters in the second phase with Bayesian Optimization. Aside from the pipeline itself, each concrete operation (action a_i) at stage i is configurable by a set of hyper-parameters Φ_{a_i}. Φ_{a_i} can be hyper-parameters set for a preprocessor like the ratio of variance to keep in PCA or hyper-parameters set for a machine learning model like the C and σ hyper-parameter for SVM. Thus a configured pipeline search space would be $\prod_{i=1}^{K} \mathcal{A}(S_i; \Phi_{a_i})$ where we use Φ_{a_i} to denote the conditional hyper-parameter space at stage i.

The connection point between reinforcement learning and Bayesian Optimization lies in the reward function design in the reinforcement learning part. During the composition process, there is no signal available to judge how good a current uncompleted pipeline is until the final learner (classifier) is configured with hyper-parameters and trained on the data. At the starting point, different pipelines are tried out randomly, which corresponds to an untrained exploration policy π. A completed pipeline with a joint non-conditional hyper-parameter search space is optimized with Bayesian Optimization for a few steps. The best negative loss is then used as a reward at the end of an episode to guide the reinforcement learning agent towards a better policy. The environment uncertainty only comes with the stochastic reward, while the transition from current state

to next state through action is deterministic. We choose to use Q-learning [32] to optimize the policy where we have tried the Tabular Q-learning and Deep Q-learning [23]. We find out that the Tabular Q-learning works better than Deep Q-learning. For space constraint, the latter is not discussed in detail in this work.

We need Bayesian Optimization to optimize the hyper-parameters in a fine grained level with limited budget, but also want to give budget preference to those promising pipelines. To circumvent the complexity of conditional and hierarchical relationship between hyper-parameters and pipeline, we use reinforcement learning to choose a pipeline and let Bayesian Optimization tune the hyper-parameters. We model the variation of the same pipeline with different hyper-parameters as the environment uncertainty. By using separate surrogate model for each pipeline, we circumvent the risk of mistakenly modeling improper dependent structure between different hyper-parameters, at a minor cost of maintaining those searched pipelines surrogate model as dictionary in memory.

3.2 Connections to Hyperband

The idea of only using a few steps of Bayesian Optimization is inspired by Hyperband [22], where the trade-off between aggressively exploring more configurations and giving each configuration more resources to be validated is solved by grid searching. Instead, in this paper, we do not need the grid search, promising pipelines will get a higher probability to be selected by our reinforcement learning agent which means these pipelines get more chances to be evaluated by the Bayesian Optimization process. The trade-off between exploitation and exploration is naturally resolved by an ϵ-greedy policy, and by annealing ϵ from a large value to a small value, we encourage more exploration at the beginning. Compared to Hyperband, our method selects the budgets allocated for a particular pipeline automatically, the effectiveness of our strategy could then rely on the recent success of reinforcement learning in different areas.

3.3 Connection and Extension to Hierarchical Reinforcement Learning

Hierarchical Reinforcement Learning (hrl) [4] is proposed to tackle the curse of dimensionality in Reinforcement Learning [20]. Although the Option approach [4] is more popular, our method has a close connection to the MAXQ subtask approach [14], which divides a task recursively into subtasks and decompose the value function accordingly. The current version of ReinBo can be treated as a special case of the MAXQ task decomposition, where we have two tasks of pipeline selection and hyper-parameter configuration. However, in the current version, most states are not shared between these two tasks, so there is no need to use MAXQ hrl algorithm to solve the problem. But our method can be naturally extended to a hrl version when our design space of pipeline allow shared state between the two subtasks. We leave it as future work to optimize such complicated pipelines using Hierarchical Reinforcement Learning.

3.4 Procedures of ReinBo

As shown in Algorithm 1, we first initialize a policy π for the agent which can be represented by neural network or a Q-table initialized with certain strategy, coupled with an exploration mechanism like the ϵ-greedy strategy. During the roll-out, initial populations of pipelines get sampled, with the corresponding hyper-parameter space $\Lambda(\prod a_i) = \prod_i \Phi_{a_i}$ to be optimized by Bayesian Optimization for several steps, where Λ means extracting the hyper-parameter set from a pipeline. The corresponding surrogate model is stored in the dictionary $\mathcal{R}$ for future episode if the same pipeline gets rolled out again. The performance of the pipeline on validation data will be used to serve as feedback signal or reward to the reinforcement learning agent to conduct policy iteration.

Algorithm 1. ML ReinBo

Require: dataset $\mathcal{D}$, pipeline operators and hyper-parameters candidates
  Initialize Policy π
  Initialize Surrogate Dictionary $\mathcal{R} \leftarrow \emptyset$ with pipeline as key
  while Budget not reached **do**
    Roll-out an **unconfigured** pipeline $\prod a_i$ according to policy π
    Extract hyper-parameters set for the ground pipeline $\Lambda(\prod a_i) = \prod_i \Phi_{a_i}$
    Reward $R \leftarrow$ BO_PROBE$(\prod a_i, \Lambda, \mathcal{R})$
    Update Policy π with reinforcement learning algorithm with reward R
  end while

Once an unconfigured pipeline is constructed at the end of the episode, running Bayesian Optimization could be beneficial in searching for a more favorable hyper-parameter setting. However, Bayesian Hyperparameter Optimization with large budgets could be rather expensive. Instead, we optimize hyper-parameters for an unconfigured pipeline only for several iterations. For example, we take the number of iterations to be 2 or 3 times the dimension of hyper-parameter space, which means that hyper-parameter spaces with higher dimension will get more sampling budgets. After each episode, the current best configuration's performance for this pipeline in question is used as reward. The next time the same pipeline is sampled, the surrogate model could be retrieved from the dictionary $\mathcal{R}$ to facilitate further search using Bayesian Optimization. We dub the hyper-parameter search process as BO_PROBE, with details shown in Algorithm 2.[2] If an unconfigured pipeline is not sampled yet, an initial design is generated to facilitate an initial surrogate model.

[2] To save budgets, when an unconfigured pipeline does not improve after a number of trials of BO_PROBE, it can also be suspended for future evaluation.

Algorithm 2. BO_PROBE($\prod a_i, \Lambda, \mathcal{R}$)

Require: Surrogate Dictionary $\mathcal{R}$ with pipeline as key
  if $\mathcal{R}\{\prod_i a_i\} = \emptyset$ **then**
    generate initial design of size n^{init} hyper-parameter configurations $\{\phi_j\}_{1:n^{init}}$ for surrogate model with corresponding hyper-parameters set $\Lambda(\prod a_i)$.
    for j in $1 : n^{init}$ **do**
      evaluate the pipeline with ϕ_j to get predicative accuracy y_j
    end for
    initialize surrogate model $\mathcal{R}\{\prod_i a_i\}$ by fitting $\{(\phi_j, 1 - y_j)\}_{1:n^{init}}$
  end if
  for k in $1 : n^{probe}$ **do**
    propose new configuration ϕ_k according to surrogate model $\mathcal{R}\{\prod a_i\}$
    evaluate new configuration to get accuracy y_k to update model $\mathcal{R}\{\prod a_i\}$
  end for
  return $y^* \leftarrow$ best accuracy until now

4 Experiments

4.1 Implementation, Comparison Methods and Setups

Our initial implementation for ReinBo is based on R machine learning packages *mlr* [10], *mlrCPO* [8] for pipeline construction and *mlrMBO* [11] for Bayesian Optimization. The R package *parabox*[3] is implemented for this project to specify conditional hierarchical hyper-parameter space and provides the conditional ancestral sampling (random search in conditional hyper-parameter space). The R package *rlR*[4] is implemented for reinforcement learning where the user could implement a custom environment as input. All python packages are invoked with the R-Python interface *reticulate* [1].

To evaluate the performance of our proposed method, we compare the performance of ReinBo with several state of the art AutoML systems, as well as several conditional hyper-parameter space tuning methods running on top of our R implementation, in order to reduce implementation and search space confounding factors. We compare against Auto-sklearn [16] and TPOT [25] (TPOT with two search spaces to reduce confounding[5]), both based on *scikit-learn* [26]. ML-Plan [24] is not included due to lack of detailed documentation and examples online when experiment is conducted. Additionally, we compare against hyper-parameter optimization techniques which preserve the hierarchical conditional structure, including Tree-structured Parzen Estimator (TPE) [7] used in *Hyperopt* [6], and Random Search with conditional Ancestral Sampling (self implemented in R package *parabox*). Random Search remains a very strong baseline in a lot of machine learning hyper-parameter optimization scenarios [5].

[3] https://github.com/smilesun/parabox.
[4] https://github.com/smilesun/rlR.
[5] We also selected a matching search space of Autosklearn according to Table 1 but still get worse results than Reinbo.

Evaluation Criteria. As warned in [24], many state of the art AutoML systems seem to have missed to deal with the risk of overfitting. Therefore, in the experiment part, we focus on evaluating the generalization capability of the selected pipeline empirically. To avoid any potential confusion from synonyms, we use D^{opt} to represent the part of a dataset fed into a given AutoML system and use D^{test} to represent the locked out part [29] of the same dataset used to test the generalization capacity. The split of D^{opt} and D^{test} is done by Cross Validation, which means for a dataset D, $D = D^{opt} \bigcup D^{test}$ and $D^{opt} \bigcap D^{test} = \emptyset$. To create the D^{opt} and D^{test} split, we use 5-fold cross-validation ($CV5$), which corresponds to the outer loop of Nested Cross Validation (NCV) [13]. We take the aggregated mmce (mean miss-classification error) across the 5-fold iterations over each D^{test} as ultimate performance measure.

As of optimization on D^{opt}, instead of using running time as budget, we use the number of configuration evaluations as the unit of budget, to circumvent effects of hardware and network load variations, etc. For each D^{opt}, we assign a budget of 1000 times of $CV5$ equivalents (5000 times model training) to each AutoML algorithm, which corresponds to the inner loop of NCV [13].

Other Setups. To account for different AutoML systems data input format incompatibility problem, we conduct dummy encoding to categorical features beforehand. Aiming for a light weight implementation, in the experiment, we limit our choice of pipeline components for ReinBo. We compose a pipeline in 3 stages, with potential operations/actions at each stage listed in Table 1. Associated hyper-parameters with an unconfigured pipeline are listed in Table 2. We call the components and associated hyper-parameters the pipeline pool. The same pipeline pool is used for ReinBo, TPE and Random Search.

For Auto-sklearn, Meta-learning and ensemble are disabled, the resampling strategy is set to be $CV5$, stop criteria is changed to budget instead of time and all other configurations are kept default. We have contacted the author of Autosklearn through Github for the right use of the API to ensure the above configuration is satisfied. For TPOT (version 0.9), the default configuration space contains a lot of operators while the light version provides only fast models and pre-processors. The light TPOT is therefore less time-consuming but it could probably lead to lower accuracy in consequence. For this reason, we compare ReinBo with both TPOT with the default configuration and TPOT with light configuration, and we call them TPOT and TPOT-light respectively. TPOT is configured to allow equal amount of budgets with all methods being compared and other configurations are left to be default.

Datasets. We experimented on a set of standard benchmarking datasets of high quality collected from OpenML-CC18[6] [9] and Auto-Weka [30], which are rather well-curated from many thousands and have diverse numbers of classes, features, observations, as well as various ratios of the minority and majority class size. Summary of these datasets is listed in Table 3.

[6] https://www.openml.org/s/99.

Table 1. List of pipeline operations. An operation of "NA" here is used to indicate that no operation would be taken in the corresponding stage. Please refer to *mlrCPO* documentation for the detailed meaning of these operators.

	Stage	Operation/action				
1	DataPreprocess	Scale(default)	Scale(center=FALSE)	Scale(scale=FALSE)	SpatialSign	NA
2	Feature engineering	Pca	FilterKruskal	FilterAnova	FilterUnivariate	NA
3	Classifier	kknn	ksvm	ranger	xgboost	naiveBayes

Table 2. List of hyper-parameters to the operations in Table 1. "p" in the column "Range" indicates the number of features of the original dataset. We invite the user to refer to the R packages *mlrCPO* and *mlr* documentations for the exact meaning of operation, hyper-parameters, etc.

Operation	Parameter	Type	Range
Anonva, Kruskal, Univariate	perc	numeric	(0.1, 1)
Pca	rank	integer	(p/10, p)
kknn	k	integer	(1, 20)
ksvm	C	numeric	$(2^{-15}, 2^{15})$
ksvm	sigma	numeric	$(2^{-15}, 2^{15})$
ranger	mtry	integer	(p/10, p/1.5)
ranger	sample.fraction	numeric	(0.1, 1)
xgboost	eta	numeric	(0.001, 0.3)
xgboost	max_depth	integer	(1, 15)
xgboost	subsample	numeric	(0.5, 1)
xgboost	colsample_bytree	numeric	(0.5, 1)
xgboost	min_child_weight	numeric	(0, 50)
naiveBayes	laplace	numeric	(0.01, 100)

4.2 Experiment Results

In Fig. 3, we compare the mmce (1-Accuracy) of each method with boxplot over the datasets listed in Table 3 across 10 statistical replications. Additionally, we list numerical results in Table 4 with statistical test, where each numerical value represents the aggregated mean mmce over the statistical replications. Underline in each row indicates the smallest mean value over the corresponding dataset. The bold-faced values indicate that the corresponding algorithm does not perform significantly worse than the underlined algorithm on the corresponding dataset based on Mann-Whitney U test. As shown in Table 4, ML-ReinBo has boldfaces for 8 of 10 datasets followed by much less boldfaces from other methods.

In Table 5, we compare win (significantly better), lose and tie (neither significantly better nor worse) relationships according to the test. As shown in Table 5,

Table 3. List of OpenML datasets for experiment. Columns are the OpenML task_id and name, the number of classes (nClass), features (nFeat) and observations (nObs), as well as the ratio of the minority and majority class sizes (rMinMaj).

task_id	Name	nClass	nFeat	nObs	rMinMaj
14	mfeat-fourier	10	77	2000	1.00
23	cmc	3	10	1473	0.53
37	diabetes	2	9	768	0.54
53	vehicle	4	19	846	0.91
3917	kc1	2	22	2109	0.18
9946	wdbc	2	31	569	0.59
9952	phoneme	2	6	5404	0.42
9978	ozone-level-8hr	2	73	2534	0.07
146817	steel-plates-fault	7	28	1941	0.08
146820	wilt	2	6	4839	0.06

ReinBo has won TPOT on 5 datasets and performed worse than TPOT for only one dataset. And not surprisingly, TPOT has performed considerably better than TPOT-light in the empirical experiments since TPOT-light has smaller search space with only fast models and preprocessors. ReinBo also performs much better than Random Search and TPE, where ReinBo has significantly won them on 5 and 6 tasks respectively and lost only on 1 task. Compared to ReinBo, Auto-sklearn has won only once and behaved worse than ReinBo on 3 of 10 datasets.

Table 4. Average performance (mmce) of algorithms across 10 statistical replications with different seeds. In each run the aggregated mmce based over the outer loop of NCV is taken as performance measure for each algorithm. Each value in this table is the mean value of the aggregated mmce values across 10 replications and the best mean value for each dataset is underlined. The bold-faced values indicate that the algorithm does not perform significantly worse than the underlined algorithm on the corresponding dataset based on Mann-Whitney U test.

Dataset	Auto-sklearn	TPE	TPOT	TPOT-light	Random	ReinBo	Underlined algorithm
mfeat-fourier	0.1412	0.1542	0.1451	0.1489	0.1580	**0.1278**	ReinBo
cmc	**0.4470**	**0.4485**	**0.4457**	**0.4506**	**0.4500**	**0.4485**	TPOT
diabetes	0.2483	**0.2436**	0.2452	**0.2413**	**0.2455**	**0.2395**	ReinBo
vehicle	**0.1679**	0.2117	0.1784	0.2057	0.2020	**0.1621**	ReinBo
kc1	0.1421	**0.1351**	**0.1380**	0.1438	**0.1353**	0.1387	TPE
wdbc	**0.0299**	0.0348	0.0353	**0.0264**	0.0341	**0.0271**	TPOT-light
phoneme	**0.0902**	**0.0920**	**0.0893**	0.1016	**0.0912**	**0.0905**	TPOT
ozone-level-8hr	**0.0588**	0.0601	**0.0577**	0.0603	0.0598	**0.0578**	TPOT
steel-plates-fault	0.2041	0.2330	**0.1985**	0.2601	0.2146	0.2141	TPOT
wilt	0.0132	0.0159	0.0141	0.0164	0.0161	**0.0123**	ReinBo

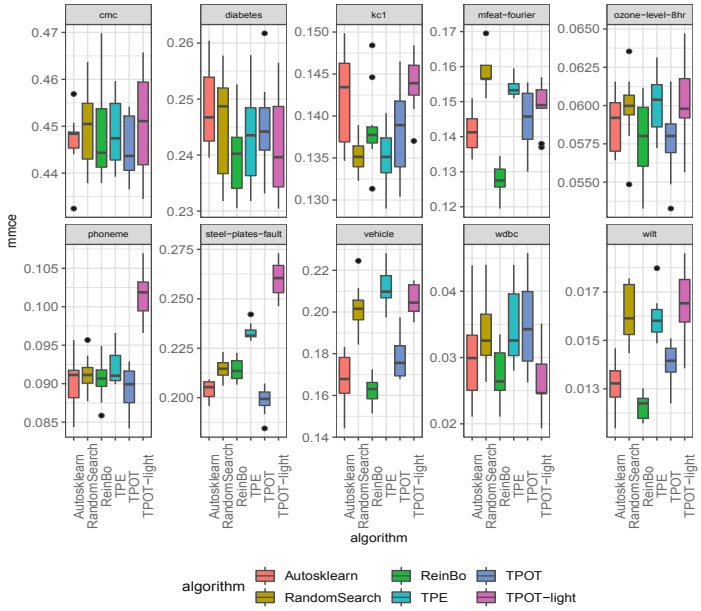


Fig. 3. Boxplots showing the distribution of aggregated mmce achieved by each algorithm within 10 statistical replications.

Meanwhile, ReinBo has comparatively short box ranges in most cases as shown in Fig. 3. Hence, we would conclude that ReinBo performed better and more stably than other algorithms in our empirical experiments. Besides comparing the final performance, it is also interesting to look into the machine learning pipelines suggested by an AutoML system. The frequencies of the operators in the pipelines suggested by ReinBo are listed in Table 6.

Running Time. Figure 4 shows the average running time each algorithm takes to complete one experiment, which corresponds to a Nested Cross Validation (NCV) process. It can be seen that Auto-sklearn is the most time-consuming

Table 5. Win-Lose-Tie comparison between ReinBo and other algorithms on benchmarking datasets based on Mann-Whitney U test (significance level $\alpha = 0.05$).

		Random_search	TPE	Auto-sklearn	TPOT-light	TPOT
ReinBo	Win	5	6	3	7	5
	Tie	4	3	6	3	4
	Lose	1	1	1	0	1

Table 6. Frequency of operators suggested by ReinBo. During empirical experiments there are 500 pipelines in total suggested by ReinBo at the end of optimization process. The frequency (Freq.) and relative frequency (Relative freq.) of each operator selected in best pipelines are shown here.

Preprocess	Freq.	Relative freq.	Feature engineering	Freq.	Relative freq.	Classifier	Freq.	Relative freq.
Scale(default)	259	51.8%	FilterAnova	210	42.0%	ksvm	276	55.2%
Scale(scale=FALSE)	106	21.2%	FilterKruskal	139	27.8%	ranger	201	40.2%
Scale(center=FALSE)	67	13.4%	PCA	63	12.6%	kknn	12	2.4%
NA	36	7.2%	Univariate	46	9.2%	xgboost	10	2.0%
SpatialSign	32	6.4%	NA	42	8.4%	naiveBayes	1	0.2%

algorithm in our empirical experiments. Although TPOT-light is the fastest algorithm, it resulted in worse performance because it contains only fast operators. Our proposed ReinBo algorithm spent less time than Random Search and state of the art AutoML systems TPOT and Auto-sklearn in average.

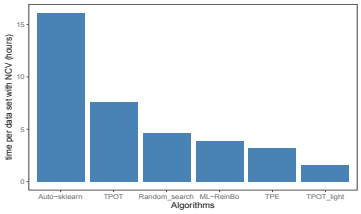


Fig. 4. Comparison of average running time of each algorithm per data set with *NCV*

5 Summary and Future Work

We present a new AutoML algorithm ReinBo by embedding Bayesian Optimization into Reinforcement Learning. The Reinforcement Learning takes care of pipeline composition, and Bayesian Optimization takes care of configuring the hyper-parameters associated with the composed pipeline. ReinBo is inspired by Hyperband and previous efforts in AutoML by considering the trade-off of assigning resources to a particular configuration and exploring more configurations as a reinforcement learning problem, where the learned policy solves the trade-off automatically. Experiments show our method has a considerable improvement compared to other state of the art systems and methods. For future work, it would be interesting to include meta learning into our system, which does not only learn how to construct a pipeline and configure it for a dataset in question, but also how to generalize the learned policy to a wide range of datasets by learning jointly on the meta features. Additionally, it would be nice to see how ReinBo performs on jointly optimizing neural architecture and continuous

hyper-parameters like learning rate and momentum, as well as applications like Computer Vision [19] and semantic web based Recommendation Systems [21] where pipeline might play a role. Multi-Objective Bayesian Optimization [17] for hyper-parameter tuning would also be future direction.

Acknowledgement. Janek Thomas gave us many helpful suggestions, Martin Binder and Florian Pfisterer helped us with mlrCPO and auto-sklearn setup.

References

1. Allaire, J., Ushey, K., Tang, Y.: Reticulate: interface to 'Python' (2019). https://CRAN.R-project.org/package=reticulate. R package version 1.11.1
2. Baker, B., Gupta, O., Naik, N., Raskar, R.: Designing neural network architectures using reinforcement learning. In: International Conference on Learning Representations (2017)
3. Banzhaf, W., Nordin, P., Keller, R.E., Francone, F.D.: Genetic Programming: An Introduction, vol. 1. Morgan Kaufmann, San Francisco (1998)
4. Barto, A.G., Mahadevan, S.: Recent advances in hierarchical reinforcement learning. Discrete Event Dyn. Syst. **13**(1–2), 41–77 (2003)
5. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. J. Mach. Learn. Res. **13**(Feb), 281–305 (2012)
6. Bergstra, J., Yamins, D., Cox, D.D.: Hyperopt: a Python library for optimizing the hyperparameters of machine learning algorithms. In: Proceedings of the 12th Python in Science Conference, pp. 13–20. Citeseer (2013)
7. Bergstra, J.S., Bardenet, R., Bengio, Y., Kégl, B.: Algorithms for hyper-parameter optimization. In: Advances in Neural Information Processing Systems, pp. 2546–2554 (2011)
8. Binder, M.: mlrCPO: composable preprocessing operators and pipelines for machine learning (2019). https://CRAN.R-project.org/package=mlrCPO. R package version 0.3.4-2
9. Bischl, B., et al.: OpenML benchmarking suites and the openml100. arXiv preprint arXiv:1708.03731 (2017)
10. Bischl, B., et al.: mlr: machine learning in R. J. Mach. Learn. Res. **17**(170), 1–5 (2016). http://jmlr.org/papers/v17/15-066.html
11. Bischl, B., Richter, J., Bossek, J., Horn, D., Thomas, J., Lang, M.: mlrMBO: a modular framework for model-based optimization of expensive black-box functions. arXiv preprint arXiv:1703.03373 (2017)
12. Brochu, E., Cora, V.M., De Freitas, N.: A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. arXiv preprint arXiv:1012.2599 (2010)
13. Cawley, G.C., Talbot, N.L.: On over-fitting in model selection and subsequent selection bias in performance evaluation. J. Mach. Learn. Res. **11**(Jul), 2079–2107 (2010)
14. Dietterich, T.G.: Hierarchical reinforcement learning with the MAXQ value function decomposition. J. Artif. Intell. Res. **13**, 227–303 (2000)
15. Drori, I., et al.: AlphaD3M: machine learning pipeline synthesis. In: AutoML Workshop at ICML (2018)

16. Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., Hutter, F.: Efficient and robust automated machine learning. In: Advances in Neural Information Processing Systems, pp. 2962–2970 (2015)
17. Horn, D., Dagge, M., Sun, X., Bischl, B.: First investigations on noisy model-based multi-objective optimization. In: Trautmann, H., et al. (eds.) EMO 2017. LNCS, vol. 10173, pp. 298–313. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-54157-0_21
18. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Coello, C.A.C. (ed.) LION 2011. LNCS, vol. 6683, pp. 507–523. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25566-3_40
19. Zhang, J., Guo, L., Yang, S., Sun, X., Li, X.: Detecting Chinese calligraphy style consistency by deep learning and one-class SVM. In: 2017 2nd International Conference on Image, Vision and Computing (ICIVC), pp. 83–86. IEEE (2017)
20. Kulkarni, T.D., Narasimhan, K., Saeedi, A., Tenenbaum, J.: Hierarchical deep reinforcement learning: integrating temporal abstraction and intrinsic motivation. In: Advances in Neural Information Processing Systems, pp. 3675–3683 (2016)
21. Kushwaha, N., Sun, X., Singh, B., Vyas, O.: A lesson learned from PMF based approach for semantic recommender system. J. Intell. Inf. Syst. **50**(3), 441–453 (2018)
22. Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., Talwalkar, A.: Hyperband: a novel bandit-based approach to hyperparameter optimization. J. Mach. Learn. Res. **18**(1), 6765–6816 (2017)
23. Mnih, V., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529 (2015)
24. Mohr, F., Wever, M., Hüllermeier, E.: ML-plan: automated machine learning via hierarchical planning. Mach. Learn. **107**(8–10), 1495–1515 (2018)
25. Olson, R.S., Moore, J.H.: TPOT: a tree-based pipeline optimization tool for automating machine learning. In: Workshop on Automatic Machine Learning, pp. 66–74 (2016)
26. Pedregosa, F., et al.: Scikit-learn: machine learning in Python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
27. de Sá, A.G.C., Pinto, W.J.G.S., Oliveira, L.O.V.B., Pappa, G.L.: RECIPE: a grammar-based framework for automatically evolving classification pipelines. In: McDermott, J., Castelli, M., Sekanina, L., Haasdijk, E., García-Sánchez, P. (eds.) EuroGP 2017. LNCS, vol. 10196, pp. 246–261. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-55696-3_16
28. Silver, D., et al.: Mastering the game of go without human knowledge. Nature **550**(7676), 354 (2017)
29. Sun, X., Bommert, A., Pfisterer, F., Rähenfürher, J., Lang, M., Bischl, B.: High dimensional restrictive federated model selection with multi-objective Bayesian optimization over shifted distributions. In: Bi, Y., Bhatia, R., Kapoor, S. (eds.) IntelliSys 2019. AISC, vol. 1037, pp. 629–647. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-29516-5_48
30. Thornton, C., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Auto-WEKA: automated selection and hyper-parameter optimization of classification algorithms. CoRR abs/1208.3719 (2012). http://arxiv.org/abs/1208.3719
31. Thornton, C., Leyton-Brown, K.: Auto-WEKA: automated selection and hyper-parameter optimization of classification algorithms (2012)
32. Watkins, C.J., Dayan, P.: Q-learning. Mach. Learn. **8**(3–4), 279–292 (1992)

33. Yang, F., Gustafson, S., Elkholy, A., Lyu, D., Liu, B.: Program search for machine learning pipelines leveraging symbolic planning and reinforcement learning. In: Banzhaf, W., Spector, L., Sheneman, L. (eds.) Genetic Programming Theory and Practice XVI. GEC, pp. 209–231. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-04735-1_11
34. Yang, F., Lyu, D., Liu, B., Gustafson, S.: PEORL: integrating symbolic planning and hierarchical reinforcement learning for robust decision-making. arXiv preprint arXiv:1804.07779 (2018)
35. Zoph, B., Le, Q.V.: Neural architecture search with reinforcement learning. arXiv preprint arXiv:1611.01578 (2016)

Appendix G

Benchmarking time series classification-Functional data vs machine learning approaches

Contributing Article Pfisterer, Florian; Beggel, Laura; Sun, Xudong; Scheipl, Fabian; Bischl, Bernd; Benchmarking time series classification-Functional data vs machine learning approaches, arXiv preprint arXiv:1911.07511, 2019.

Author Contributions Bernd Bischl initiated the project, with assistance on the first architecture design and data flow implementation by Xudong Sun and Laura Beggel, while the current version of the architecture of the software is refactored and implemented by Florian Pfisterer and Bernd Bischl. Laura Beggel independently provided the datasets repository for the benchmark and gathered the results from the benchmark experiments of Bagnall et al. (2017). Laura Beggel implemented the first version of several classification algorithms for mlrFDA, followed by refactor of the code by Florian Pfisterer. Xudong Sun implemented the first version of several functional feature extraction algorithms used in mlrFDA, including bsignal, Dynamic Time Warping (multiple reference input), multiresolution (proposed by Bernd Bischl with major code snippet), Florian Pfisterer refactored these codes and added several new feature extraction methods. Xudong Sun implemented the first version of classification and regression of FDboost, FGAM, with the help of Fabian Scheipl, followed by refactor by Florian Pfisterer. Laura Beggel implemented the first version of Fourier and Wavelets feature extraction, Florian Pfister and Xudong Sun refactored the codes. Xudong Sun implemented the first version of the Benchmark code and conducted a first batch of experiments, with Laura Beggel analyzing the results. Florian Pfisterer refactored the benchmark code and rerunned the experiment and collected the experiment results independently. The paper writing consist of joint work with all 3 Phd candidates and 2 senior authors, with each person focusing on different parts of the paper,

while Florian Pfisterer takes a leading role. Florian Pfisterer and Laura Beggel created most of the plotting codes used in the paper, with minor refinement from Xudong Sun.

Benchmarking time series classification - Functional data vs machine learning approaches

Florian Pfisterer, Xudong Sun[1], Laura Beggel[1], Fabian Scheipl, Bernd Bischl

Department of Statistics, Ludwig-Maximilians-Universität München
Ludwigstr. 33,
80539, München, Germany

Abstract

Time series classification problems have drawn increasing attention in the machine learning and statistical community. Closely related is the field of functional data analysis (FDA): it refers to the range of problems that deal with the analysis of data that is continuously indexed over some domain. While often employing different methods, both fields strive to answer similar questions, a common example being classification or regression problems with functional covariates. We study methods from functional data analysis, such as functional generalized additive models, as well as functionality to concatenate (functional) feature extraction or basis representations with traditional machine learning algorithms like support vector machines or classification trees. In order to assess the methods and implementations, we run a benchmark on a wide variety of representative (time series) data sets, with in-depth analysis of empirical results, and strive to provide a reference ranking for which method(s) to use for non-expert practitioners. Additionally we provide a software framework in R for functional data analysis for supervised learning, including machine learning and more linear approaches from statistics. This allows convenient access, and in connection with the machine-learning toolbox *mlr*, those methods can now also be tuned and benchmarked.

☆source code available at `https://github.com/mlr-org/mlr`

*Corresponding author: florian.pfisterer@stat.uni-muenchen.de

[1]Equal Contribution

Keywords: Functional Data Analysis, Time Series, Classification, Regression

1. Introduction

The analysis of functional data is becoming more and more important in many areas of application such as medicine, economics, or geology (cf. Ullah and Finch [1], Wang et al. [2]), where this type of data occurs naturally. In industry, functional data are often a by-product of continuous monitoring of production processes, yielding great potential for data mining tasks. A common type of functional data are time series, as time series can often be considered as discretized functions over time.

Many researchers publish software implementations of their algorithms, therefore simplifying the access to already established methods. Even though such a readily available, broad range of methods to choose from is desirable in general, it also makes it harder for non-expert users to decide which method to apply to a problem at hand and to figure out how to optimize their performance. As a result, there is an increasing demand for automated model selection and parameter tuning.

Furthermore, the functionality of available pipeline steps ranges from simple data structures for functional data, to feature extraction methods and packages offering direct modeling procedures for regression and classification. Users are again faced with a multiplicity of software implementations to choose from and, in many instances, combining several implementations may be required. This can be difficult and time-consuming, since the various implementations utilize a multiplicity of different workflows which the user needs to become familiar with and synchronize in order to correctly carry out the desired analysis.

There is a wide variety of packages for functional data analysis in R [3] available that provide functionality for analyzing functional data. Examples range from the **fda** [4] package which includes object types for functional data and allows for smoothing and simple regression, to, e.g., boosted additive regression models for functional data in **FDboost** [5]. For an extensive overview, see the CRAN task view [6].

Many of those packages are designed to provide algorithmic solutions for one

specific problem, and each of them requires the user to become familiar with its user interface. Some of the packages, however, such as **fda.usc** [7] or **refund** [8] are not designed for only one specific analysis task, but combine several approaches. Nevertheless, these packages do not offer unified frameworks or consistent user interfaces for their various methods, and most of the packages can still only be applied separately.

A crucial advantage of providing several algorithms in one package with a unified and principled user interface is that it becomes much easier to compare the provided methods with the intention to find the best solution for a problem at hand. But to determine the best alternative, one still has to be able to compare the methods at their best performance on the considered data, which requires hyperparameter search and, more preferably, efficient tuning methods.

While the different underlying packages are often difficult and sometimes even impossible to extend to new methods, custom implementations and extensions can be easily included in the accompanying software.

We want to stress that the focus of this paper does not lie in proposing new algorithms for functional data analysis. Its added value lies in a large comparison of algorithms while providing a unified and easily accessible interface for combining statistical methods for functional data with the broad range of functions provided by **mlr**, most importantly benchmarking and tuning. Additionally, the often overlooked possibility of extracting non-functional features from functional data is integrated, which enables the user to apply classical machine learning algorithms such as *support vector machines* [9] to functional data problems.

In a benchmark study similar to Bagnall et al. [10] and Fawaz et al. [11], we explore the performance of implemented methods, and try to answer the following questions:

1. Can functional data problems be solved with classical machine learning methods ignoring the functional structure of the data as well as with more elaborate methods designed for this type of data? Bagnall et al. [10]

measure the performance of some non-functional-data-specific algorithms such as the rotation forest [12], but this does not yield a complete picture.

2. Guidance on the wide range of available algorithms is often hard to obtain. We aim to make some recommendations in order to simplify the choice of learning algorithm.
3. Do statistical methods explicitly tailored to the analysis of functional data [e.g. **FDboost**, 5] perform well on classical time series tasks? No benchmark results for these methods, which provide interpretable results, are currently available.
4. Many methods that represent functional data in a non-functional domain have been proposed and are also often applied in practice. Examples for this include either hand crafted features [cf. 13], summary statistics [14], or generally applicable methods such as wavelet decomposition [15].
5. Hyperparameter optimization is a very important step in many machine learning applications. In our benchmark, we aim to quantify the impact of hyperparameter optimization for a set of given algorithms on several data sets.

Contributions. As contributions of this paper, we aim to answer the questions posed above. Additionally, we provide a toolbox for the analysis of functional data. It implements several methods for feature extraction and directly modeling functional data, including a thorough benchmark of those algorithms. This toolbox also allows for full or partial replication of the conducted benchmark comparison.

2. Related Work

In the remainder of the paper, we focus on comparing algorithms from the functional data analysis and the machine learning domain. Functional data analysis traditionally values interpretable results and valid statistical inference over prediction quality. Therefore functional data algorithms are often not compared with respect to their predictive performance in literature. We aim to close this gap. On the other hand, machine learning algorithms often do not yield interpretable results. While we consider both aspects to be important, we want to focus on predictive performance in this paper.

2.1. Feature extraction and classical machine learning methods

In this work, we differentiate between machine learning algorithms that can directly be applied to functional data, and algorithms intended for scalar features, which we call *classical* machine learning methods.

A popular approach when dealing with functional data is to reduce the problem to a non-functional task by extracting relevant non-functional features [1]. Applying classical machine learning methods after extracting meaningful features can then lead to competitive results [cf. 16, e.g.] or at least provide baselines, which are in general not covered by functional data frameworks. In our framework, such functionality is easily available by combining feature extraction, e.g., based on extracting heuristic properties [cf. **tsfeatures**; 14] or wavelet coefficients [17, 15] and analyzing these derived scalar features with classical machine learning tools provided by **mlr**.

Based on some existing functionality of the listed packages, we adapt different feature extraction methods. Along with different algorithms already proposed in literature, we propose two new custom methods, *DTWKernel* and *MultiResFeatures*:

> **tsfeatures** [14] extracts scalar features, such as auto-correlation functions, entropy and other heuristics from a time series.

fourier transforms data from the time domain into the frequency domain using the *fast fourier transform* [18]. Extracted features are either phase or amplitude coefficients.

bsignal B-Spline representations from package *FDboost* [5] are used as feature extractors. Given the knots vector and effective degree of freedom, we extract the design matrix for the functional data using *mboost*.

wavelets [15] applies a discrete wavelet transform to time series or functional data, e.g., with Haar or Daubechies wavelets. The extracted features are wavelet coefficients at several resolution levels.

PCA projects the data on their principal component vectors. Only a subset of the principal component scores representing a given proportion of signal variance is retained.

DTWKernel computes the dynamic time warping distances of functional or time series data to (a set of) reference data. We implement *dynamic time warping* (DTW) based feature extraction. This method computes the dynamic time warping distance of each observed function to a (user-specified) set of reference curves. The distances of each observation to the reference curves is then extracted as a vector-valued feature. The reference curves can either be supplied by the user, e.g., they could be several typical functions for the respective classes, or they can be obtained from the training data. In order to compute *dynamic time warping distances*, we use a fast dynamic time warping [19] implementation from package **rucrdtw** [20].

MultiResFeatures extracts features, such as the mean at different levels of resolution (zoom-in steps). Inspired by the image pyramid and wavelet methods, we implement a feature extraction method, *multi-resolution feature extraction* where we extract features like mean and variance computed over specified windows of varying widths. Starting from the full sequence, the sequence is repeatedly divided into smaller pieces, where at

each resolution level, a scalar value is extracted. All extracted features are concatenated to form the final feature vector.

2.2. Methods for functional data

Without feature extraction, direct functional data modeling (both classification and regression) methods incorporated in our package span two families: The first family of semi-parametric approaches includes FGAM [8], FDboost [5], and the functional generalized linear model [FGLM; 21], which are all structured additive models. Those methods use (tensor product) spline basis functions or functional principal components (FPCs) [22] to represent effects fitted in a generalized additive model. While FGAM and FGLM use the iterated weighted least square (IWLS) method to generate maximum likelihood estimates, FDboost uses component-wise gradient boosting [23] to optimize the parameters. Additionally, the estimated parameters can be penalized. A general formula for this family of methods is $\zeta(Y|X = x) = h(x) = \sum_{j=1}^{J} h_j(x)$, where ζ represents a functional of the conditional response distribution (e.g., an expectation or a quantile), x is a vector of (functional) covariates and $h_j(x)$ are partial additive effects of subsets of x in basis function representation, cf. Greven and Scheipl [24] for a general introduction.

The second family of methods are non-parametric methods as introduced in Ferraty and Vieu [25], e.g., based on (semi-)metrics which quantify local or global differences or distances across curves. For example, the distance between two instances could be defined by the L_2 distance of two curves $d(x_i(t), x_j(t)) = \sqrt{\int (x_i(t) - x_j(t))^2 dt}$. Kernel functions are used to average over the training instances and weigh their respective contributions based on the value of their distance semi-metric to the predicted instance. Functional k-nearest neighbors algorithms can also be defined based on such semi-metrics. Implementations can be found in packages **fda.usc** [7] and **classiFunc** [26].

2.3. Toolboxes for functional data analysis

The package **fda** [4] contains several object types for functional data and allows for smoothing and regression for functional data. Analogously, the R-

package **fda.usc** [7] contains several classification algorithms that can be used with functional data. In Python, **scikit-fda** [27] offers both representation of and (pre-)processing methods for functional data, but only a very small set of machine learning methods for classification or regression problems is implemented at the time of writing.

As a byproduct of the Time-Series Classification Bake-off [10], a wide variety of algorithms were implemented and made available. But this implementation emphasizes the benchmark over providing a data analysis toolbox for users, and is therefore not easily usable for inexperienced users.

2.4. Benchmarks

The recently published benchmark analysis **Time-Series Classification Bake-off** by Bagnall et al. [10] provides an overview of the performance of 18 state-of-the-art algorithms for time series classification. They re-implement (in Java) and compare 18 algorithms designed especially for time series classification on 85 benchmark time series data sets from Bagnall et al. [28]. In their analysis, they also include results from several standard machine learning algorithms. They note that the *rotation forest* [12] and *random forest* [29] are competitive with their time series classification baseline [1-nearest neighbor with dynamic time warping distance; 30]. Their results show that ensemble methods such as *collection of transformation ensembles* [COTE; 31] perform best, but for the price of considerable runtime.

Deep learning methods applied to time series classification tasks have also shown competitive prediction power. For example, [11] provide a comprehensive review of state-of-the-art methods. The authors compared both generative models and discriminative models, including *fully connected neural networks, convolutional neural networks, auto-encoders* and *echo state networks*, whereas only discriminative end-to-end approaches were incorporated in the benchmark study.

The benchmark study conducted in this work does not aim to replicate or compete with earlier studies like [10], but instead tries to extend their results.

3. Functional Data

In contrast to non-functional data analysis, where the measurement of a single observation is a vector of scalar components whose entries represent values of the separate multidimensional features, functional data analysis treats and analyses the features themselves as functions over their domain. By learning to represent the underlying function, the carried out analysis is not just restricted to the measured discrete values but it is possible to sample from (and analyze) the entire domain space.

In this work, we focus on pairs of features and corresponding labels (x, y) for supervised learning. In contrast to non-functional data analysis, where the measurement of a single observation is a vector of scalar components, functional features are function-valued over their domain. The features $x = (x_1, ..., x_p)$ can thus also be a function, i.e., $x_j = g_j(t)$, $g : T \to \mathbb{R}$. In practice, functional data comes in the form of observed values $g_j(t), t \in \{1, ..., L\}$, where each t corresponds to a discrete point on the continuum. Those observed values stem from an underlying function f evaluated over a set of points. A frequent type of functional data is time series data, i.e., measurements of a process measured at discrete time-points.

For example, in some electrical engineering applications, signals are obtained over time at a certain sampling rate, but other domains are possible as well. Spectroscopic data, for example, are functional data recorded over certain parts of the electromagnetic spectrum. One such example is depicted in Figure 1. It shows spectroscopy data of fossil fuels [32] where the measured signal represents reflected energies in the ultraviolet-visible (UV-VIS) and the near infrared spectrum (NIR). In the plot, different colors correspond to different instances. This is a typical example of a scalar-on-function regression problem, where the inputs are a collection of spectroscopic curves for a fuel, and the prediction target is the heating value of the fossil fuel.

In Figure 2, we display two functional classification scenarios. The goal in those scenarios is to distinguish the class type of the curve, which can also

be understood as a function-on-scalar problem. Figure 2a shows the vertical position of an actor's hand while either drawing a toy-gun and aiming at a target, or just imitating the motion with the blank hand. This position is measured over time. The two different types of classes of the curves can be distinguished by the color scheme.

Figure 2b shows a data set built for distinguishing images of beetles from images of flies based on their outlines. While following the outline, the distance to the center of the object is measured which is then used for classification purposes. The latter data sets are available from [28].

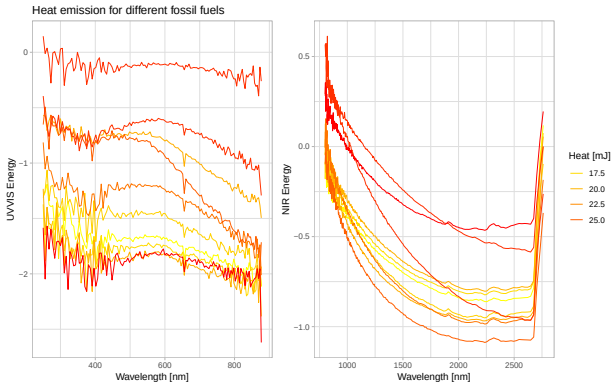
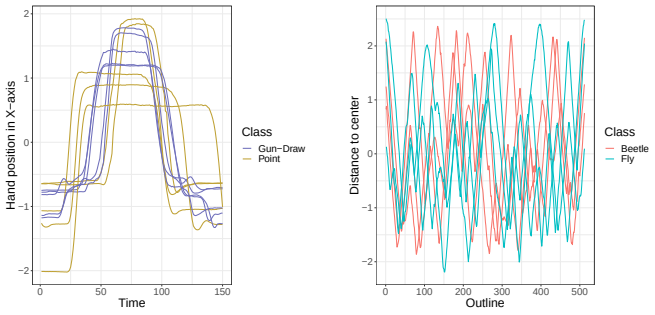


Figure 1: Scalar-on-function regression: Spectral data for fossil fuels [32]

The interested reader is referred to Ramsay [21] and Kokoszka and Reimherr [35] for more in-depth introductions to this topic.

4. Functional Data Analysis with mlrFDA

Along with the benchmark, we implement the software **mlrFDA**, which extends the popular machine learning framework **mlr** for the analysis of functional data. As the implemented functionality is an extension of the **mlr** package, all of the functionality available in **mlr** transfers to the newly added methods for



(a) Gunpoint (b) BeetleFly

Figure 2: Excerpts from two time series classification data sets. (a): Gunpoint data [33], (b): BeetleFly Data [34].

functional data analysis. We include a brief overview of the implemented functionality in Appendix A.1. A more in-detail overview and tutorial on **mlr** can be found in the mlr tutorial [36].

mlr provides a unified framework for machine learning methods in R, currently supporting *tasks* from 4 main problem types: (multilabel-)classification, regression, cluster analysis, and survival analysis. For each problem type, many algorithms (called *learners*) are integrated. This yields an extensive set of modeling options with a unified, simple interface. Moreover, advanced techniques such as hyperparameter tuning, preprocessing and feature selection are also part of the package. An additional focus lies on extensibility, allowing the user to integrate their own algorithms, performance measures and preprocessing methods. As **mlrMBO** seamlessly integrates into the new software, many different tuning procedures can be readily adapted by the user. Tuning of hyperparameters is usually not integrated in software packages for functional data analysis and thus would require the user to write additional, non-trivial code that handles (nested) resampling, evaluation and optimization methods.

mlrFDA contains several functional data algorithms from several R packages, e.g., **fda.usc**, **refund** or **FDboost**. The algorithms' functionality, however,

remains unchanged, only their user interface is standardized for use with **mlr**. For detailed insights into the respective algorithms, full documentation is available in the respective packages.

Since our toolbox is built on **mlr**'s extensible class system, our framework is easily extensible to other methods that have not yet been integrated, and the user can include his or her own methods which do not necessarily need to be available as a packaged implementation. Additionally, **mlrFDA** inherits **mlr**'s functionality for performance evaluation and benchmarking, along with extensive and advanced (hyperparameter) tuning. This makes our platform very attractive for evaluating which algorithm fits best to a problem at hand, and even allows for large benchmark studies.

5. Benchmark Experiment

In order to enable a comparison of the different approaches, an extensive benchmark study is conducted. This paper does not aim to replicate or reproduce results obtained by Bagnall et al. [10] or Fawaz et al. [11]. Instead we focus on providing a benchmark complementary to previous benchmarks. This is done because *i*) the experiments require large amounts of computational resources, and *ii*) the added value of an exact replication of the experiments (with open source code) is comparatively small. Nonetheless, we aim for results that can be compared, and thus extend the results obtained by Bagnall et al. [10] by staying close to their setup. The experiments were carried out on a high performance computing cluster, supported by the Leibniz Rechenzentrum Munich. Individual runs were allowed up to 2.2 GB of RAM and 4 hours run-time for each evaluation. We want to stress that this benchmark compares *implementations*, which does not always necessarily correspond to the performance of the corresponding theoretical *algorithm*. Additionally, methods for functional data analysis are traditionally more focused on valid statistical inference and interpretable results, which does not necessarily coincide with high predictive performance.

5.1. Benchmark Setup

Data sets	51 Data sets, see table B.7
Algorithms	Function (Package)
Machine Learning:	- `glmnet` (**glmnet**) - `rpart` (**rpart**) - `ksvm`* (**kernlab**) - `ranger`* (**ranger**) - `xgboost`* (**xgboost**)
Functional Data	- `classif.knn`(**fda.usc**) - `classif.glm` (**fda.usc**) - `classif.np` (**fda.usc**) - `classif.kernel`(**fda.usc**) - `FDboost` (**FDboost**) - `fgam` (**refund**) - `knn with dtw` (**classiFunc**)
Feature Extraction + ML	- feature extraction: see table A.6 - in combination with ML algorithms marked with a *.
Measures	mean misclassification error, training time
Resampling	20-fold stratified sub-sampling; class balances and train/test set size as in [10].
Tuning	100 iterations of Bayesian optimization (3-fold inner CV). Corresponding hyperparameter-ranges can be obtained from tables 3 and 5.

Table 1: Benchmark experiment setup

A benchmark experiment is defined by four important characteristics: The *data sets* algorithms are tested on, the *algorithms* to be evaluated, the *measures* used for evaluating predictive performance, and a *resampling strategy* used for generating train and test splits of the data. A comprehensive overview of the conducted benchmark setup can be obtained from Table 1.

These characteristics are briefly described subsequently before providing and discussing the results. We use a subset of 51 data sets from the popular UCR archive [28] in order to enable a comparison of results in [10] with the additional methods described in this paper. The data sets stem from various application types such as ECG measurements, sensor data, or image outlines, therefore having varying training set sizes or measurement lengths. For more detailed information about the data sets, see Bagnall et al. [28].

We selected data using the following criteria: In order to reduce the computational resources we did *i*) not run data sets that have multiple versions,

ii) exclude data sets with less then 3 examples in each class *iii*) remove data sets with more than 10000 instances or time series longer than 750 measurements. As some of the classifiers only work with multi-class targets via *1-vs-all* classification, we *iv*) additionally excluded data sets with more then 40 classes. In essence, we benchmark small and medium sized data sets with a moderate amount of different classes.

We add 7 new algorithms and 6 feature extraction methods which can be combined with arbitrary machine learning methods for scalar features (c.f. Table 1). Additionally we test 5 classical machine learning methods, in order to obtain a broader perspective on expected performance if the functional nature of the data is ignored. As we benchmark default settings as well as tuned algorithms, in total 80 different algorithms are evaluated across all data sets. When combining feature extraction and machine learning methods, we fuse the learning algorithm and the preprocessing, thus treating them as a pipeline where data is internally transformed before applying the learner. This allows us to jointly tune the hyperparameters of learning algorithm and preprocessing method. The respective defaults and parameter ranges can be obtained from Table 3 (feature extractors) and Table 5 (learning algorithms). More detailed description of the hyperparameters can be obtained from the respective packages documentation.

In order to generate train/test splits, and thus obtain an unbiased estimate of the algorithm's performance, we use stratified sub-sampling. We use 20 different train/test splits for each data set in order to reduce variance and report the average. For tuned models, we use use nested cross-validation [37] to ensure unbiased estimates, where the outer loop is again subsampling with 20 splits, and the inner resampling for tuning is a 3-fold (stratified) cross-validation. All compared 80 algorithms are presented exactly the same index sets for the 20 train-test outer subsampling splits.

Mean misclassification error (MMCE) is chosen as a measure of predictive performance in order to stay consistent with Bagnall et al. [10]. Other measures, such as area under the curve (AUC) require predicted probabilities and do not trivially extend to multi-class settings.

While Bagnall et al. [10] tune all algorithms across a carefully handcrafted grid, we use *Bayesian optimization* [38]. In order to stay comparable, we analogously fix the amount of tuning iterations to 100.

We use **mlrMBO** [39] in order to perform Bayesian optimization of the hyperparameters of the respective algorithm. Additionally, in order to scale the method to a larger amount of data sets and machines, the R-package **batchtools** (Bischl et al. [40], Lang et al. [41]) is used. This enables running benchmark experiments on high-performance clusters. For the benchmark experiment, a job is defined as re-sampling of a single algorithm (or tuning thereof) on a single version of a data set. This allows for parallelization to an arbitrary number of CPU's, while at the same time guaranteeing reproducibility. The code for the benchmark is available from `https://github.com/compstat-lmu/2019_fda_benchmark` for reproducibility.

5.2. Results

This Section tries to answer the questions posed in section 1. We evaluate *i*) various machine learning algorithms in combination with feature extraction, *ii*) classical time series classification approaches, *iii*) the effect of tuning hyperparameters for several methods, and *iv*) try to give recommendations with respect to which algorithm(s) to choose for new classification problems.

Algorithms evaluated in this benchmark have been divided into three groups: Algorithms specifically tailored to functional data, *classical* machine learning algorithms without feature extraction and *classical* machine learning algorithms in combination with feature extraction.

5.2.1. Algorithms for functional data

Performances of algorithms specifically tailored to functional data analysis can be obtained from Figure 3. The *k-nearest neighbors* algorithm from package **classiFunc** [26] in combination with dynamic time warping [19] distance seems to perform best across data sets. It is also considered a *strong baseline* in Bagnall et al. [10].

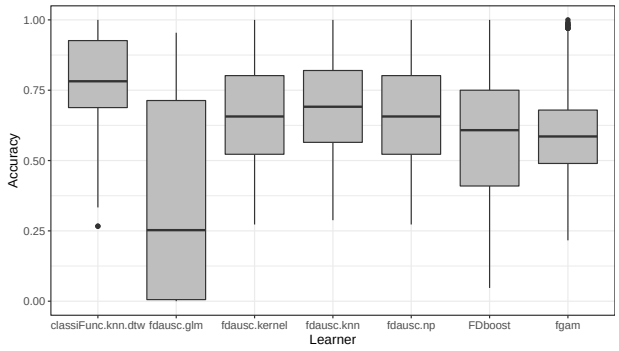


Figure 3: Performances for functional data analysis algorithms in default settings (untuned) across all 51 data sets.

5.2.2. Machine Learning algorithms with feature extraction

Performances of different machine learning algorithms in combination with feature extraction with and without tuning can be obtained from Figure 4.

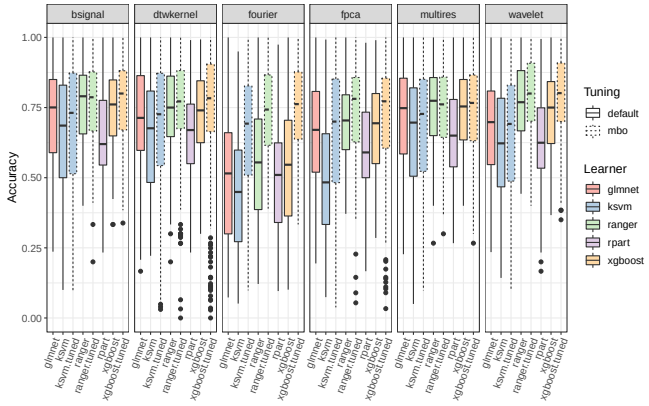


Figure 4: Results for feature extraction-based machine learning algorithms with default and tuned (MBO) hyperparameters across 51 data sets. Hyperparameters are tuned jointly for learner and feature extraction method.

We conclude that feature extraction using splines (bsignal) and wavelets as well as extracting dynamic time warping distances works well when combined with conventional machine learning algorithms, even at their default hyper-parameters. Among the learners, random forests, especially in combination with *bsignal* show quite advantageous performance. In addition, we find an obvious improvement from hyper-parameter tuning for the Fourier feature extraction. In terms of learners, random forest and gradient boosted tree learners (`xgboost`) perform better than support vector machines.

5.2.3. Machine Learning algorithms without feature extraction

Additionally, we aim to provide some insight with regards to the performance of machine learning algorithms that ignore the functional nature of our data. Figure 5 provides an overview over the performance of different machine learning algorithms that are often used together with traditional tabular data. Performances in this figure are obtained from algorithms directly applied to the functional data without any additional feature extraction. The widely used gradient boosting (`xgboost`) and random forest (`ranger`) implementations seem to work reasonably well for functional data even without additional feature extraction.

5.2.4. The effect of tuning hyperparameters

From our experiments, we conclude, that tuning hyperparameters of machine learning algorithms in general has a non-negligible effect on the performance. Using Bayesian optimization in order to tune algorithm hyperparameters on average yielded an absolute increase in accuracy of 5.4% across data sets and learners.

Figure 6 displays the aggregated time over all data sets, taking into account the time required for hyperparameter tuning. All experiments have been run on equivalent hardware on high-performance computing infrastructure. Due to fluctuations in server load, this does not allow for an exact comparison with respect to computation time, but we hope to achieve comparable results as

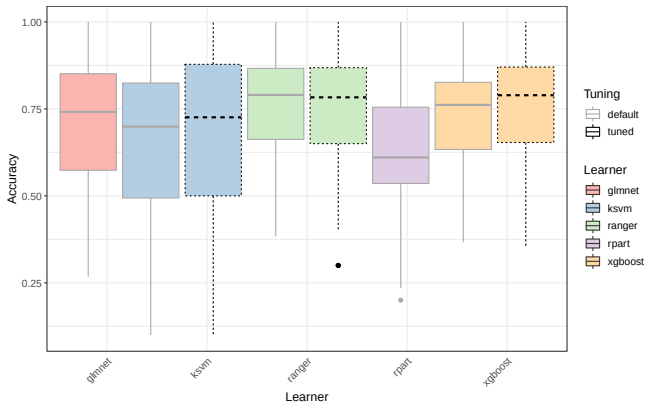


Figure 5: Performance of non-functional machine learning algorithms across 51 data sets applied directly to functional data with and without tuning.

we repeatedly evaluate on sub-samples. Note that we restrict the *tuning* to 3 algorithms where tuning traditionally leads to higher performances.[2]

5.2.5. Top 10 Algorithms and recommendations

Table 2 showcases the top 10 algorithms from the benchmark in terms of average rank in predictive accuracy across data sets. With this list, we aim to provide some initial understanding of the performance of different algorithms and feature extraction methods. Note that this list by no means reflects performance on future data sets, but might serve as an indicator, of which algorithms one might want to try first given computational constraints.

We observe that wavelet extraction in combination with either ranger or xgboost seems to be very strong. They obtain an average rank of 12.90 and 14.45 (out of 80) respectively. Dynamic time warping distances for k-nearest neighbors indeed seems to be a strong baseline, even without tuning. Another strong feature extraction method seems to be the extraction of B-spline features.

[2]Additionally, we find significantly improved performance for tuned FDboost in Figure 3

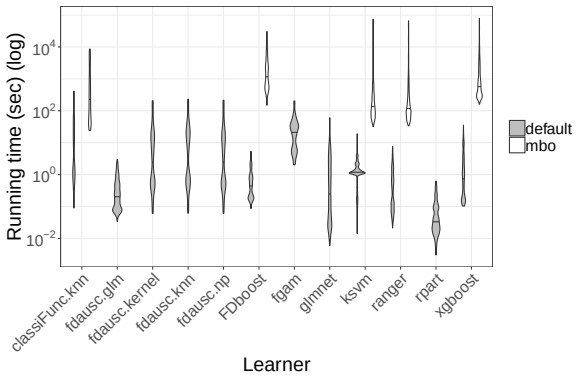


Figure 6: Comparison of running time for the different learner classes with default and tuned hyperparameters across 51 data sets. A log transformation on the running time in seconds is applied, and the mean running time is visualized for each stratification as a horizontal line within the violin plot.

Using the 10 algorithms above allows us to obtain an accuracy within 5% of the maximum on 49 of the 51 data sets.

If the only criterion for model selection is predictive performance, (tuned) machine learning models in combination with feature extraction is a competitive baseline. This class of methods achieves within 95% of the optimal performance on 47 out of 51 data sets, while they include the best performing classifier in 35 cases.

5.2.6. Comparison to classical time series classification

Even though the main purpose of this paper is not a direct comparison with the results from [10], we can use our results to show that applying functional data approaches and classical machine learning approaches together with feature extraction can still improve classification accuracy compared to current state-of-the-art time series classification methods.

In the experiments we conducted, the methods described in this paper improved accuracy on 9 out of the 51 data sets which is displayed in Figure 7. The 9 data

Table 2: Top 10 algorithms by average rank across all data sets. Percent Accuracy describes the fraction of the maximal accuracy reached for each task.

Algorithm Setting	Accuracy %	Average Rank
ranger_wavelet_tuned	0.92	12.90
xgboost_wavelet_tuned	0.92	14.45
ranger_bsignal_tuned	0.90	15.02
knn_dtw_tuned	0.92	15.22
ranger_none_default	0.90	15.59
ranger_bsignal_default	0.89	15.71
ranger_wavelet_default	0.90	16.33
knn_dtw_default	0.92	16.43
xgboost_bsignal_tuned	0.90	17.57
ranger_none_tuned	0.89	18.49

sets and the corresponding best learner are displayed in Table 4. For each data set, only the best reached accuracy for both sets of algorithms is displayed.

Additionally, we evaluate how our learners rank in comparison to the individual bake-off algorithms from [10]. The algorithm which performs best on a data set obtains the rank 1. The mean rank of the individual learners over all 49 data sets (we take the intersection of the data sets from our benchmark and the ones from [10]). The average sorted ranks for the top 50% algorithms are displayed in Figure 8. We observe that the ensemble methods get the top ranks, which is no surprise, as for instance the COTE algorithm [42] internally combines several classifiers from 4 different time series domains.

However, compared to the classical time series algorithms from [10] with the ensemble methods removed, our functional data algorithms obtain an overall good rank in accuracy performance, interleaved with the algorithms from [10]. Note that the benchmarks are not exactly comparable due to minor differences in the benchmark setup, and we instead only include their reported results.

6. Summary and Outlook

In this work, we provide a benchmark along with a software implementation that integrates the functionality of a diverse set of R-packages into a single user interface and API. Both contributions come with a multiplicity of benefits:

id	type	values	def.	trafo
bsignal				
bsignal.knots	int	{3,...,500}	10	-
bsignal.df	int	{1,...,10}	3	-
multires				
res.level	int	{2,...,5}	-	-
shift	num	[0.01,1]	-	-
pca				
rank.	int	{1,...,30}	-	-
wavelets				
filter	chr	d4,d8,d20,la8,la20,bl14,bl20,c6,c24	-	-
boundary	chr	periodic,reflection	-	-
fourier				
trafo.coeff	chr	phase,amplitude	-	-
dtwkernel				
ref.method	chr	random,all	random	-
n.refs	num	[0,1]	-	-
dtwwindow	num	[0,1]	-	-

Table 3: Parameter spaces and default settings for feature extraction methods.

- The user is not required to learn and deal with the vast complexity of the different interfaces the underlying packages expose.
- All of the existing functionality (e.g., preprocessing, resampling, performance measures, tuning, parallelization) of the **mlr** ecosystem can now be used in conjunction with already existing algorithms for functional data.
- We expose functionality that allows us to work with functional data using *traditional* machine learning methods via feature extraction methods.
- Integration of additional preprocessing methods or models is (fairly) trivial and automatically benefits from the full **mlr** ecosystem.

In order to obtain a broader overview of the performance of the integrated methods, we perform a large benchmark study. This allows users to get an initial overview of potential performances of the different algorithms. Specifically,

- We open up new perspectives for time series classification tasks by incorporating methods from functional data analysis, as well as feature trans-

Name	Algorithm_Setting	Accuracy
Beef	xgboost_wavelet_tuned	0.83
ChlorineConcentration	ksvm_none_tuned	0.91
DistalPhalanxOutlineAgeGroup	ranger_none_default	0.83
DistalPhalanxOutlineCorrect	ranger_dtwkernel_default	0.83
DistalPhalanxTW	ranger_bsignal_default	0.76
Earthquakes	FDboost_none_default	0.80
Ham	xgboost_wavelet_tuned	0.84
InsectWingbeatSound	ranger_wavelet_default	0.65
SonyAIBORobotSurface1	ksvm_wavelet_default	0.94

Table 4: Data sets together with corresponding **mlrFDA** learner and accuracy for which our learners were able to improve accuracy in the conducted experiments.

formations combined with conventional machine learning models.

- Based on the large scale benchmark, we conclude that many learners have competitive performance (Figure 7) and additionally offer the interpretability of many functional data analysis methods. Our toolbox serves as a strong complement and alternative to other time series classification software.

- The presented benchmark study uses state-of-the-art Bayesian optimization for hyperparameter optimization, which results in significant improvements over models that are not tuned. This kind of hyperparameter tuning is easy to do with **mlrFDA**. Tuning, albeit heavily influencing performance is often not investigated. Our benchmark closes this gap in existing literature.

- We find that extracting vector valued features and feeding them to a conventional machine learning model can often form competitive learners.

- The pareto-optimal set in terms of performance on each data set contains 23 different algorithm−feature-extraction combinations. Our toolbox *i*) offers the same API for all methods and *ii*) allows to automatically search over this space, and thus allows users to obtain optimal models without knowing all underlying methods.

parameter	type	values	default	trafo
ksvm				
C	num	[-15,15]	-	2^x
sigma	num	[-15,10]	-	2^x
ranger				
mtry.power	num	[0,1]	-	p^x
min.node.size	num	[0,0.99]	-	2^($log2(n) * x$)
sample.fraction	num	[0.1,1]	-	-
xgboost				
nrounds	int	{1,...,5000}	100	-
eta	num	[-10,0]	-	2^x
subsample	num	[0.1,1]	-	-
booster	chr	gbtree,gblinear	-	-
max_depth	int	{1,...,15}	-	-
min_child_weight	num	[0,7]	-	2^x
colsample_bytree	num	[0,1]	-	-
colsample_bylevel	num	[0,1]	-	-
lambda	num	[-10,10]	-	2^x
alpha	num	[-10,10]	-	2^x
FDboost				
mstop	int	{1,...,5000}	100	-
nu	num	[0,1]	0.01	-
df	num	[1,5]	4	-
knots	int	{5,...,100}	10	-
degree	int	{1,...,4}	3	-

Table 5: Parameter spaces and defaults used for tuning machine learning and functional data algorithms. In case no default is provided, package defaults are used. Additional information can be found in the respective packages documentation.

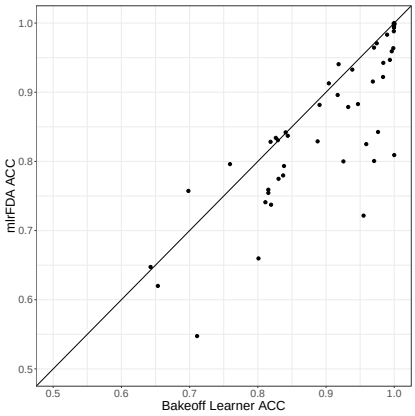


Figure 7: Comparing accuracy between our **mlrFDA** learners and the classical time series classification algorithms in [10]. For each data set, only the best accuracy for each of the two benchmarks is shown. We observe that for 9 of the evaluated data sets the classification performance can directly be improved solely by applying our **mlrFDA** learners, while we perform on par with the classical time series classification algorithms (when rounding to 3 decimal digits) on two data sets.

Concerning the questions we proposed at the beginning of the paper, we draw the following conclusions:

- Tuning only a subset of the presented learners and feature extractions, i.e., the methods listed in Table 2, is sufficient to achieve good performances on almost all data sets in our benchmark.
- A simple random forest without any preprocessing can also be a reasonable baseline for time series data. It achieves an average rank of 15.59 (top 4) in our benchmark.
- Most algorithms for functional data (e.g., `FDboost`) do not perform well in our benchmark study. As those algorithms are fully interpretable and offer statistically valid coefficients, they can still be useful in some applications, and should thus not be ruled out.
- Feature extraction techniques, such as b-spline representations (*bsignal*)

and wavelet extraction work well in conjunction with machine learning techniques for vector valued features such as `xgboost` and `random forest`.

- Tuning leads to an average reduction in absolute MMCE of 3.59% (ranger), 5.69% (xgboost), 7.78% (ksvm) (across feature extraction techniques) and 11% (FDboost). This holds for all feature extraction techniques, where improvements range from 1.12% *multires* to 20.3% *fourier*.

In future work we will continue to expand the available toolbox along with a benchmark of new methods, and provide the R community a wider range of methods that can be used for the analysis of functional data. This includes not only integrating many already available packages, and as a result to enable preprocessing operations such as smoothing (e.g., **fda** [4]) and alignment (e.g., **fdasrvf** [43] or **tidyfun** [44]), but also to explore and integrate advanced imputation methods for functional data. Further work will also extend the current implementation to support data that is measured on unequal or irregular grids. Additionally, we aim to implement some of the current state-of-the art machine learning models from the time series classification bake-off [10], such as the *Collective of Transformation-Based Ensembles* (COTE) [31]. This enables researchers to use and compare with current state-of-the-art methods.

References

[1] S. Ullah, C. F. Finch, Applications of functional data analysis: A systematic review, BMC Medical Research Methodology 13 (2013) 43. URL: https://doi.org/10.1186/1471-2288-13-43. doi:10.1186/1471-2288-13-43.

[2] J.-L. Wang, J.-M. Chiou, H.-G. Müller, Functional data analysis, Annual Review of Statistics and Its Application 3 (2016) 257–295. URL: https://doi.org/10.1146/annurev-statistics-041715-033624. doi:10.1146/annurev-statistics-041715-033624.

[3] R Core Team, R: A Language and Environment for Statistical Computing, R Foundation for Statistical Computing, Vienna, Austria, 2014. URL: http://www.R-project.org/.

[4] J. O. Ramsay, H. Wickham, S. Graves, G. Hooker, fda: Functional Data Analysis, 2018. URL: https://CRAN.R-project.org/package=fda, r package version 2.4.8.

[5] S. Brockhaus, D. Ruegamer, FDboost: Boosting Functional Regression Models, 2018.

[6] F. Scheipl, Cran task view - functional data analysis, 2018. URL: https://cran.r-project.org/web/views/FunctionalData.html.

[7] M. Febrero-Bande, M. Oviedo de la Fuente, Statistical computing in functional data analysis: The r package fda.usc, Journal of Statistical Software 51 (2012) 1–28. URL: http://www.jstatsoft.org/v51/i04/.

[8] J. Goldsmith, F. Scheipl, L. Huang, J. Wrobel, J. Gellar, J. Harezlak, M. W. McLean, B. Swihart, L. Xiao, C. Crainiceanu, P. T. Reiss, refund: Regression with Functional Data, 2018. URL: https://CRAN.R-project.org/package=refund, r package version 0.1-17.

[9] C. Cortes, V. Vapnik, Support-vector networks, Mach. Learn. 20 (1995) 273–297. URL: https://doi.org/10.1023/A:1022627411411. doi:10.1023/A:1022627411411.

[10] A. Bagnall, J. Lines, A. Bostrom, J. Large, E. Keogh, The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances, Data Mining and Knowledge Discovery 31 (2017) 606–660.

[11] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, P.-A. Muller, Deep learning for time series classification: a review, Data Mining and Knowledge Discovery (2019) 1–47.

[12] J. J. Rodriguez, L. I. Kuncheva, C. J. Alonso, Rotation forest: A new classifier ensemble method, IEEE Transactions on Pattern Analysis and Machine Intelligence 28 (2006) 1619–1630. doi:10.1109/TPAMI.2006.211.

[13] C. Stachl, M. Bühner, Show me how you drive and i'll tell you who you are. recognizing gender using automotive driving parameters, Procedia Manufacturing 3 (2015) 5587 – 5594. URL: http://www.sciencedirect.com/science/article/pii/S2351978915007441. doi:https://doi.org/10.1016/j.promfg.2015.07.743, 6th International Conference on Applied Human Factors and Ergonomics (AHFE 2015) and the Affiliated Conferences, AHFE 2015.

[14] R. Hyndman, E. Wang, Y. Kang, T. Talagala, Y. Yang, tsfeatures: Time Series Feature Extraction, 2018. URL: https://github.com/robjhyndman/tsfeatures/, r package version 0.1.

[15] E. Aldrich, wavelets: A package of functions for computing wavelet filters, wavelet transforms and multiresolution analyses, 2013. URL: https://CRAN.R-project.org/package=wavelets, r package version 0.3-0.

[16] J. Goldsmith, F. Scheipl, Estimator selection and combination in scalar-on-function regression, Computational Statistics & Data Analysis 70 (2014) 362–372.

[17] S. Mallat, A theory for multiresolution signal decomposition: The wavelet representation, IEEE Trans. Pattern Anal. Mach. Intell. 11 (1989) 674–693.

[18] E. O. Brigham, R. E. Morrow, The fast fourier transform, IEEE Spectrum 4 (1967) 63–70. doi:10.1109/MSPEC.1967.5217220.

[19] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, E. Keogh, Searching and mining trillions of time series subsequences under dynamic time warping, in: Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2012, pp. 262–270. URL: http://doi.org/10.1145/2339530.2339576. doi:10.1145/2339530.2339576.

[20] P. Boersch-Supan, rucrdtw: Fast time series subsequence search in r, The Journal of Open Source Software 1 (2016) 1–2. URL: http://doi.org/10.21105/joss.00100. doi:10.21105/joss.00100.

[21] J. Ramsay, Functional data analysis, Wiley Online Library, 2006.

[22] A. Srivastava, E. P. Klassen, Functional and Shape Data Analysis, Springer, 2016.

[23] T. Hothorn, P. Bühlmann, T. Kneib, M. Schmid, B. Hofner, Model-based boosting 2.0, Journal of Machine Learning Research 11 (2010) 2109–2113.

[24] S. Greven, F. Scheipl, A general framework for functional regression modelling, Statistical Modelling 17 (2017) 1–35.

[25] F. Ferraty, P. Vieu, Nonparametric functional data analysis: theory and practice, Springer Science & Business Media, 2006.

[26] T. Maierhofer, F. Pfisterer, classiFunc: Classification of Functional Data, 2018. URL: https://CRAN.R-project.org/package=classiFunc, r package version 0.1.1.

[27] Grupo de Aprendizaje Automatico - Universidad Autonoma de Madrid, scikit-fda: Functional Data Analysis in Python, 2019. URL: https://fda.readthedocs.io.

[28] A. Bagnall, J. Lines, W. Wickers, E. Keogh, The UEA & UCR time series classification repository, 2017. www.timeseriesclassification.com.

[29] L. Breiman, Random forests, Mach. Learn. 45 (2001) 5–32. URL: https://doi.org/10.1023/A:1010933404324. doi:10.1023/A:1010933404324.

[30] P. Tormene, T. Giorgino, S. Quaglini, M. Stefanelli, Matching incomplete time series with dynamic time warping: An algorithm and an application to post-stroke rehabilitation, Artificial Intelligence in Medicine 45 (2008) 11–34. doi:10.1016/j.artmed.2008.11.007.

[31] A. Bagnall, J. Lines, J. Hills, A. Bostrom, Time-series classification with cote: The collective of transformation-based ensembles, IEEE Transactions on Knowledge and Data Engineering 27 (2015) 2522–2535. doi:10.1109/TKDE.2015.2416723.

[32] K. Fuchs, F. Scheipl, S. Greven, Penalized scalar-on-functions regression with interaction term, Computational Statistics & Data Analysis 81 (2015) 38âĂŞ51. doi:10.1016/j.csda.2014.07.001.

[33] C. A. Ratanamahatana, E. J. Keogh, Three myths about dynamic time warping data mining., in: H. Kargupta, J. Srivastava, C. Kamath, A. Goodman (Eds.), SDM, SIAM, 2005, pp. 506–510.

[34] J. Hills, J. Lines, E. Baranauskas, J. Mapp, A. Bagnall, Classification of time series by shapelet transformation, Data Min. Knowl. Discov. 28 (2014) 851–881. URL: http://dx.doi.org/10.1007/s10618-013-0322-1. doi:10.1007/s10618-013-0322-1.

[35] P. Kokoszka, M. Reimherr, Introduction to functional data analysis, CRC Press, 2017.

[36] J. Schiffner, B. Bischl, M. Lang, J. Richter, Z. M. Jones, P. Probst, F. Pfisterer, M. Gallo, D. Kirchhoff, T. Kühn, et al., mlr tutorial, arXiv preprint arXiv:1609.06146 (2016).

[37] B. Bischl, O. Mersmann, H. Trautmann, C. Weihs, Resampling methods for meta-model validation with recommendations for evolutionary computation, Evolutionary Computation 20 (2012) 249–275. URL: https://doi.org/10.1162/EVCO.a.00069. doi:10.1162/EVCO.a.00069, pMID: 22339368.

[38] J. Snoek, H. Larochelle, R. P. Adams, Practical bayesian optimization of machine learning algorithms, in: F. Pereira, C. J. C. Burges, L. Bottou, K. Q. Weinberger (Eds.), Advances in Neural Information Processing Systems 25, Curran Associates, Inc., 2012, pp. 2951–2959. URL: http://papers.nips.cc/paper/4522-practical-bayesian-optimization-of-machine-learning-algorithms.pdf.

[39] B. Bischl, J. Richter, J. Bossek, D. Horn, J. Thomas, M. Lang, mlrMBO: A Modular Framework for Model-Based Optimization of Expensive Black-Box Functions, 2017. URL: http://arxiv.org/abs/1703.03373.

[40] B. Bischl, M. Lang, O. Mersmann, J. Rahnenführer, C. Weihs, BatchJobs and BatchExperiments: Abstraction mechanisms for using R in batch environments, Journal of Statistical Software 64 (2015) 1–25. URL: http://www.jstatsoft.org/v64/i11/.

[41] M. Lang, B. Bischl, D. Surmann, batchtools: Tools for r to work on batch systems, The Journal of Open Source Software 2 (2017). URL: https://doi.org/10.21105/joss.00135. doi:10.21105/joss.00135.

[42] A. Bagnall, J. Lines, J. Hills, A. Bostrom, Time-series classification with cote: the collective of transformation-based ensembles, IEEE Transactions on Knowledge and Data Engineering 27 (2015) 2522–2535.

[43] J. D. Tucker, fdasrvf: Elastic Functional Data Analysis, 2016. URL: https://CRAN.R-project.org/package=fdasrvf, r package version 1.6.0.

[44] F. Scheipl, J. Goldsmith, tidyfun, https://github.com/fabian-s/tidyfun, 2019.

[45] L. Jin, Q. Niu, Y. Jiang, H. Xian, Y. Qin, M. Xu, Driver sleepiness detection system based on eye movements variables, Advances in Mechanical Engineering 5 (2013) 648431. URL: https://doi.org/10.1155/2013/648431. doi:10.1155/2013/648431.

[46] M. Murugappan, M. Rizon, R. Nagarajan, S. Yaacob, Eeg feature extraction for classifying emotions using fcm and fkm, in: Proceedings of the 7th WSEAS International Conference on Applied Computer and Applied Computational Science, ACACOS'08, World Scientific and Engineering Academy and Society (WSEAS), Stevens Point, Wisconsin, USA, 2008, pp. 299–304. URL: http://dl.acm.org/citation.cfm?id=1415743.1415793.

[47] S. Soltani, On the use of the wavelet decomposition for time series prediction, Neurocomputing 48 (2002) 267 – 277. URL: http://www.sciencedirect.com/science/article/pii/S0925231201006488. doi:https://doi.org/10.1016/S0925-2312(01)00648-8.

Appendix A. API overview

For the interested reader, we introduce a brief overview of the API and functionality.

Appendix A.1. Representing functional data in mlrFDA

A sketch of the data structure we use to represent functional data can be found in the right part of Figure A.9. We assume a data set consists of data for N observational units, organized in rows of features, where one row contains all observed features for one observational unit, i.e., each row typically contains several functional and/or scalar covariates. For a classical, non-functional data set, the P features are single columns (as depicted in the left part of Figure A.9). A functional data set, on the other hand, consists of single-column scalar features as well as functional features of different length for each functional covariate, each represented by multiple adjacent and connected columns. Each of these columns contains the evaluations of the functional feature at a certain argument value for all observational units (right part of Figure A.9).

As an example which will be used throughout the remainder of this paper, we use the `fuelSubset` data set from package **FDboost**, see also Figure 1. It contains a numeric target variable `heatan`, the fuel's heating value, a scalar feature `h2O`, the fuel's water content, and two functional features `NIR` and `UVVIS`, measured at 231 and 129 wavelengths, respectively. To start with a clean sheet, we create a `data.frame` containing all features as separate columns.

```
R> library(mlr)
R> library(FDboost)
R> df = data.frame(fuelSubset[c("heatan", "h2o", "UVVIS", "NIR")])
```

The first step when setting up an experiment in any analysis is to make the data accessible for the specific algorithms that will be applied. In **mlr**, the data itself, and additional information, such as which column corresponds to the target variable is stored as a `Task`, requiring the input data to be of type

`data.frame`.

The list of column positions of the functional features is then passed as argument `fd.features` to `makeFunctionalData()`, which returns an object of type `data.frame` in which the columns corresponding to each functional feature are combined into `matrix` columns. [3]

```
R> fd.features = list("UVVIS" = 3:136, "NIR" = 137:367)
R> fdf = makeFunctionalData(df, fd.features = fd.features)
R> str(fdf)
'data.frame':        129 obs. of  4 variables:
 \$ heatan: num  26.8 27.5 23.8 18.2 17.5 ...
 \$ h2o   : num  2.3 3 2 1.85 2.39 ...
 \$ UVVIS : num [1:129, 1:134] 0.145 -1.584 -0.814 -1.311 -1.373 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..\$ : NULL
  .. ..\$ : chr  "UVVIS.1" "UVVIS.2" "UVVIS.3" "UVVIS.4" ...
 \$ NIR   : num [1:129, 1:231] 0.2818 0.2916 -0.0042 -0.034 -0.1804 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..\$ : NULL
  .. ..\$ : chr  "NIR.1" "NIR.2" "NIR.3" "NIR.4" ...
```

We additionally specify the name `"fuelsubset"` and the target variable `"heatan"`. The structure of the functional `Task` object is rather similar to the non-functional **Task**, with the additional information `functionals`, which states how many functional features are present in the underlying data.

```
R> tsk1 = makeRegrTask("fuelsubset", data = fdf, target = "heatan")
R> print(tsk1)
```

[3] As an alternative, a *list* of the column names containing the functional features is also valid as argument to *fd.features*, which is especially useful if columns are already labeled.

```
Supervised task: fuelsubset
Type: regr
Target: heatan
Observations: 129
Features:
   numerics     factors     ordered functionals
          1           0           0           2
Missings: FALSE
Has weights: FALSE
Has blocking: FALSE
Has coordinates: FALSE
```

After defining the task, a `learner` is created by calling `makeLearner`. This contains the algorithm that will be fitted on the data in order to obtain a model. Currently, **mlrFDA** supports both functional regression and functional classification. A list of supported learners can be found in Table 1

Appendix A.2. Machine Learning and Feature Extraction

Classical machine learning algorithms do not take into account the characteristics of functional data and treat the input data as vector valued features. Without additional preprocessing, this typically yields poor performance on, as the models cannot exploit the lower intrinsic dimensionality of the functional covariates nor the fact that they represent observations over a continuum.

In **mlrFDA**, classical algorithms can be applied to functional data, however, a warning message will be displayed. In our example, we train a partitioning tree on the functional data, while ignoring the functional structure.

```
R> rpart.lrn = makeLearner("regr.rpart")
R> m = train(learner = rpart.lrn, task = tsk1)
   Functional features have been converted to numerics
```

For conventional learning algorithms to work well on functional data, informative scalar features need to be extracted from the functional features.

Name	Function	Package
Discrete Wavelet Transform	`extractFDAWavelets()`	**wavelets**
Fast Fourier Transform	`extractFDAFourier()`	**stats**
Principal Component Analysis	`extractFDAPCA()`	**stats**
B-Spline Features	`extractFDABsignal()`	**FDboost**
Multi-Resolution Feature Extraction	`extractFDAMultiResFeatures()`	-
Time Series Features	`extractFDATsfeatures()`	**tsfeatures**
Dynamic Time-Warping Kernel	`extractFDADTWKernel()`	**rucrdtw**

Table A.6: Feature extraction methods currently implemented in **mlrFDA** and underlying packages

Feature extraction is applied in practice for a multiplicity of reasons, as it often not only reduces the dimensionality of the resulting problem, but also allows researchers to make use of domain knowledge, for example by handcrafting features from measurements of continuous processes. Examples for this include deriving features that allow for sleepiness detection [45], or by extracting features from electro-cardiogram data in order to detect emotions [46]. The resulting features often have a much lower dimensionality, which often improves fitted models. Other preprocessing methods for functional or time series data include extracting general purpose features such as wavelet coefficients [15, 47], principal component scores or Fourier coefficients. The resulting scalar features can then be used with different machine learning methods such as k-nearest neighbors.

In the following section, we showcase the feature extraction procedure using general purpose features as an example. We want to emphasize that it is also easily possible to write custom feature extraction methods using the `makeFeatureExtractionMethod` function.

In our example, we extract the Fourier coefficients from the functional feature `UVVIS`, and principal component scores from the second functional feature `NIR` in order to transform the original task with functional data into a conventional task.

```
R> feat.methods = list("UVVIS" = extractFDAFourier(),
"NIR" = extractFDAPCA())
R> extracted = extractFDAFeatures(tsk, feat.methods = feat.methods)
```

```
R> extracted

  $task
  Supervised task: fuelsubset
  Type: regr
  Target: heatan
  Observations: 129
  Features:
     numerics     factors     ordered functionals
           137           0           0           0
  Missings: FALSE
  Has weights: FALSE
  Has blocking: FALSE
  Has coordinates: FALSE

  $desc
  Extraction of features from functional data:
  Target: heatan
  Functional Features: 2; Extracted features: 2
```

As an alternative, the feature extraction can be applied in a wrapper method `makeExtractFDAFeatsWrapper()`. In general, a wrapper combines a learner method with another method, thereby creating a new learner that can be handled like any other learner. In our case, a classical machine learning method is combined with the data preprocessing step of feature transformation from functional to non-functional data.

```
R> wrapped.lrn = makeExtractFDAFeatsWrapper("regr.rpart",
 feat.methods = feat.methods)
```

This is suitable for honest cross-validation of data-adaptive feature extraction methods like principal components. We can now cross-validate the learner created above using **mlr**'s `resample` function with 10-fold cross-validation.

```
R> res = resample(learner = wrapped.lrn, task = tsk1,
resampling = cv10)
```

In the same way, we can `train` and `predict` on data, or `benchmark` multiple learners across multiple data sets. Additionally, we can apply a `tuneWrapper` to our learner in order to automatically tune hyperparameters of the learner and the preprocessing method during cross-validation.

Appendix B. Data sets used in the Benchmark

Table B.7 contains all data sets used in the benchmark along with additional data properties.

Appendix C. Failed and missing experiments

Experiments for some algorithm / data set combinations failed due to implementation details or algorithm properties. In order to increase transparency, failed algorithms are listed here, and if available reasons for failure are provided.

At the time of the benchmark, the implementation in the **tsfeatures** package was not stable enough to be included in the benchmark.

- `classif.fgam`
  Data sets: BeetleFly, BirdChicken, Coffee, Computers, DistalPhalanxOutlineCorrect, Earthquakes, ECG200, ECGFiveDays, ElectricDeviceOn, GunPoint, Ham, Herring, ItalyPowerDemand, Lightning2, MoteStrain, ShapeletSim, SonyAIBORobotSurface1, Strawberry, ToeSegmentation1, TwoLeadECG, Wafer, Wine, Yoga
  Reason: Too few instances in some classes, such that p > n.
- `classif.fdausc.kernel and .np` **Data sets:** ElectricDeviceOn, ShapeletSim
- `classif.fdausc.knn`
  Data sets: DistalPhalanxTW, EpilepsyX

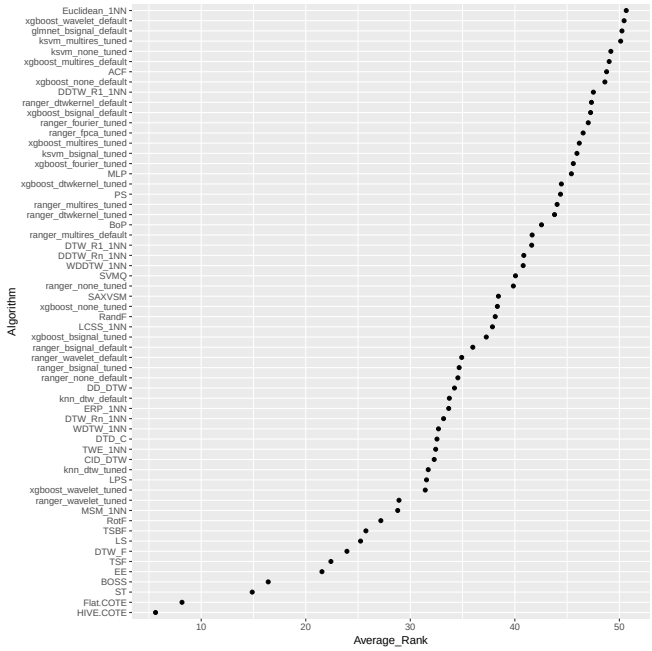


Figure 8: Comparing sorted average performance ranks between our **mlrFDA** learners (algorithm names in lower case) and the classical time series classification algorithms (algorithm names in capital) in [10], The mean rank of each individual learner over all 49 data sets is displayed. Only the first half of all algorithms being compared are displayed here. We observe that the Ensemble Methods like HIVE.COTE, FLAT.COTE, ST, BOSS, EE occupy the top tier, while the rest of the rank space are interleaved by our **mlrFDA** algorithms and algorithms from [10]

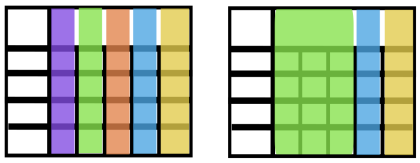


Figure A.9: Schematic comparison of non-functional and functional data representation in mlrFDA. The green feature is a functional feature spanning multiple columns.

Name	Obs.	Classes	Length	Type	Split
Adiac	781	37	176	IMAGE	0.50
ArrowHead	211	3	251	IMAGE	0.17
Beef	60	5	470	SPECTRO	0.50
BeetleFly	40	2	512	IMAGE	0.50
BirdChicken	40	2	512	IMAGE	0.50
Car	120	4	577	SENSOR	0.50
CBF	930	3	128	SIMULATED	0.03
ChlorineConcentration	4307	3	166	SIMULATED	0.11
Coffee	56	2	286	SPECTRO	0.50
Computers	500	2	720	DEVICE	0.50
CricketX	780	12	300	MOTION	0.50
DistalPhalanxOutlineAgeGroup	539	3	80	IMAGE	0.74
DistalPhalanxOutlineCorrect	876	2	80	IMAGE	0.68
DistalPhalanxTW	539	6	80	IMAGE	0.74
Earthquakes	461	2	512	SENSOR	0.70
ECG200	200	2	96	ECG	0.50
ECGFiveDays	884	2	136	ECG	0.03
ElectricDeviceOn	1008	2	360	DEVICE	0.63
EpilepsyX	275	4	208	HAR	0.61
FaceAll	2250	14	131	IMAGE	0.25
FacesUCR	2250	14	131	IMAGE	0.09
Fish	350	7	463	IMAGE	0.50
GunPoint	200	2	150	MOTION	0.25
Ham	214	2	431	SPECTRO	0.51
Herring	128	2	512	IMAGE	0.50
InsectWingbeatSound	2200	11	256	SENSOR	0.10
ItalyPowerDemand	1096	2	24	SENSOR	0.06
LargeKitchenAppliances	750	3	720	DEVICE	0.50
Lightning2	121	2	637	SENSOR	0.50
Lightning7	143	7	319	SENSOR	0.49
Meat	120	3	448	SPECTRO	0.50
MedicalImages	1141	10	99	IMAGE	0.33
MoteStrain	1272	2	84	SENSOR	0.02
OSULeaf	442	6	427	IMAGE	0.45
Plane	210	7	144	SENSOR	0.50
RefrigerationDevices	750	3	720	DEVICE	0.50
ScreenType	750	3	720	DEVICE	0.50
ShapeletSim	200	2	500	SIMULATED	0.10
SmallKitchenAppliances	750	3	720	DEVICE	0.50
SonyAIBORobotSurface1	621	2	70	SENSOR	0.03
Strawberry	983	2	235	SPECTRO	0.62
SwedishLeaf	1125	15	128	IMAGE	0.44
SyntheticControl	600	6	60	SIMULATED	0.50
ToeSegmentation1	268	2	277	MOTION	0.15
Trace	200	4	275	SENSOR	0.50
TwoLeadECG	1162	2	82	ECG	0.02
TwoPatterns	5000	4	128	SIMULATED	0.20
UWaveGestureLibraryX	4478	8	315	MOTION	0.20
Wafer	7164	2	152	SENSOR	0.14
Wine	111	2	234	SPECTRO	0.51
Yoga	3300	2	426	IMAGE	0.09

Table B.7: Data sets from the UCI Archive used in the benchmark.

Appendix H

Benchmark for filter methods for feature selection in high-dimensional classification data

Contributing Article Andrea Bommert, Xudong Sun, Bernd Bischl, Joerg Rahnenfuehrer, Michel Lang, Benchmark for filter methods for feature selection in high-dimensional classification data, Computational Statistics & Data Analysis, Volume 143, 2020, 106839, ISSN 0167-9473, https://doi.org/10.1016/j.csda.2019.106839.

Author Contributions In Bommert et al. (2020), Xudong Sun implemented the mutual information based filter methods from package `praznik`, and the entropy based filter methods from package `FSelectorRcpp` into the `mlr` package with refinements from Michel Lang. All authors contribute to dataset collection, design of benchmark experiment and discussion on the experimental results. Andrea Bommert implemented the benchmark code used in the paper, conducted the experiment, analyzed the experimental results and wrote the manuscript. Xudong Sun and the other authors proofread the manuscripts.

Computational Statistics and Data Analysis 143 (2020) 106839

Contents lists available at ScienceDirect

Computational Statistics and Data Analysis

journal homepage: www.elsevier.com/locate/csda

Benchmark for filter methods for feature selection in high-dimensional classification data ☆

Andrea Bommert [a,*], Xudong Sun [b], Bernd Bischl [b], Jörg Rahnenführer [a], Michel Lang [a]

[a] Department of Statistics, TU Dortmund University, 44221 Dortmund, Germany
[b] Department of Statistics, Ludwig-Maximilians-Universität München, Ludwigstr. 33, 80539 München, Germany

ARTICLE INFO

Article history:
Received 27 July 2018
Received in revised form 24 June 2019
Accepted 15 July 2019
Available online 19 September 2019

Keywords:
Feature selection
Filter methods
High-dimensional data
Benchmark

ABSTRACT

Feature selection is one of the most fundamental problems in machine learning and has drawn increasing attention due to high-dimensional data sets emerging from different fields like bioinformatics. For feature selection, filter methods play an important role, since they can be combined with any machine learning model and can heavily reduce run time of machine learning algorithms. The aim of the analyses is to review how different filter methods work, to compare their performance with respect to both run time and predictive accuracy, and to provide guidance for applications. Based on 16 high-dimensional classification data sets, 22 filter methods are analyzed with respect to run time and accuracy when combined with a classification method. It is concluded that there is no group of filter methods that always outperforms all other methods, but recommendations on filter methods that perform well on many of the data sets are made. Also, groups of filters that are similar with respect to the order in which they rank the features are found. For the analyses, the R machine learning package *mlr* is used. It provides a uniform programming API and therefore is a convenient tool to conduct feature selection using filter methods.

1. Introduction

Feature selection has become increasingly important for data analysis, machine learning, and data mining. Especially for high-dimensional data sets, it is necessary to filter out the irrelevant and redundant features by choosing a suitable subset of relevant features in order to avoid over-fitting and tackle the curse of dimensionality. With respect to data sets from the bioinformatics domain, feature selection often allows identifying the features that are important for biological processes of interest.

When fitting a statistical model for such high-dimensional data sets, one needs to decide first which feature selection method to use. As many of them exist, this is not an easy decision. Regarding the comparison of different methods, benchmark studies have gained increasing attention in the machine learning community (Fernández-Delgado et al., 2014).

In this paper, we benchmark state-of-the-art feature selection techniques on high-dimensional data sets. We compare 22 filter methods from different toolboxes on 16 high-dimensional classification data sets from various domains. We investigate which methods select the features of a data set in a similar order. Additionally, we search for optimal methods

☆ The R source code for the analyses of this article is publicly available at https://github.com/bommert/filter-benchmark-paper.

* Corresponding author.
E-mail address: bommert@statistik.tu-dortmund.de (A. Bommert).

https://doi.org/10.1016/j.csda.2019.106839

with respect to predictive accuracy and run time. The results of our analyses show three groups of similar filter methods as well as several filter methods that are not very similar to any other filter method. Some of the filter methods seem to work better than others, but which filter methods perform best depends on the data set. There is no one-fits-all solution, but some recommendations can be made on the choice of filter methods.

In the past decades, many feature selection methods have been proposed. The methods can be categorized into three classes (Guyon and Elisseeff, 2003): *Filter methods* rank features by calculating a score for each feature independent of a model. Either the m features with the highest scores or all the features whose scores exceed a threshold τ are selected (with $m \in \mathbb{N}$ or $\tau \in \mathbb{R}$ being pre-specified). For many filter methods, the score calculation can be done in parallel. Lazar et al. (2012) give an extensive overview of existing filter methods. *Wrapper methods* (Kohavi and John, 1997) consider subsets of the set of all features. For each of the subsets, a supervised learning (e.g. classification) model is fitted. The subsets are evaluated by a performance measure calculated on the resulting model (e.g. classification accuracy). Wrapper methods include simple approaches like greedy sequential searches (Kittler, 1978), but also more elaborate algorithms like recursive feature elimination (Huang et al., 2018) as well as evolutionary and swarm intelligence algorithms for feature selection (Yang and Honavar, 1998; Xue et al., 2016; Brezočnik et al., 2018). *Embedded methods* include the feature selection in the model fitting process. Examples for predictive methods that perform embedded feature selection are Lasso regression (Tibshirani, 1996), tree based methods like classification and regression trees (Breiman et al., 1984) or random forests (Breiman, 2001), and gradient boosting (Biau et al., 2019). There are many overview papers that describe in detail, categorize, and suggest how to evaluate existing feature selection methods, e.g. Guyon and Elisseeff (2003), Liu and Yu (2005), Saeys et al. (2007), Tang et al. (2014), Chandrashekar and Sahin (2014), Hira and Gillies (2015), Jović et al. (2015), Li et al. (2018), Cai et al. (2018) and Venkatesh and Anuradha (2019).

There are also several papers in which feature selection methods are compared. In many of these, the feature selection methods are combined with classification methods in order to assess the predictive performance of the selected features. Liu et al. (2002) compare filter methods based on two gene expression data sets, counting the number of misclassified samples. Bolón-Canedo et al. (2013) analyze the classification accuracy of different filter, wrapper, and embedded methods on several artificial data sets. Bolón-Canedo et al. (2014) and Inza et al. (2004) compare filter methods with respect to classification accuracy based on microarray data sets. Forman (2003) and Aphinyanaphongs et al. (2014) conduct extensive comparisons based on text classification data sets. They analyze filter and wrapper methods, respectively. Darshan and Jaidhar (2018) compare filter methods with respect to classification accuracy on malware detection data. Liu (2004) and Peng et al. (2005) study filter methods on large data sets, analyzing the predictive accuracy with respect to the number of features to be chosen. Dash and Liu (1997) and Sánchez-Maroño et al. (2007) use small artificial data sets to assess whether the correct features are chosen. Dash and Liu (1997) compare different feature selection methods while Sánchez-Maroño et al. (2007) consider filter methods only. Wah et al. (2018) compare filter and wrapper methods on large simulated data sets with respect to the correctness of the chosen features. Additionally, they conduct comparisons with respect to classification accuracy on real data sets. Xue et al. (2015) comprehensively compare filter and wrapper methods with respect to classification accuracy and run time, considering each of the two objectives separately. Most of the data sets on which the comparison is based contain a small or medium number of features.

In some papers, only filter methods whose scores are based on similar concepts are compared. Both Meyer et al. (2008) and Brown et al. (2012) compare several filter methods that are based on mutual information. Meyer et al. (2008) analyze the accuracy and the run time of the methods separately. Additionally, they take into account theoretical properties and look at the percentages of correctly identified features on artificial data. Brown et al. (2012) assess the similarity of the filter methods with respect to feature subsets of size 10. Moreover, they analyze the classification accuracy with respect to the number of chosen features and search for Pareto optimal methods considering the accuracy and feature selection stability. Hall (1999) conducts an extensive study of correlation based feature selection. The author analyzes the classification accuracy based on real data sets as well as the choosing of relevant or irrelevant features based on artificial data sets.

Several authors introduce new feature selection methods and compare them in a benchmark study to competing approaches. Zhu et al. (2007) and Mohtashami and Eftekhari (2019) present new wrapper methods. The comparisons they conduct for the new methods also include filter methods. The authors assess the classification accuracy of the methods on several data sets. Zhu et al. (2007) also consider the number of features for which the best performance is achieved. Yu and Liu (2004), Fleuret (2004), and Ke et al. (2018) present new filter methods and conduct a comparison of their new methods with established feature selection methods. All of them consider the classification accuracy of the feature selection methods and the run time separately. Hoque et al. (2018) develop an ensemble filter method that aggregates the scores of several filter methods and compare it to the single filter methods. They compare the methods with respect to classification accuracy considering both small and high-dimensional data sets. Ghosh et al. (2019) create an ensemble filter method for guiding an evolutionary algorithm. This algorithm is compared to evolutionary algorithms guided by single filter methods with respect to classification accuracy. The comparison is based on high-dimensional data sets and small numbers of features to be chosen.

None of the above studies conduct an extensive comparison of feature selection methods on high-dimensional data with respect to both accuracy and run time jointly. However, for the analysis of high-dimensional data sets it is crucial to know which feature selection methods are suitable. For high-dimensional data, it is necessary to perform feature selection. Also, it is more difficult to perform feature selection than for low-dimensional data. The decision on which features should

be selected is more complicated due to the curse of dimensionality. Moreover, the feature selection method must be fast to compute because of the large number of features.

In this work, we compare 22 filter methods on high-dimensional classification data sets. The filter methods compared by us are representatives of the most prominent general concepts for filter methods. These classes of filter methods are univariate statistical tests, univariate predictive performance indicators, feature variance, random forest importance and information theoretic measures. We use the R package *mlr* as a basis for our experiments. *mlr* is a comprehensive package for machine learning and a standard in the R community. It provides a unified platform that can also be used to concatenate filter methods with predictive methods. All of the compared filter methods have been integrated into *mlr* and are ready to use. We focus on the comparison of filter methods based on the following considerations. Most wrapper methods, evolutionary feature selection algorithms and recursive feature elimination methods are computationally infeasible for high-dimensional data sets. Embedded methods require that a certain predictive model is used. Most filter methods, however, are fast to calculate and can be combined with any kind of predictive method, even methods with embedded feature selection, see Bommert et al. (2017). We extensively compare different toolboxes with filter methods available in R (R Core Team, 2017). To the best of our knowledge, our approaches of considering the accuracy and the run time jointly as well as considering the similarity of the ranking of all features have not been analyzed yet by researchers in a filter comparison study.

The remainder of this paper is organized as follows: In Section 2, we explain a variety of filter methods. In Section 3, we describe the setup of the experiments we conduct to compare the filter methods and analyze their results. Section 3.3 investigates the similarity of the filter methods based on the feature ranking for several data sets and on their scaling behavior. In Section 3.4, we search for optimal filter methods with respect to predictive accuracy and run time. Section 4 contains a summary and the conclusions of our work.

2. Filter methods

All filter methods described in this section are applicable for classification data sets with numeric features. Some of the methods also work for categorical features. We present two kinds of filter methods: Most filter methods calculate a score for all features and then select the features with the highest scores. Some filter methods, however, select features iteratively in a greedy forward fashion. For these filters, in each iteration the feature with the maximal score is selected but the scores of different iterations are not comparable. Notation-wise, we consider a data set with n instances of the p features $X_1, \ldots, X_p$ and class variable Y. The filter methods in Sections 2.1–2.3 are univariate, i.e. they do not consider interactions between the p features. Most of the filter methods in Sections 2.4 and 2.5 are multivariate.

2.1. Univariate statistical tests

Filter *anova.test* performs for each feature an analysis of variance where the class variable is explained by the feature. The value of the F statistic is used as the score. The higher the F statistic, the more different are the mean values of the corresponding feature between the classes. For each feature X_k, the filter score is defined as

$$J_{\text{anova.test}}(X_k) = \frac{\sum_{i=1}^{l} n_i \left(\bar{x}_{i\bullet}^{(k)} - \bar{x}_{\bullet\bullet}^{(k)}\right)^2 / (l-1)}{\sum_{i=1}^{l} \sum_{j=1}^{n_i} \left(x_{ij}^{(k)} - \bar{x}_{i\bullet}^{(k)}\right)^2 / (n-l)}, \tag{1}$$

where l denotes the number of classes of Y and $x_{ij}^{(k)}$, $i \in \{1, \ldots, l\}$, $j \in \{1, \ldots, n_i\}$, denote the observed values of feature X_k for instances of class i. $\bar{x}_{i\bullet}^{(k)} = \frac{1}{n_i} \sum_{j=1}^{n_i} x_{ij}^{(k)}$ is the mean value of X_k in class i and $\bar{x}_{\bullet\bullet}^{(k)} = \frac{1}{n} \sum_{i=1}^{l} \sum_{j=1}^{n_i} x_{ij}^{(k)}$ is the mean value of X_k of all instances in the data set (Rasch et al., 2011, pp. 241 ff.).

Both filters *limma* and *sam* perform a moderated version of the F test. The basic idea of both approaches is to stabilize the estimation of the variance by using information about the variation of all features. For *limma*, this is done with a linear regression model (Smyth, 2004). For *sam*, the observed statistics are compared to the expected values of the statistics based on permutations (Tusher et al., 2001). Both methods are popular for gene expression data analysis. For more details about the two methods, the interested reader is referred to Smyth (2004) and Tusher et al. (2001).

Filter *kruskal.test* applies for each feature a Kruskal–Wallis rank sum test which is the non-parametric equivalent of the analysis of variance. Analogously, the test statistic is used as the filter score and higher values of this test statistic mean that the values of the corresponding feature differ more between the classes. The filter score for feature X_k is

$$J_{\text{kruskal.test}}(X_k) = \frac{12}{n(n+1)} \sum_{i=1}^{l} \frac{1}{n_i} \left(R_i^{(k)} - \frac{n_i(n+1)}{2}\right)^2, \tag{2}$$

where n_i denotes the number of instances in class i, $i \in \{1, \ldots, l\}$, and $R_i^{(k)} = \sum_{j=1}^{n_i} \text{rank}\left(x_{ij}^{(k)}\right)$ is the sum of the ranks of the observations of X_k belonging to class i among all observed values for X_k (Kruskal and Wallis, 1952).

Filter *chi.squared* performs for each feature a χ^2 test of independence between a dichotomized transformation of this feature and the class variable. The value of the χ^2 statistic is used as the score

$$J_{\text{chi.squared}}(X_k) = \sum_{i=1}^{l} \sum_{j=1}^{s} \frac{\left(o_{ij}^{(k)} - e_{ij}^{(k)}\right)^2}{e_{ij}^{(k)}}. \tag{3}$$

$o_{ij}^{(k)}$ denotes the observed number of instances with class i, $i \in \{1, \ldots, l\}$, and a value of X_k of the jth category, $j \in \{1, \ldots, s\}$. $e_{ij}^{(k)} = \frac{1}{n} \cdot \sum_{i=1}^{l} o_{ij}^{(k)} \cdot \sum_{j=1}^{s} o_{ij}^{(k)}$ is the expected number of instances with class i and a value of X_k of the jth category under the assumption of independence (Rasch et al., 2011, pp. 221). The higher the value of the χ^2 statistic, the higher the dependency between the corresponding feature and the class variable. To calculate this test, continuous features have to be discretized. For the implementation in the toolbox *FSelector* (Romanski and Kotthoff, 2016), this is done with the MDL method, see Section 2.6.

2.2. Univariate predictive performance

The score of the *auc* filter represents the classification accuracy when each feature is used directly and separately for class prediction. For each feature X_k, the following prediction rule for the class variable Y is used: $\widehat{Y} = \mathbb{I}_{[c,\infty)}(X_k)$, with $\mathbb{I}$ denoting the indicator function. The receiver operating curve displays the sensitivity and specificity of a classification rule for all choices of a threshold c, see Sammut and Webb (2011). The area under the receiver operating curve (AUC) of the classification rule $\widehat{Y} = \mathbb{I}_{[c,\infty)}(X_k)$ is used to measure how well X_k separates the target variable. An AUC value of 1 means that there is a threshold c for which the prediction rule is perfectly accurate. The value 0 indicates that there is a threshold c for which the rule predicts all labels wrongly which implies that X_k can achieve perfect classification with the rule $\widehat{Y} = \mathbb{I}_{(-\infty,c)}(X_k)$. A value of 0.5 is the worst possible in this application. The value 0.5 is attained, e.g. when the feature and the class variable are independent. Therefore,

$$J_{\text{auc}}(X_k) = |0.5 - \text{AUC}|, \tag{4}$$

where AUC is calculated for the classification rule $\widehat{Y} = \mathbb{I}_{[c,\infty)}(X_k)$, is used as the AUC filter score. This filter is only applicable for two-class data sets.

The idea of the simple association rule filter *oneR* (Romanski and Kotthoff, 2016) is to predict the class based on the value of a single feature. For this, continuous features have to be discretized in advance. For the implementation in the toolbox *FSelector* (Romanski and Kotthoff, 2016), this is done using the MDL method, see Section 2.6. The score of a feature X_k is calculated in the following way: Let $V^{(k)}$ denote the set of possible values for feature X_k. Also, for each value $v \in V^{(k)}$, let $n_{vi}^{(k)}$ denote the number of instances with $X_k = v$ and class i, $i \in \{1, \ldots, l\}$. A simple classification rule for instances with $X_k = v$ is predicting the class i with the highest count $n_{vi}^{(k)}$. The proportion of correctly classified instances by this rule without conducting any resampling is used as filter score:

$$J_{\text{oneR}}(X_k) = \frac{1}{n} \sum_{v \in V^{(k)}} \max_{i \in \{1,\ldots,l\}} n_{vi}^{(k)}. \tag{5}$$

The higher the accuracy of the rule, the more the feature X_k is considered as suitable for univariate class prediction.

Filter *univariate.model.score* (Bischl et al., 2016) fits a model for each feature in which only this feature is used to predict the class variable. Based on these models, the predictive performance of all features is assessed and the resulting filter score is

$$J_{\text{univariate.model.score}}(X_k) = \text{accuracy of univariate predictive model that only uses } X_k. \tag{6}$$

The implementation in *mlr* (Bischl et al., 2016) uses per default a classification tree with default hyper parameters (Therneau et al., 2017), measures the predictive performance by the accuracy (see Section 2.9) and performs a train–test split with ratio 2:1 in order to avoid overestimation of the accuracy. These specifications can be changed.

2.3. Variance

The *variance* filter uses the variance of a feature as its score

$$J_{\text{variance}}(X_k) = \frac{1}{n-1} \left(x_i^{(k)} - \frac{1}{n} \sum_{i=1}^{n} x_i^{(k)} \right)^2. \tag{7}$$

$x_i^{(k)}$, $i \in 1, \ldots, n$, denote the observed values of feature X_k. The idea of this filter is to get rid of features that only consist of noise and therefore have very little variation. This filter only makes sense for data where the features are measured on the same scale and have not been scaled to unit variance.

2.4. Random forest importance

Random forests are bagging ensembles with trees as base learners (Izenman, 2013, pp. 536 ff.). There are two popular ways of measuring feature importance based on a random forest: permutation importance and impurity importance. To calculate the permutation importance, the out of bag (oob) instances for each tree, i.e. the instances that were not used for fitting this tree, are considered. For the oob instances of each tree, feature X_k is permuted. Then the permuted instances are classified by the corresponding trees. The resulting classification accuracy (see Section 2.9) is compared to the classification accuracy without permuting feature X_k. The score of a permutation importance filter is the decrease in classification accuracy from original oob instances to permuted instances (Izenman, 2013, pp. 542 ff.). Features that are important for class prediction cause a large decrease in accuracy as their relevant information is not available when the feature is permuted. Filter *cforest.importance* is a permutation importance filter of a random forest with conditional inference trees as base learners. Filter *permutation* is a permutation importance filter for a forest of classification trees.

$$J_{\text{permutation}}(X_k) = J_{\text{cforest.importance}}(X_k) = \text{accuracy for original oob instances} - \text{accuracy for oob instances with permuted values of } X_k. \tag{8}$$

Filter *impurity* considers the node impurities of the trees. A node containing only instances of one class is called pure, a node with many instances of different classes is considered as impure. For each node in each tree of the forest, the impurity – before and after the split is made – is measured. This can be done for example with the Gini index. The filter score of feature X_k is the mean decrease in impurity due to the splits based on X_k (Izenman, 2013, pp. 542 ff.)

$$J_{\text{impurity}}(X_k) = \frac{\sum_{i \in N^{(k)}} (\text{impurity before node } i - \text{impurity after node } i)}{\left|N^{(k)}\right|}, \tag{9}$$

with $N^{(k)}$ denoting the set of nodes in the random forest in which a split based on X_k is made. A feature that is important for class prediction causes on average large decreases in impurity.

2.5. Mutual information

Let X and Y be two discrete variables with respective (empirical) probability mass function p. Then the entropy of Y is defined as

$$H(Y) = -\sum_y p(y) \log_2 (p(y)) \tag{10}$$

and the conditional entropy of Y given X is given by

$$H(Y|X) = \sum_x p(x) H(Y|X = x) = \sum_x p(x) \left(-\sum_y p(y|x) \log_2 (p(y|x)) \right). \tag{11}$$

The entropy measures the uncertainty of the variable. When all possible values occur with about the same probability, the entropy is high. If the probabilities of occurrence are very different from each other, the entropy is low. The mutual information of two variables is defined as

$$I(Y; X) = H(Y) - H(Y|X). \tag{12}$$

It can be interpreted as the decrease in uncertainty about Y conditional on knowing X. Considering the symmetry property $I(Y; X) = I(X; Y)$ it can also be seen as the amount of information shared by X and Y. There exist several filter methods that are based on this mutual information, see Hall (1999) and Brown et al. (2012) for more information. Continuous features have to be discretized before applying these filters. In the following, we describe filters from two different toolboxes, which calculate similar scores but differ in the way they discretize the features. The filters from the toolbox *FSelectorRcpp* (Zawadzki and Kosinski, 2017) use the MDL method for discretization, while the filters from the toolbox *praznik* (Kursa, 2018) perform a discretization into equally spaced intervals. For both discretization methods see Section 2.6.

Filter *info.gain* (Zawadzki and Kosinski, 2017) uses

$$J_{\text{info.gain}}(X_k) = I(Y; X_k), \tag{13}$$

the reduction of uncertainty about the class variable Y due to feature X_k, as the score for this feature.

The score of filter *gain.ratio* (Zawadzki and Kosinski, 2017) is

$$J_{\text{gain.ratio}}(X_k) = \frac{I(Y; X_k)}{H(X_k)}, \tag{14}$$

the ratio of the mutual information and the entropy of feature X_k. Out of two features with the same information about Y, this filter favors the feature with the smaller entropy, e.g. the feature that attains less different values. The reason for

dividing by the entropy is to balance out the bias of the mutual information towards selecting features that take on many different values like credit card numbers or similar.

For filter *sym.uncert* (Zawadzki and Kosinski, 2017), the score

$$J_{\text{sym.uncert}}(X_k) = \frac{2 \cdot I(Y; X_k)}{H(X_k) + H(Y)} \tag{15}$$

is used. This score also reduces the bias towards features with many values and additionally normalizes it to the range [0,1].

Filter *MIM* (Kursa, 2018) ranks all features according to the information they share with the target variable Y

$$J_{\text{MIM}}(X_k) = I(Y; X_k). \tag{16}$$

This is the same score that filter *info.gain* uses as well. However, the filters differ in the way they discretize the features.

The following filter methods calculate the scores of all features iteratively. Thus, the features are selected in a greedy forward manner. Let S denote the set of already chosen features. S is initialized as $S = \{X_k\}$ with $I(Y; X_k) = \max_{j \in \{1,\ldots,p\}} I(Y; X_j)$. In each iteration, the feature that maximizes the respective score is added to S.

Filter *MRMR* (Kursa, 2018) uses the score

$$J_{\text{MRMR}}(X_k) = I(Y; X_k) - \frac{1}{|S|} \sum_{X_j \in S} I(X_k; X_j). \tag{17}$$

The term $I(Y; X_k)$ measures the relevance of the feature by the information this feature has about Y. The term $\frac{1}{|S|} \sum_{X_j \in S} I(X_k; X_j)$ judges its redundancy by assessing the mean information that the feature shares with the features in S. The idea is to find maximally relevant and minimally redundant (MRMR) features.

For filter *JMI* (Kursa, 2018), the score

$$J_{\text{JMI}}(X_k) = \sum_{X_j \in S} I(Y; X_k, X_j) \tag{18}$$

is employed. $I(Y; X_k, X_j)$ is the amount of information about Y that X_k and X_j provide jointly. This quantity can be calculated by using the multivariate variable $X = (X_k, X_j)'$ in the definition of mutual information in Eq. (12). The idea of this score is to include features that are complementary to the already chosen features.

Filter *JMIM* (Kursa, 2018) is a modification of filter *JMI*. The score

$$J_{\text{JMIM}}(X_k) = \min_{X_j \in S} \left\{ I(Y; X_k, X_j) \right\} \tag{19}$$

considers the minimal joint information over all already selected features instead of the sum.

For filter *DISR* (Kursa, 2018), the score

$$J_{\text{DISR}}(X_k) = \sum_{X_j \in S} \frac{I(Y; X_k, X_j)}{H(Y, X_k, X_j)} \tag{20}$$

is used. Like *JMI*, it uses the information about Y provided jointly by X_k and X_j. But additionally, this information is divided by the joint entropy of Y, X_k and X_j. To obtain this entropy, consider the multivariate variable $\widetilde{Y} = (Y, X_k, X_j)'$ and plug it into the above definition of the entropy in Eq. (10). The reason for dividing by the entropy is to avoid selecting features that e.g. attain many different values, see filter *gain.ratio*.

Filter *NJMIM* (Kursa, 2018) is a modification of filter *DISR*. Its score

$$J_{\text{NJMIM}}(X_k) = \min_{X_j \in S} \left\{ \frac{I(Y; X_k, X_j)}{H(Y, X_k, X_j)} \right\} \tag{21}$$

considers the minimal relative joint information over all already selected features instead of the sum.

Filter *CMIM* (Kursa, 2018) has the score

$$J_{\text{CMIM}}(X_k) = \min_{X_j \in S} \left\{ I(Y; X_k | X_j) \right\}. \tag{22}$$

It uses the conditional mutual information

$$I(Y; X_k | X_j) = H(Y | X_j) - H(Y | X_k, X_j) \tag{23}$$

that can be interpreted as the difference in uncertainty about Y before and after X_k is known, while X_j is known anyway. The idea is to select features that provide much information about the class variable, given the information of the already selected features.

2.6. Discretization

Fayyad and Irani (1993) define the minimal description length (MDL) discretization method for continuous features. Their discretization method works recursively: A feature is split at an optimal cut point into two categories. Then, these categories are split recursively until a stopping condition is reached. Let $a_1 < \cdots < a_m$ denote the values of the feature to be discretized. Then the points that are considered as cut points are $\frac{1}{2}(a_1 + a_2), \ldots, \frac{1}{2}(a_{m-1} + a_m)$. The criterion for determining which of these cut points is optimal is the difference in entropy of the class variable before and after the split. The cut point with the greatest decrease in entropy is considered the best cut point. However, this split is only made if the decrease in entropy exceeds a boundary. This boundary is chosen such that the decrease in entropy is greater than the boundary if and only if the costs of performing the split are lower than the costs of not performing it. The costs are assessed by the minimal description length, which is an information theoretic measure. For details and formulas see Fayyad and Irani (1993). Not carrying out a split because of falling below the boundary works as stopping condition. When all recursions have stopped, some cut points $\tilde{a}_1 < \cdots < \tilde{a}_k$ are selected. The feature is then discretized into the categories $(-\infty, \tilde{a}_1], (\tilde{a}_1, \tilde{a}_2], \ldots, (\tilde{a}_k, \infty)$. Note that this method discretizes all values of a feature into one single category if the best cut point does not cause enough decrease in entropy.

A different method to discretize a numeric feature is to simply cut the range of values into q equally spaced intervals and use these intervals as categories. The number of intervals is determined as $q = \max\left\{\min\left\{\frac{n}{3}, 10\right\}, 2\right\}$ where n is the number observations in the data set (Kursa, 2018).

2.7. Overview of all considered filter methods

Table 1 gives an overview of all considered filter methods. If the target variable can be of type multi-class, also binary class variables are allowed. If categorical features are required, numeric features can be used as well. They are automatically discretized by the filter methods. All filter methods are available in the machine learning R package *mlr* (Bischl et al., 2016). If for the implementation in *mlr* other R packages are used, they are indicated.

2.8. Stability of feature selection

The stability of feature selection is defined as the robustness of the set of selected features with respect to different training data sets drawn from the same distribution (Kalousis et al., 2007). To quantify stability, stability measures are used. The stability measure that performs best both in theoretical (Nogueira and Brown, 2016) and in empirical (Bommert et al., 2017) comparisons is the Pearson correlation SC:

Assume that there is a data set containing n observations of the p features $X_1, \ldots, X_p$. Resampling is used to split the data set into m subsets. The feature selection method is then applied to each of the m subsets. Let $V_i \subset \{X_1, \ldots, X_p\}$, $i = 1, \ldots, m$, denote the set of chosen features for the ith subset of the data set. For each set of selected features V_i the vector $z_i \in \{0, 1\}^p$ is defined to indicate which features are chosen. The jth component of z_i is equal to 1 iff V_i contains X_j, i.e. $z_{ij} = \mathbb{I}_{V_i}(X_j)$, $j = 1, \ldots, p$. The resulting stability measure is

$$SC = \frac{2}{m(m-1)} \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} \mathrm{Cor}(z_i, z_j) \tag{24}$$

with $\mathrm{Cor}(z_i, z_j)$ denoting the Pearson correlation between z_i and z_j (Nogueira and Brown, 2016). The Pearson correlation measures the linear association between continuous variables. When applied to binary data like the vectors $z_1, \ldots, z_m$, the Pearson correlation is equivalent to the ϕ-coefficient for the contingency table of each two of these vectors (Rasch et al., 2011, pp. 339 f.). SC takes on values in the interval $[-1, 1]$. A value of 1 means maximal stability, 0 indicates that the feature selection method is as stable as a random feature selection.

2.9. Assessment of predictive performance

To evaluate the predictive performance of a classification method on a binary classification data set, the accuracy is defined as

$$\text{accuracy} = \frac{\text{number of correctly classified instances}}{\text{number of all classified instances}}. \tag{25}$$

The accuracy takes on values in the interval $[0, 1]$. The higher the accuracy, the better the predictive performance of the classification method.

Table 1
Names in this paper and in *mlr* and general concepts of the filter methods, requirements on the target variable and on the features, and R packages in which the filter methods are implemented. If categorical features are required, numeric features are discretized.

Name	Concept	Target	Features	R package
anova.test (anova.test)	univariate statistical test	multi-class	numeric or binary	*mlr* (Bischl et al., 2016)
limma (available on GitHub)	univariate statistical test	multi-class	numeric or binary	*limma* (Ritchie et al., 2015)
sam (available on GitHub)	univariate statistical test	multi-class	numeric or binary	*samr* (Tibshirani et al., 2011)
kruskal.test (kruskal.test)	univariate statistical test	multi-class	numeric or binary	*mlr* (Bischl et al., 2016)
chi.squared (FSelector_chi.squared)	univariate statistical test	multi-class	categorical	*FSelector* (Romanski and Kotthoff, 2016)
auc (auc)	univariate predictive performance	two-class	numeric	*mlr* (Bischl et al., 2016)
oneR (FSelector_oneR)	univariate predictive performance	multi-class	categorical	*FSelector* (Romanski and Kotthoff, 2016)
univariate.model.score (univariate.model.score)	univariate predictive performance	depends on model, here: multi-class	depends on model, here: numeric or categorical	depends on model, here: *rpart* (Therneau et al., 2017)
variance (variance)	feature variance	arbitrary	numeric	*mlr* (Bischl et al., 2016)
cforest.importance (party_cforest.importance)	random forest importance	multi-class	numeric or categorical	*party* (Strobl et al., 2008)
permutation (ranger_permutation)	random forest importance	multi-class	numeric or categorical	*ranger* (Wright and Ziegler, 2017)
impurity (ranger_impurity)	random forest importance	multi-class	numeric or categorical	*ranger* (Wright and Ziegler, 2017)
info.gain (FSelectorRcpp_ information.gain)	mutual information	multi-class	categorical	*FSelectorRcpp* (Zawadzki and Kosinski, 2017)
gain.ratio (FSelectorRcpp_ gain.ratio)	mutual information	multi-class	categorical	*FSelectorRcpp* (Zawadzki and Kosinski, 2017)
sym.uncert (FSelectorRcpp_ symmetrical.uncertainty)	mutual information	multi-class	categorical	*FSelectorRcpp* (Zawadzki and Kosinski, 2017)
MIM (praznik_MIM)	mutual information	multi-class	categorical	*praznik* (Kursa, 2018)
MRMR (praznik_MRMR)	mutual information	multi-class	categorical	*praznik* (Kursa, 2018)
JMI (praznik_JMI)	mutual information	multi-class	categorical	*praznik* (Kursa, 2018)
JMIM (praznik_JMIM)	mutual information	multi-class	categorical	*praznik* (Kursa, 2018)
DISR (praznik_DISR)	mutual information	multi-class	categorical	*praznik* (Kursa, 2018)
NJMIM (praznik_NJMIM)	mutual information	multi-class	categorical	*praznik* (Kursa, 2018)
CMIM (praznik_CMIM)	mutual information	multi-class	categorical	*praznik* (Kursa, 2018)

3. Experiments and results

In this section, we conduct and evaluate experiments to compare and to benchmark the filter methods described in Section 2. In Section 3.1, all data sets and their properties are given. Section 3.2 provides information about the classification methods used in the experiments. In Sections 3.3 and 3.4, we explain the setup of the experiments and analyze their results in detail. We perform two analyses. The first analysis in Section 3.3 aims to find out which filter

Table 2
Name, number of features, number of instances, relative size of majority class and source of the data sets, sorted by number of features. All data sets are binary classification data sets and contain only numeric features.

Name	p	n	Majority.perc	Source
scene	299	2 407	0.82	OpenML, ID 312
madelon	500	2 600	0.50	OpenML, ID 1485
gina_prior	645	3 468	0.51	OpenML, ID 1042
gina_agnostic	970	3 468	0.51	OpenML, ID 1038
christensen	1 413	198	0.57	datamicroarray
Internet-Advertisements	1 558	3 279	0.86	OpenML, ID 40 978
hiva_agnostic	1 617	4 229	0.96	OpenML, ID 1039
Bioresponse	1 776	3 751	0.54	OpenML, ID 4134
gravier	2 905	168	0.66	datamicroarray
gisette	4 971	7 000	0.50	OpenML, ID 41 026
chiaretti	12 625	111	0.67	datamicroarray
tian	12 625	173	0.79	datamicroarray
yeoh	12 625	143	0.55	datamicroarray
chin	22 215	118	0.64	datamicroarray
burczynski	22 283	101	0.58	datamicroarray
chowdary	22 283	104	0.60	datamicroarray

methods are similar with respect to feature ranking. We also compare the scaling behavior of the filter methods. In the second analysis in Section 3.4, we investigate the performance of the filter methods with respect to classification accuracy and run time.

3.1. Data sets

For our analyses, we use 16 large classification data sets from various domains. Information about these data sets is displayed in Table 2. The selected data sets are from *OpenML* (Vanschoren et al., 2013; Casalicchio et al., 2017) and the R package *datamicroarray* (Ramey, 2016). The data sets from *OpenML* have been selected according to the following criteria: They have exactly two classes, at least 250 features, more instances than features and they have no missing values or nominal features (binary features are converted to numeric features). Also, the smaller class should consist of at least 3% of the instances. For run time reasons, the product of the number of features and the number of instances should not exceed 100 000 000.

The data sets from the R package *datamicroarray* are high-dimensional data sets and contain more features than instances. We use data sets with two classes and create two-class data sets out of the data sets with more classes by selecting only the instances that belong to one of the two largest classes. We only take into account resulting data sets with at least 100 instances and without missing values. For the data sets *burczynski*, *chowdary*, and *tian* we perform a $\log(x+1)$ transformation to normalize the features. The other microarray data sets in Table 2 are already normalized. We omit all data sets where a normalization seems to be necessary, but for which the usual $\log(x+1)$ transformation is not possible due to negative values. Constant features are removed from all data sets.

The data sets contain 299 to 22 283 features and 101 to 7000 instances. The class imbalance reaches from 0.50 (perfect balance) to 0.96 (data set consists almost only of instances of the larger class).

3.2. Classification methods

We employ three classification methods to determine the predictive performance of subsets of features selected by a filter method. We choose specifically these classification methods because they are popular methods that do not perform embedded feature selection. This is important for judging the direct impact of the filter on prediction performance, not in combination with the subsequent classification algorithm that might perform an additional embedded feature selection. There are other state-of-the-art methods for feature selection like Lasso regression, but in this study we focus on filter methods. Also, for very high-dimensional data sets (several hundred thousand features) these methods may need to be applied to a pre-filtered data set in order to achieve acceptable run time or memory consumption. Table 3 displays the names of the methods, the corresponding hyper parameters that must be set and the R package from which the implementation is taken.

k Nearest Neighbors classifies a new instance by a majority vote of the k closest instances (Larose and Larose, 2014, pp. 149 ff.). In order to obtain an ordinary unweighted KNN method, we have to set the parameter *kernel* to *rectangular*.

Table 3
Classification methods with corresponding hyper parameters and R package of which the implementation is used.

Method	R package	Hyper parameters
k Nearest Neighbors (KNN)	kknn (Schliep and Hechenbichler, 2016)	$k \in \{1, 2, \ldots, 20\}$
Logistic Ridge Regression (LRR)	glmnet (Simon et al., 2011)	$\lambda \in \{2^x : x \in [-15, 15]\}$
Support Vector Machine (SVM)	kernlab (Karatzoglou et al., 2004)	$C, \sigma \in \{2^x : x \in [-15, 15]\}$

Logistic Ridge Regression is logistic regression combined with a ridge penalty (Izenman, 2013, pp. 150 ff.). The ridge parameter λ balances the goodness of fit (log likelihood) and the size of the regression parameters. For $\lambda = 0$ this is ordinary logistic regression without the ridge penalty. If there are more features than instances in a data set or if there is a hyperplane in feature space which perfectly separates the two classes, $\lambda = 0$ is not feasible. Large values of λ cause all regression coefficients to be shrunken towards 0.

Support Vector Machines use the hyperplane in feature space that is optimal with respect to the maximum margin principle as decision boundary. Kernel functions are used to change the shape of the hyperplane into something non-linear (Izenman, 2013, pp. 369 ff.). We use Support Vector Machines with RBF kernel. There are two hyper parameters: the regularization parameter C and the kernel width parameter σ.

3.3. Similarity of the filter methods

3.3.1. Feature ranking

The goal of this analysis is finding out which filter methods are similar with respect to feature ranking. The question behind this analysis is if the filter methods can be grouped into sets with similar behavior and if these groups contain redundant information such that some members of each group can be neglected. For each data set, we compute the filter scores for all features. For the iterative *praznik* filters, we assess the selection order instead of the filter scores because the scores of different iterations are not comparable.

To assess the similarity of the filter methods, we compare the orders in which they select features. For each data set and each filter method, we determine the selection order of all features. Then we compute the rank correlation between the orders of all pairs of filter methods. This calculation is performed separately for each data set. The results are displayed in Section 1 of the supplementary material. To draw conclusions based on all data sets, we average the results for all data sets with the arithmetic mean. Fig. 1 displays the mean rank correlations between all pairs of filter methods. The higher the rank correlation between two filter methods, the more similar they are.

Fig. 1 shows three groups of similar filter methods. The first group consists of 6 out of the 7 *praznik* filters. The second group is formed by the filters *kruskal.test*, *auc*, *limma*, *anova.test*, and *sam*. The third group contains the filters *info.gain*, *chi.squared*, *sym.uncert*, *oneR*, and *gain.ratio* from the toolboxes *FSelector* and *FSelectorRcpp*. The other filter methods are not similar to any other filter method.

The average similarity value in is 0.5028. The highest mean rank correlations are observed between *anova.test* and *limma* (0.9998), *info.gain* and *chi.squared* (0.9989), *info.gain* and *sym.uncert* (0.9970), *limma* and *sam* (0.9954), *chi.squared* and *sym.uncert* (0.9949), *anova.test* and *sam* (0.9946), *kruskal.test* and *auc* (0.9676), *JMIM* and *JMI* (0.9520), *kruskal.test* and *limma* (0.9198), *kruskal.test* and *anova.test* (0.9195), and *kruskal.test* and *sam* (0.9148).

The similarity of the filters *kruskal.test*, *auc*, *anova.test*, *limma*, and *sam* is easy to understand. The Kruskal–Wallis test can be seen as the non-parametric equivalent of the analysis of variance. The filters *limma* and *sam* perform a moderated version of the F test conducted by filter *anova.test*. Also, there are strong links between the AUC and the Kruskal–Wallis test (Hanley and McNeil, 1982).

Considering the group of similar *praznik* filters, it appears plausible that they select features in a similar order because their scores are all based on mutual information. Within this group, *JMI* and *JMIM* as well as *DISR* and *NJMIM* are especially similar to each other. This makes sense as the scores are modified versions of each other, see Section 2. The filter *MRMR* comes from the same toolbox, but does not appear to be very similar to the other *praznik* filters. In contrast to the other *praznik* filters, *MRMR* also considers mutual information between features, see Section 2.

MIM and *info.gain* use the same score. However, the different discretization methods used by the two toolboxes cause an average rank correlation of only 0.6290 between the two filter methods. The filters from the toolboxes *FSelector* and *FSelectorRcpp* use different filter criteria but all employ the same discretization method. This makes it seem likely that the similarity of these filter methods is due to the same discretization method.

Considering the *ranger* toolbox, the filters *impurity* and *permutation* are not very similar (mean rank correlation 0.4582). Although both of them use a random forest to assess the feature importances, the resulting feature rankings are quite different on average.

All of the previous analyses are based on the aggregated rank correlations. A brief analysis of the plots in Section 1 of the supplementary shows that the similarity structure displayed in Fig. 1 does not represent the similarity structure for all single data sets. Instead, there are mainly two groups of data sets. The first group contains the high-dimensional data sets from the R package *datamicroarray* and some other data sets. The second group is formed by data sets from various

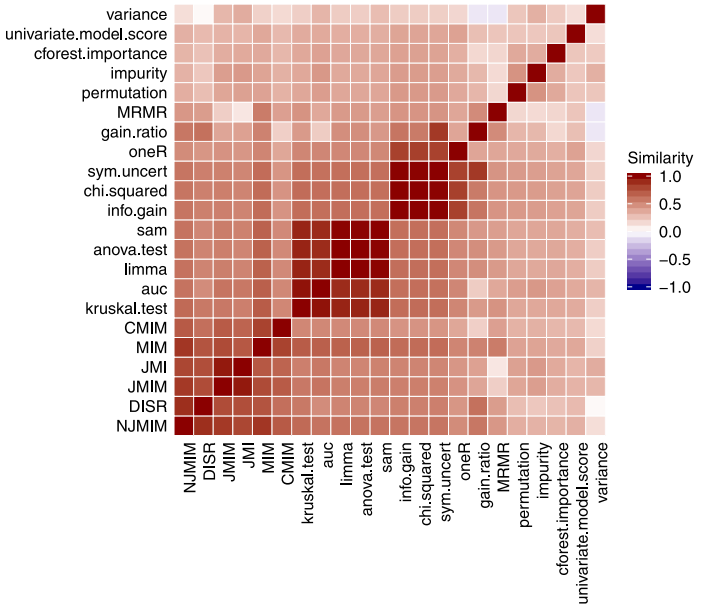


Fig. 1. Rank correlations between the selection order of all pairs of filter methods on all data sets, averaged by the arithmetic mean. The filter methods are ordered by average linkage hierarchical clustering using the mean rank correlation as similarity measure.

domains. In the first group, the similarity structure strongly resembles the one in Fig. 1. In the other group, most filter methods select the features in a similar order. There are only few filter methods that are not similar to the other methods, and these methods differ between the data sets of the second group.

For deciding which of the filter methods in the groups of similar filters can be neglected, we refer to the analysis in Section 3.4 where the filter methods are compared with respect to their performance.

3.3.2. Run times

A similarity analysis of the filter methods with respect to their run times and scaling behaviors can be found in Section 3 of the supplementary material.

3.4. Optimal filter methods

In this analysis, we investigate the performance of the filter methods with respect to predictive performance and run time.

3.4.1. Setup

To determine the predictive performance of the features chosen by the filter methods, we combine each filter method from Section 2 with a selected classification method from Section 3.2. The combined methods first apply the filter, choosing a given percentage of features, and then learn the classification rule using only the remaining features.

To ensure a fair comparison of the filter methods, we compare the performances of the filters with the best classifier and the best hyper parameter settings we found. More precisely, for each filter method and each data set, we tune the classification part and the percentage of features to be chosen simultaneously. We consider the classification method as a hyper parameter as well, which gives us a hierarchical search space. Possible configurations of this search space are for example (12% of features, KNN, $k = 7$) or (94% of features, SVM, $C = 2^{-1.3}$, $\sigma = 2^{13.28}$). We allow percentages of features to be chosen in the range [0%, 100%]. The ranges of the classification hyper parameters are given in Table 3. To find the best configuration, we conduct a random search with budget 100. To obtain unbiased estimates of the performances of the filters, we perform nested cross-validation with 10 outer and 10 inner iterations (Bischl et al., 2012). We make sure that the considered configurations are identical for all filter methods on the same data set in the same outer iteration, in order not to favor any filter method because it happens to be evaluated in combination with better configurations. Also, all filter methods use the same cross-validation splits. The experiments are conducted on a high performance compute cluster in a randomized order using the R package *batchtools* (Lang et al., 2017).

Table 4
Performance criteria and measures for evaluating the filter methods combined with a classification method.

Performance criterion	Performance measure	Aggregation across the 10 outer cross-validation iterations
Predictive Accuracy	Accuracy (see Section 2.9)	Arithmetic mean
Run time	Time for filtering	Median
Run time	Time for training the combined model (filter and classification method)	Median

For each filter method and each data set, we do the following: In each outer iteration, 10% of the data set is used as evaluation data. On the remaining 90% of the data, 10-fold cross-validation is conducted to estimate the performance of 100 randomly drawn configurations. We select the best configuration based on maximal mean accuracy. Ties are resolved by selecting the configuration with the smallest run time (median time for training the combined model). We use the median to aggregate the run times in order to obtain an estimate that is robust against variation caused by the high performance compute cluster. This tuning procedure makes sense because in practice, one is interested in filter methods that allow a good predictive performance, and among these methods, one is interested in methods with short run times. The selected configuration is then evaluated on the evaluation data, calculating the accuracy and the time for training the combined model. Additionally, we measure the time that is needed for filtering only. This way, for each data set and each filter, we obtain 10 evaluations of the best methods from the 10 outer iterations. We aggregate these values by calculating the mean accuracy and the median run times. In the end we have three performance values (mean accuracy, median time for filtering, and median entire run time) per filter method and data set. The performance criteria and measures are summarized in Table 4.

As an important baseline, we compare the results of the filters to results without filtering. Again, we perform a random search with a budget of 100 evaluations and nested cross-validation with 10 outer as well as 10 inner iterations. We calculate the same performance measures and select the best methods with respect to the same criteria. The only difference is that there is no need for tuning a filter percentage and that the run time consists only of the time for training the classification model because no filtering is done.

There are two configurations for which no results can be obtained. As all filters try the same configurations, this means that for all filters the results for the corresponding configuration are missing. One of the configurations is tried on data set *gina_agnostic*, the other one on *madelon*. The failing of both configurations is caused by the selection of less than two features by the filters and the implementation of Logistic Ridge Regression in *glmnet* that requires at least two features. We ignore the two configurations for which no results were obtained for the analyses.

3.4.2. Results

First, we compare the filter methods only with respect to the predictive performance. Fig. 2 shows for two data sets the accuracies of the 10 best configurations corresponding to the 10 outer cross-validation iterations, separately for all filter methods. Remember that the performances are evaluated on data that is not used for tuning. Therefore, the performance values may be interpreted as unbiased estimates of the performances on new data from the data generating process that created the respective data set. The results for the other 14 data sets are displayed in Section 2 of the supplementary material.

The left plot in Fig. 2 shows that for data set *gina_agnostic*, some filters perform considerably better than others. The filters *JMIM*, *permutation*, *impurity*, and *cforest.importance* perform quite well while the filters *DISR*, *MRMR*, and *variance* perform comparably bad. All filters lead on average to a better predictive performance than not filtering at all. The right plot in Fig. 2 demonstrates that for data set *gravier* there are only little differences in the central locations of the accuracies.

In Section 2 of the supplementary material, it can be observed that also for most of the other data sets there are only little differences between the majority of the filter methods with respect to the location of the predictive performance taking into account the spread. For the data sets *scene*, *madelon*, *gina_prior*, *gisette*, *chiaretti*, and *burczynski*, however, some noticeable differences can be observed. The filter methods *permutation*, *impurity*, and *cforest.importance*, lead to comparably high accuracy on several of these data sets. For the filters *JMIM*, *JMI*, *MRMR*, *NJMIM*, *variance*, and no filter a comparably good performance is observed for at least one of them. Not filtering at all, *MRMR*, and *DISR* lead to comparably low predictive accuracy on most of the data sets. The filters *variance*, *oneR*, and *univariate.model.score* perform bad on at least one of them.

Fig. 3 shows the number of times that the filter methods outperform each other. The number displayed in the row of filter A and the column of filter B indicates the number of data sets on which filter A is better than filter B with respect to mean accuracy. Given that two filter methods are equally good but never have exactly the same performance, we expect that each of the filters outperforms the other on approximately 8 of the 16 data sets.

We mark in red when a filter method is better than another filter method more often than 8 times and in blue when this happens less than 8 times. Note that there are several ties with respect to the accuracy of the filter methods. It can

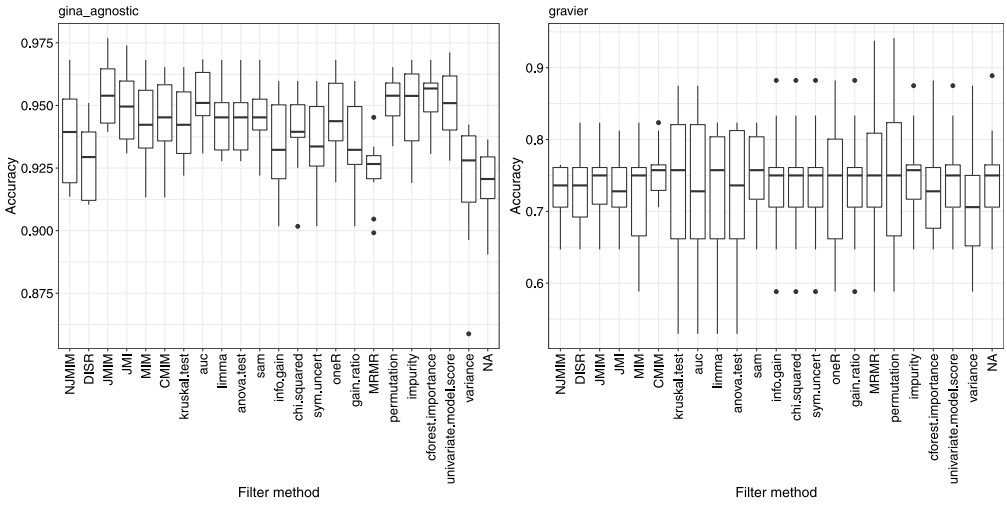


Fig. 2. Boxplots of the accuracies of the best configurations in the 10 outer cross-validation iterations per filter method and data set.

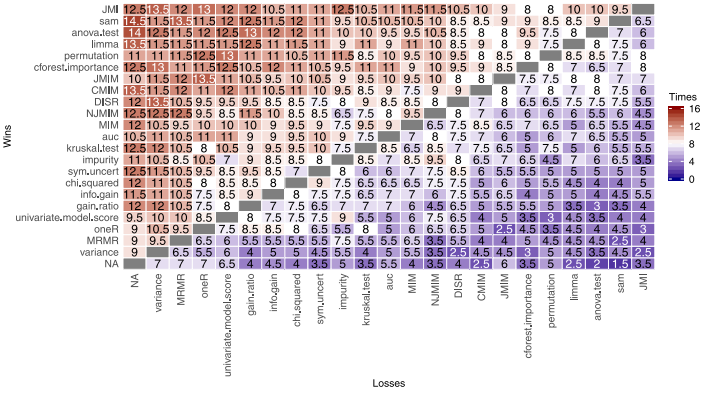


Fig. 3. Number of data sets on which the filter method in the row has a higher mean accuracy than the filter method in the column. Ties are counted as 0.5 for both filters. The filter methods are sorted decreasingly by row sums.

be observed that most filter methods are better than filter *variance* and not filtering at all on most data sets. Also, filter *MRMR* is outperformed by most of the other filters on many data sets. Filters *JMI*, *sam*, *anova.test*, *limma*, *permutation*, and *cforest.importance* are only outperformed by few other filter methods and they outperform many other filter methods themselves on most data sets.

Concluding, there is no filter method that is better than all the other methods on all data sets. However, there are some filters like *permutation* and *cforest.importance* that achieve high accuracy on some data sets and only have a comparably poor performance on few data sets in the analysis.

Next, we compare the filter methods with respect to predictive performance and run time across data sets. For the analysis presented in Fig. 3, we only looked at good or bad predictive performances in comparison to the other filter methods. This analysis also takes into account how much the filter methods differ in performance. We aggregate the performance values of the 10 outer cross-validation iterations into one value per performance measure. Remember that we analyze the accuracy of the best configuration found and the time for fitting the best model. In the following, we first investigate the run time for filtering only and later, we consider the entire run time for filtering and fitting the best classification model.

Fig. 4 displays the mean accuracy and the median run time for filtering of the filter methods separately for all data sets. Fig. 5 aggregates the information of Fig. 4 into one graphic. The right plot focuses on parts of the left plot. To compare the

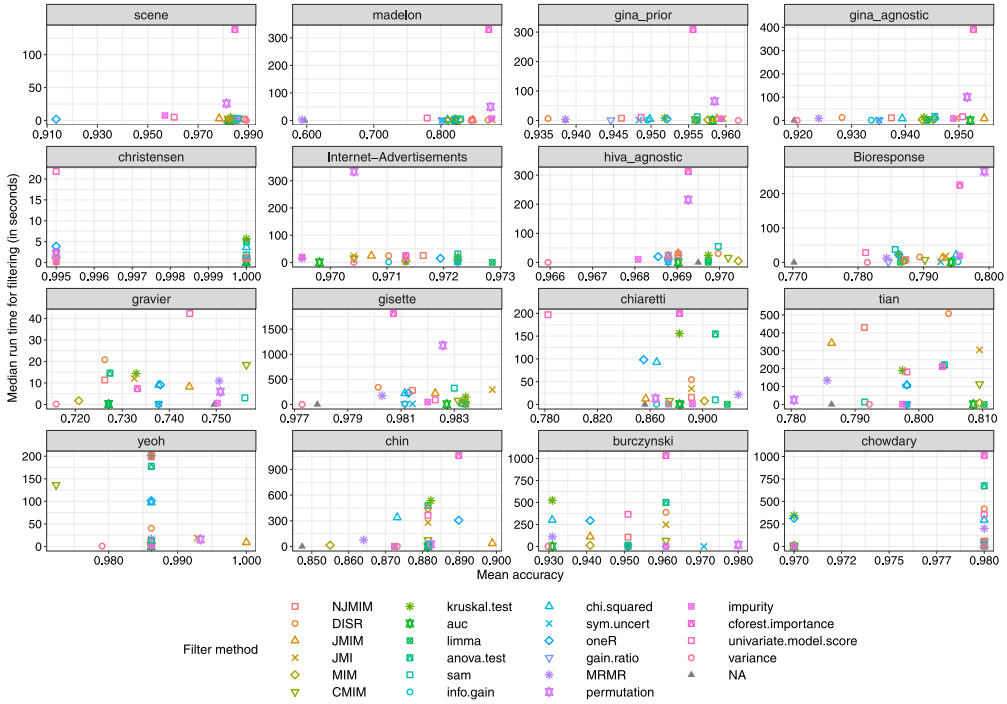


Fig. 4. Mean accuracy and median run time for filtering of filter methods with optimal configurations per data set.

predictive accuracy across data sets, relative accuracies are considered. More precisely, the differences between the mean accuracies of all filter methods to the highest mean accuracy observed on the same data set are considered. Therefore, the smaller the relative accuracy, the better the predictive performance of the filter method when combined with the best configuration. The best filter method per data set always has relative mean accuracy 0. For the time for filtering, the fastest run time (not filtering at all) is already 0 for all data sets. The (transformed) performance criteria are then aggregated over the data sets by computing the median values as well as the upper and lower quartiles. The median value provides information about the central location of the performance measures. The distance of the quartiles shows the variation of the performances across data sets. Fig. 5 displays the median of the performance criteria values for each filter method by a symbol. The quartiles are located at the respective ends of the horizontal and vertical lines. The longer the lines, the greater the variation across data sets. The optimal filter method would be located in $(0, 0)'$.

The left plot in Fig. 5 shows that the relative accuracy of the filter methods and the time for filtering vary across data sets. The interquartile range of the relative mean accuracy across data sets is around 0.02 for most filter methods. As this is a small number, it means that the median value summarizes the central location well. However, the variation is too large to draw conclusions about which filter methods perform best only based on the median. The left plot also shows that filter *cforest.importance* needs much longer for filtering than the other filter methods and that its accuracy is among the best, but not the single best.

The right plot shows the median relative accuracy values and the variation in accuracy for all filter methods except for *cforest.importance*. The variation in time for filtering is omitted because only a small part of the y-axis is investigated. Considering the median performances in both criteria, it can be seen that the filters *JMIM*, *impurity*, *auc*, *limma*, and *variance* as well as not filtering at all are Pareto optimal. Pareto optimality means that there is no other method that performs as good in all criteria and better in at least one criterion. Considering the non Pareto optimal filter methods in Fig. 5, at least one of the Pareto optimal filters performs better in both criteria. Filter *JMIM* is Pareto optimal because it has the best median accuracy among all filter methods. Filter methods *impurity*, *auc*, and *limma* have a good median accuracy and a low median run time for filtering. The *variance* filter and not filtering at all are Pareto optimal because of their fast run time. Applying no filter takes no time, which is always faster than applying any filter and therefore this is always Pareto optimal, independent of its accuracy.

If we now consider the upper quartile of the relative mean accuracy, i.e. the right ends of the horizontal lines, as measure of the predictive accuracy instead of the median value, we obtain a different set of Pareto optimal methods. In

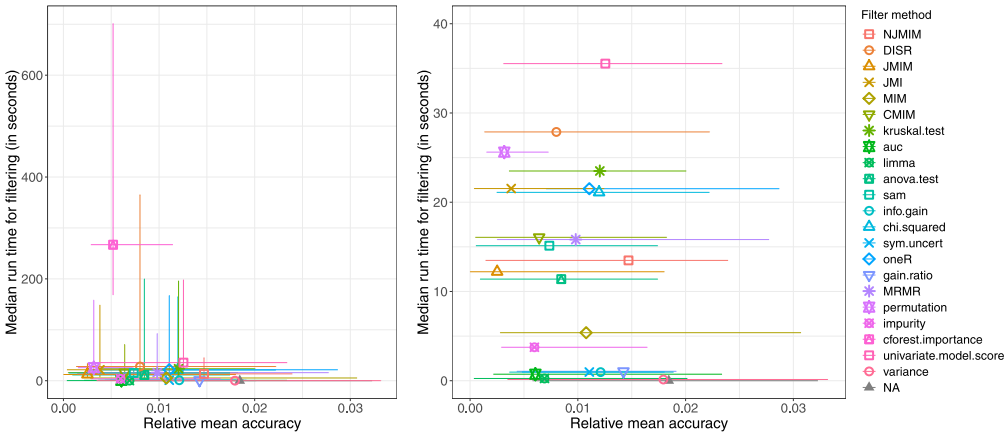


Fig. 5. Relative mean accuracy and median run time for filtering of the filter methods with optimal configurations aggregated over all data sets. The median of both performance measures (relative mean accuracy and median filtering time) across all data sets is displayed by a symbol. The upper and lower quartiles are located at the respective ends of the horizontal and vertical lines. The right plot is a part of the left plot: it focuses on small run times. The optimal filter method would be located in $(0, 0)'$.

this case, *permutation*, *JMI*, *impurity*, *sym.uncert*, *info.gain*, *limma*, and no filter are Pareto optimal. Especially *permutation* but also *JMI* and *impurity* are Pareto optimal because of high accuracy. The others are Pareto optimal because of low run times. Filter *impurity* seems to provide a good compromise of accuracy and run time for filtering. It should be noted that we have selected the configurations based on accuracy. Because the run times of the filters from package *praznik* depend on the number chosen features (see Section 3 of the supplementary material), it might be possible to make them Pareto optimal by selecting fewer features. This would make them faster at the cost of predictive accuracy. As the run times depend on the implementation, any filter method could potentially become Pareto optimal if it were implemented efficiently enough.

Now we analyze the predictive accuracy in combination with the entire run time for filtering and fitting the best model. Considering the entire run time makes sense because it also assesses how long it takes to fit the best classification model with the features selected by the filter. As most filters rank all features and because there are groups of filters with very similar run times (see Section 3 of the supplementary material), it is likely that many filters achieve similar run times even if they choose different numbers of features. However, selecting more features may have a huge impact on the run time for fitting the predictive model of interest.

Because we observe some very long run times in this analysis, we consider the logarithmic run times instead. To be able to apply the logarithm, we have to add a small constant to all run times, as KNN without filtering can be so fast that its run time is measured as 0. For comparing the predictive accuracy and run time across data sets, relative accuracies and relative logarithmic run times are considered. A relative mean accuracy of x means that the mean accuracy of the filter (with the best configuration found) is worse than the mean accuracy of the best filter on the same data set, by an additive factor of x. A relative logarithmic median run time of x means that the median run time of the filter equals the median run time of the best filter on the same data set multiplied with 10^x.

Fig. 6 is an analogous plot to Fig. 5 but shows the logarithmic run time for fitting the best model instead of only the run time for filtering. In comparison to Fig. 5, the relative run time for *permutation* is lower. For the filters *variance* and *anova.test* and not filtering at all, the relative run time is higher than in Fig. 5. When considering only the median values of the performance measures, the filter methods *JMIM*, *impurity*, *auc*, and *limma* are Pareto optimal. With respect to the upper quartile of the relative mean accuracy and the median of the relative logarithmic run time, *permutation*, *impurity*, *sym.uncert*, *info.gain*, and *limma* are Pareto optimal. The sets of Pareto optimal methods here are subsets of the Pareto optimal methods in the analyses based on the run time for filtering only. The filter methods that are not Pareto optimal any more when considering the entire run time are *variance* and no filter, which have a very short time for filtering, as well as *JMI*, which is outperformed by *permutation* with respect to the time for model fitting.

Like in the discussion of Fig. 5, it also has to be mentioned that filter methods could become Pareto optimal by sacrificing accuracy and thus saving run time. Here, this is possible for all filter methods by selecting a configuration that is faster but provides less predictive accuracy. Also, the variation in the performance criteria should be taken into account when interpreting the Pareto optimal methods: there is no subset of filter methods that is clearly better than the others. Filter methods like *permutation* and *impurity* seem favorable because of a comparably good performance and a comparably low variation.

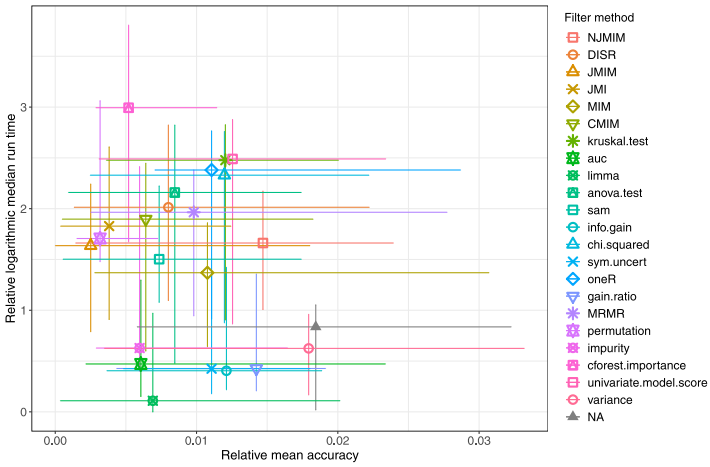


Fig. 6. Relative mean accuracy and relative logarithmic median run time of the filter methods with optimal configurations aggregated over all data sets. The median of both performance measures (relative mean accuracy and relative logarithmic median run time) across all data sets is displayed by a symbol. The upper and lower quartiles are located at the respective ends of the horizontal and vertical lines. The optimal filter method would be located in $(0, 0)'$.

To sum up, there is no subset of filter methods that outperforms all other filter methods. Which filters work best depends on the data set. The results are subject to variation within and across data sets. Nevertheless, some filter methods like *permutation*, *impurity*, *sym.uncert*, *info.gain*, *limma*, *auc*, and *JMIM* seem to work well in many data situations.

Fig. 4 allows a comparison of the two studies in Sections 3.3 and 3.4. It shows for each data set the mean accuracy and the run time for filtering for all filter methods combined with the best configurations. It can be observed that the filter methods that are similar according to our analysis in Section 3.3.1 often have a similar predictive performance when applied on the same data set. However, in some cases even very similar filter methods lead to models with quite different predictive accuracy. This can be explained by the fact that the inclusion or omission of a single relevant feature can already cause big differences in accuracy. Also, filters that needed a similar amount of time for filtering in Section 3 of the supplementary material often have similar run times for filtering on the other data sets as well. Regarding the question whether some of the filter methods can be neglected when searching for a good filter method, it seems reasonable to limit the search space to *permutation*, *impurity*, *sym.uncert*, *limma*, and *JMIM*. With respect to the analyses in Section 3.3, the filters *JMIM*, *limma*, *sym.uncert* can be interpreted as representatives of their groups.

3.4.3. Stability

Another aspect for the comparison of filter methods is their stability. In Section 3.4.2, we have conducted a random search for a good configuration with respect to accuracy for each data set and each filter method. We have performed a nested cross-validation with 10 outer iterations. For each outer iteration, we have chosen one best configuration, which results in 10 configurations per data set and filter method.

Now, we analyze the stability of the feature selection for each filter method on each data set. To do so, we compare the 10 respective sets of selected features and quantify the stability with the stability measure *SC*, see Section 2.8. For comparability across data sets, we transform each stability value x into $x + 1 - m$ where m is the maximal stability value observed for any of the filters on the same data set. This sets the best stability value on each data set to 1. With respect to stability, high values are desirable. When no filter is applied, no feature selection is performed and the stability of feature selection cannot be assessed. *SC* is not defined in this situation.

Fig. 7 shows boxplots of the relative stability values of the filter methods. We observe that all filter methods from the toolbox *FSelectorRcpp* as well as the filters *chi.squared*, *auc*, and *sam* achieve comparably high stability. From our analyses in Section 3.4.2 we know that most of these filter methods are fast in computation. Filters *permutation* and *impurity*, which showed good results in Section 3.4.2 with respect to predictive performance, attain comparably low stability values.

The comparably low stability of the random forest based filter methods and of filter *univariate.model.score* can be explained with the stochasticity of these methods. When a random forest model is fitted to a data set, the output is not deterministic, because the construction of each tree involves a random choice of observations and features. So, when fitting a random forest several times to the same data set, the resulting models are generally a bit different from each other. Now, when fitting random forest models to slightly different data sets, the resulting models are expected to vary even more. With respect to filter *univariate.model.score*, the choice of train and test data is stochastic. When this filter

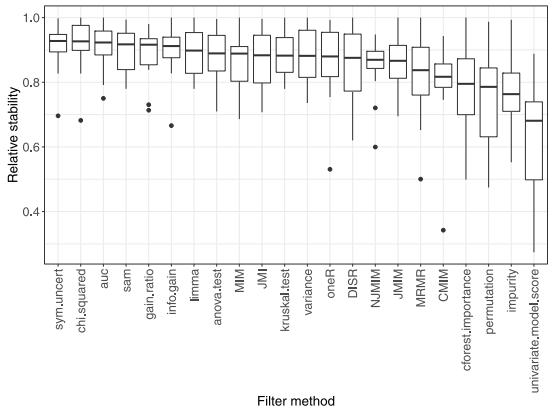


Fig. 7. Relative stability of the filter methods with optimal configurations for all data sets, sorted by median relative stability.

is applied several times to the same data set, the results can be different due to different train–test splits. For slightly different data sets, even larger differences are expected.

4. Summary and conclusions

Feature selection is a key part of data analysis, machine learning and data mining. There are many methods for feature selection, but it is unclear which of these methods perform best. In this analysis we focused on the comparison of filter methods for feature selection. We analyzed 22 filter methods based on 16 high-dimensional classification data sets from various domains.

To find out which of the filter methods are similar with respect to the order in which they rank the features, we computed rank correlations. We observed that for some data sets, especially the data sets containing gene expression data with large number of features, there were three groups of similar filter methods and many filter methods that were not similar to any other method. The filters that were similar to each other mostly came from the same toolboxes. For the other data sets, most filter methods were very similar. Also, we investigated the scaling behavior of the filter methods, identifying groups of filters that behave similarly with respect to run time.

Next, we analyzed the classification accuracy of the features selected by the filter methods and the run time needed for feature selection and for building a good classification model based on the selected features. We found out that there is no subset of filter methods that performs better than the rest of the filter methods on all data sets. Instead, the best filter methods differed between the data sets. Nevertheless, on average all filter methods performed better than not filtering at all. Also, even though there was no clear winner, the random forest importance filter methods *permutation* and *impurity*, the information theoretic filter methods *sym.uncert* and *JMIM*, and the univariate statistical test filter method *limma* performed well on most data sets, which makes them seem advisable. The filters *sym.uncert*, *limma*, and *JMIM* can be seen as representatives of the three groups of similar filter methods. Filters *impurity*, *sym.uncert*, and *limma* achieved very low run times. For filter *permutation* we observed comparably high run times but also very high predictive accuracy.

Choosing the best filter method for a new data set is a matter of available computational resources. Based on our analyses, we come to the following conclusions. If the computational resources only allow trying one filter method, we recommend *permutation*. This filter method allowed fitting classification models with high accuracy on most of the data sets. If some computational resources are available for finding a suitable filter method, we recommend trying *permutation*, *impurity*, *sym.uncert*, *limma*, and *JMIM*. If the resources allow trying all filter methods, we recommend doing this, because only this way the very best filter method for a new data set can be found. Searching for the best filter method can be done by considering it as a tuning parameter, just like we did with respect to the classification method in our experiments.

This paper serves as a reference to people who would like to conduct feature filtering. It can help them choose appropriate filters according to their application scenario, computational resources, and so on. In the future, we will use the results of our analyses to determine the search space for good models. Our analyses could be extended by additionally considering other feature selection methods apart from pure filter methods, which was out of scope for this analysis. Also, in this paper, we performed tuning with respect to predictive performance and then analyzed not only the predictive performance but also the run time and the stability. In future analyses, one could perform multi-criteria tuning with respect to all performance criteria at the drawback of a much more complicated aggregation of the results. Another interesting aspect of research would be discovering data set characteristics that indicate which filter and classification methods perform best. If such characteristics were available, much run time could be saved in the tuning process. Ideally,

the characteristics should be cheap to compute in order to obtain a benefit in run time over trying different methods. In the field of optimization, exploratory landscape analysis (Kerschke and Trautmann, 2019) is an approach for defining characteristics of objective functions in order to automatically select the best optimization algorithm. In future research, this approach could be transferred to model selection.

Acknowledgments

This work was supported by Deutsche Forschungsgemeinschaft (DFG), Projects RA 870/7-1 and BI 1902/1-1, and Collaborative Research Center SFB 876, A3.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.csda.2019.106839.

References

Aphinyanaphongs, Y., Fu, L.D., Li, Z., Peskin, E.R., Efstathiadis, E., Aliferis, C.F., Statnikov, A., 2014. A comprehensive empirical comparison of modern supervised classification and feature selection methods for text categorization. J. Assoc. Inf. Sci. Technol. 65 (10), 1964–1987.
Biau, G., Cadre, B., Rouvière, L., 2019. Accelerated gradient boosting. Mach. Learn. 108 (6), 971–992.
Bischl, B., Lang, M., Kotthoff, L., Schiffner, J., Richter, J., Studerus, E., Casalicchio, G., Jones, Z.M., 2016. mlr: Machine learning in R. J. Mach. Learn. Res. 17 (170), 1–5.
Bischl, B., Mersmann, O., Trautmann, H., Weihs, C., 2012. Resampling methods for meta-model validation with recommendations for evolutionary computation. Evol. Comput. 20 (2), 249–275.
Bolón-Canedo, V., Sánchez-Maroño, N., Alonso-Betanzos, A., 2013. A review of feature selection methods on synthetic data. Knowl. Inf. Syst. 34 (3), 483–519.
Bolón-Canedo, V., Sánchez-Marono, N., Alonso-Betanzos, A., Benítez, J.M., Herrera, F., 2014. A review of microarray datasets and applied feature selection methods. Inform. Sci. 282, 111–135.
Bommert, A., Rahnenführer, J., Lang, M., 2017. A multicriteria approach to find predictive and sparse models with stable feature selection for high-dimensional data. Comput. Math. Methods Med. 2017.
Breiman, L., 2001. Random forests. Mach. Learn. 45 (1), 5–32.
Breiman, L., Friedman, J., Stone, C.J., Olshen, R., 1984. Classification and Regression Trees. CRC Press, Boca Raton, FL, USA.
Brezočnik, L., Fister, I., Podgorelec, V., 2018. Swarm intelligence algorithms for feature selection: A review. Appl. Sci. 8 (9).
Brown, G., Pocock, A., Zhao, M.-J., Luján, M., 2012. Conditional likelihood maximisation: A unifying framework for information theoretic feature selection. J. Mach. Learn. Res. 13, 27–66.
Cai, J., Luo, J., Wang, S., Yang, S., 2018. Feature selection in machine learning: A new perspective. Neurocomputing 300, 70–79.
Casalicchio, G., Bossek, J., Lang, M., Kirchhoff, D., Kerschke, P., Hofner, B., Seibold, H., Vanschoren, J., Bischl, B., 2017. OpenML: An R package to connect to the machine learning platform OpenML. Comput. Stat. 1–15.
Chandrashekar, G., Sahin, F., 2014. A survey on feature selection methods. Comput. Electr. Eng. 40 (1), 16–28.
Darshan, S.S., Jaidhar, C., 2018. Performance evaluation of filter-based feature selection techniques in classifying portable executable files. Procedia Comput. Sci. 125, 346–356.
Dash, M., Liu, H., 1997. Feature selection for classification. Intell. Data Anal. 1, 131–156.
Fayyad, U., Irani, K., 1993. Multi-Interval Discretization of Continuous-Valued Attributes for Classification Learning. Technical report, California Institute of Technology.
Fernández-Delgado, M., Cernadas, E., Barro, S., Amorim, D., 2014. Do we need hundreds of classifiers to solve real world classification problems? J. Mach. Learn. Res. 15, 3133–3181.
Fleuret, F., 2004. Fast binary feature selection with conditional mutual information. J. Mach. Learn. Res. 5, 1531–1555.
Forman, G., 2003. An extensive empirical study of feature selection metrics for text classification. J. Mach. Learn. Res. 3, 1289–1305.
Ghosh, M., Adhikary, S., Ghosh, K.K., Sardar, A., Begum, S., Sarkar, R., 2019. Genetic algorithm based cancerous gene identification from microarray data using ensemble of filter methods. Med. Biol. Eng. Comput. 57 (1), 159–176.
Guyon, I., Elisseeff, A., 2003. An introduction to variable and feature selection. J. Mach. Learn. Res. 3, 1157–1182.
Hall, M.A., 1999. Correlation-Based Feature Selection for Machine Learning (Ph.D. thesis). University of Waikato, Hamilton, New Zealand.
Hanley, J.A., McNeil, B.J., 1982. The meaning and use of the area under a receiver operating characteristic (ROC) curve. Radiology 143 (1), 29–36.
Hira, Z.M., Gillies, D.F., 2015. A review of feature selection and feature extraction methods applied on microarray data. Adv. Bioinform. 2015.
Hoque, N., Singh, M., Bhattacharyya, D.K., 2018. EFS-MI: An ensemble feature selection method for classification. Complex Intell. Syst. 4 (2), 105–118.
Huang, X., Zhang, L., Wang, B., Li, F., Zhang, Z., 2018. Feature clustering based support vector machine recursive feature elimination for gene selection. Appl. Intell. 48 (3), 594–607.
Inza, I., Larrañaga, P., Blanco, R., Cerrolaza, A.J., 2004. Filter versus wrapper gene selection approaches in DNA microarray domains. Artif. Intell. Med. 31 (2), 91–103.
Izenman, A.J., 2013. Modern Multivariate Statistical Techniques: Regression, Classification, and Manifold Learning, second ed. Springer, New York, USA.
Jović, A., Brkić, K., Bogunović, N., 2015. A review of feature selection methods with applications. In: 38th International Convention on Information and Communication Technology, Electronics and Microelectronics, pp. 1200–1205.
Kalousis, A., Prados, J., Hilario, M., 2007. Stability of feature selection algorithms: A study on high-dimensional spaces. Knowl. Inf. Syst. 12 (1), 95–116.
Karatzoglou, A., Smola, A., Hornik, K., Zeileis, A., 2004. kernlab – an S4 package for kernel methods in R. J. Stat. Softw. 11 (9), 1–20.
Ke, W., Wu, C., Wu, Y., Xiong, N.N., 2018. A new filter feature selection based on criteria fusion for gene microarray data. IEEE Access 6, 61065–61076.
Kerschke, P., Trautmann, H., 2019. Automated algorithm selection on continuous black-box problems by combining exploratory landscape analysis and machine learning. Evol. Comput. 27 (1), 99–127.
Kittler, J., 1978. Feature set search algorithms. In: Pattern Recognition and Signal Processing. Sijthoff and Noordhoff, Alphen aan den Rijn, Netherlands, pp. 41–60.
Kohavi, R., John, G.H., 1997. Wrappers for feature subset selection. Artificial Intelligence 97 (1–2), 273–324.
Kruskal, W.H., Wallis, W.A., 1952. Use of ranks in one-criterion variance analysis. J. Amer. Statist. Assoc. 47 (260), 583–621.
Kursa, M.B., 2018. praznik: Collection of information-based feature selection filters. https://CRAN.R-project.org/package=praznik.

Lang, M., Bischl, B., Surmann, D., 2017. batchtools: Tools for R to work on batch systems. J. Open Source Softw. 2 (10).
Larose, D.T., Larose, C.D., 2014. Discovering Knowledge in Data, second ed. John Wiley & Sons, Inc., Hoboken, NJ, USA.
Lazar, C., Taminau, J., Meganck, S., Steenhoff, D., Coletta, A., Molter, C., de Schaetzen, V., Duque, R., Bersini, H., Nowe, A., 2012. A survey on filter techniques for feature selection in gene eexpression microarray analysis. IEEE/ACM Trans. Comput. Biol. Bioinform. 9 (4), 1106–1119.
Li, J., Cheng, K., Wang, S., Morstatter, F., Trevino, R.P., Tang, J., Liu, H., 2018. Feature selection: A data perspective. ACM Comput. Surv. 50 (6).
Liu, Y., 2004. A comparative study on feature selection methods for drug discovery. J. Chem. Inf. Comput. Sci. 44 (5), 1823–1828.
Liu, H., Li, J., Wong, L., 2002. A comparative study on feature selection and classification methods using gene expression profiles and proteomic patterns. Genome Inform. 13, 51–60.
Liu, H., Yu, L., 2005. Toward integrating feature selection algorithms for classification and clustering. IEEE Trans. Knowl. Data Eng. 17 (4), 491–502.
Meyer, P.E., Schretter, C., Bontempi, G., 2008. Information-theoretic feature selection in microarray data using variable complementarity. IEEE J. Sel. Top. Sign. Proces. 2 (3), 261–274.
Mohtashami, M., Eftekhari, M., 2019. A hybrid filter-based feature selection method via hesitant fuzzy and rough sets concepts. Iran. J. Fuzzy Syst. 16 (2), 165–182.
Nogueira, S., Brown, G., 2016. Measuring the stability of feature selection. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases. pp. 442–457.
Peng, H., Long, F., Ding, C., 2005. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. IEEE Trans. Pattern Anal. Mach. Intell. 27 (8), 1226–1238.
R Core Team, 2017. R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria.
Ramey, J.A., 2016. datamicroarray: Collection of data sets for classification. https://github.com/ramhiser/datamicroarray.
Rasch, D., Kubinger, K.D., Yanagida, T., 2011. Statistics in Psychology using R and SPSS. John Wiley & Sons, Inc., Hoboken, NJ, USA.
Ritchie, M.E., Phipson, B., Wu, D., Hu, Y., Law, C.W., Shi, W., Smyth, G.K., 2015. limma powers differential expression analyses for RNA-sequencing and microarray studies. Nucleic Acids Res. 43 (7), e47.
Romanski, P., Kotthoff, L., 2016. Fselector: Selecting attributes. https://CRAN.R-project.org/package=FSelector.
Saeys, Y., Inza, I., Larrañaga, P., 2007. A review of feature selection techniques in bioinformatics. Bioinformatics 23 (19), 2507–2517.
Sammut, C., Webb, G.I., 2011. Encyclopedia of Machine Learning. Springer, New York, USA.
Sánchez-Maroño, N., Alonso-Betanzos, A., Tombilla-Sanromán, M., 2007. Filter methods for feature selection – A comparative study. In: International Conference on Intelligent Data Engineering and Automated Learning. pp. 178–187.
Schliep, K., Hechenbichler, K., 2016. kknn: Weighted k-nearest neighbors. https://CRAN.R-project.org/package=kknn.
Simon, N., Friedman, J., Hastie, T., Tibshirani, R., 2011. Regularization paths for cox's proportional hazards model via coordinate descent. J. Stat. Softw. 39 (5), 1–13.
Smyth, G.K., 2004. Linear models and empirical Bayes methods for assessing differential expression in microarray experiments. Stat. Appl. Genet. Mol. Biol. 3 (1).
Strobl, C., Boulesteix, A.-L., Kneib, T., Augustin, T., Zeileis, A., 2008. Conditional variable importance for random forests. BMC Bioinformatics 9 (307).
Tang, J., Alelyani, S., Liu, H., 2014. Feature selection for classification: A review. In: Data Classification: Algorithms and Applications. CRC Press, Boca Raton, FL, USA, pp. 37–64.
Therneau, T., Atkinson, B., Ripley, B., 2017. rpart: Recursive partitioning and regression trees. https://CRAN.R-project.org/package=rpart.
Tibshirani, R., 1996. Regression shrinkage and selection via the lasso. J. R. Stat. Soc. Ser. B Stat. Methodol. 58 (1), 267–288.
Tibshirani, R., Chu, G., Narasimhan, B., Li, J., 2011. samr: SAM: Significance analysis of microarrays. https://CRAN.R-project.org/package=samr.
Tusher, V.G., Tibshirani, R., Chu, G., 2001. Significance analysis of microarrays applied to the ionizing radiation response. Proc. Natl. Acad. Sci. USA 98 (9), 5116–5121.
Vanschoren, J., Van Rijn, J.N., Bischl, B., Torgo, L., 2013. OpenML: Networked science in machine learning. ACM SIGKDD Explor. Newsl. 15 (2), 49–60.
Venkatesh, B., Anuradha, J., 2019. A review of feature selection and its methods. Cybern. Inf. Technol. 19 (1), 3–26.
Wah, Y.B., Ibrahim, N., Hamid, H.A., Abdul-Rahman, S., Fong, S., 2018. Feature selection methods: case of filter and wrapper approaches for maximising classification accuracy. Pertanika J. Sci. Technol. 26 (1), 329–340.
Wright, M.N., Ziegler, A., 2017. ranger: A fast implementation of random forests for high dimensional data in C++ and R. J. Stat. Softw. 77 (1), 1–17.
Xue, B., Zhang, M., Browne, W.N., 2015. A comprehensive comparison on evolutionary feature selection approaches to classification. Int. J. Comput. Intell. Appl. 14 (2).
Xue, B., Zhang, M., Browne, W.N., Yao, X., 2016. A survey on evolutionary computation approaches to feature selection. IEEE Trans. Evol. Comput. 20 (4), 606–626.
Yang, J., Honavar, V., 1998. Feature subset selection using a genetic algorithm. In: Feature Extraction, Construction and Selection: A Data Mining Perspective. Springer, New York, USA, pp. 117–136.
Yu, L., Liu, H., 2004. Efficient feature selection via analysis of relevance and redundancy. J. Mach. Learn. Res. 5, 1205–1224.
Zawadzki, Z., Kosinski, M., 2017. FSelectorRcpp: 'Rcpp' implementation of 'FSelector' entropy-based feature selection algorithms with a sparse matrix support. https://CRAN.R-project.org/package=FSelectorRcpp.
Zhu, Z., Ong, Y.-S., Dash, M., 2007. Wrapper-filter feature selection algorithm using a memetic framework. IEEE Trans. Syst. Man Cybern. B 37 (1), 70–76.

Appendix I

Extra Methodology

I.1 Gaussian Process from the view of Bayesian Linear Regression

Consider a space of dimension p spanned by the set of basis function $\phi^T(x) = [\phi_1(x), \ldots, \phi_p(x)]$ with coefficients $w^T = [w_1, \ldots, w_p]$ in the way $y(x|w) = w^T\phi(x) + \sigma_y\epsilon$, where $\epsilon \sim \mathcal{N}(0, 1)$ and σ_y does not depend on x.

Consider data $\{x_i, y_i\}$ with n observations, the likelihood function for w is

$$l(y; w) = p(y_{1:n}|w, x_{1:n}) = \prod_i p(y_i|w, x_i) = \prod_i \mathcal{N}(w^T\phi(x_i), \sigma_y^2) \tag{I.1}$$

Define design matrix $\Phi = \begin{pmatrix} \phi_1(x_1) & \phi_2(x_1) & \ldots & \phi_p(x_1) \\ \vdots & \vdots & \vdots & \vdots \\ \phi_1(x_n) & \phi_2(x_n) & \ldots & \phi_p(x_n) \end{pmatrix}^T$ and $\lambda_y = \frac{1}{\sigma_y^2}$, then

$$\begin{aligned} l(y; w) =& p(y_{1:n}|w, x_{1:n}) = \mathcal{N}\left(w^T\Phi, D_n(\sigma_y^2)\right) && \text{(I.2)} \\ =& \frac{1}{(2\pi)^{n/2}|D_n(\sigma_y^2)|^{1/2}} \exp^{-\frac{1}{2}(y^T - w^T\Phi)D_n^{-1}(\sigma_y^2)(y^T - w^T\Phi)^T} && \text{(I.3)} \\ =& \frac{1}{(2\pi)^{n/2}D_n(\sigma_y^2)|^{1/2}} \exp^{-\frac{1}{2}\lambda_y y^T y + \frac{1}{2}\lambda_y y^T\Phi^T w + \frac{1}{2}\lambda_y w^T\Phi y - \frac{1}{2}\lambda_y w^T\Phi\Phi^T w} && \text{(I.4)} \end{aligned}$$

, where $D_n(\sigma_y^2)$ represent diagonal matrix with element σ_y^2 of size n.

Define a prior distribution over the coefficients as $p(w)$, the posterior distribution of coefficients w is

$$p(w|y_{1:n}, x_{1:n}) = \frac{p(w)p(y_{1:n}|w, x_{1:n})}{\int p(w)p(y_{1:n}|w, x_{1:n})dw} = \frac{p(w)p(y_{1:n}|w, x_{1:n})}{p(y_{1:n}|x_{1:n})} \propto \exp^{s(w)} \tag{I.5}$$

where $s(w)$ on the shoulder of the exponential can be computed by the complete the square

trick by taking $p(w) \propto e^{-\frac{1}{2}w^T\Sigma_p^{-1}w}$ in Equation I.6.

$$s(w) = -\frac{1}{2}\lambda_y y^T y + \lambda_y y^T \Phi^T w - \frac{1}{2}\lambda_y w^T \Phi\Phi^T w - \frac{1}{2}w^T\Sigma_p^{-1}w \tag{I.6}$$

$$= -\frac{1}{2}w^T(\lambda_y\Phi\Phi^T + \Sigma_p^{-1})w + \lambda_y y^T\Phi^T w - \frac{1}{2}\lambda_y y^T y \tag{I.7}$$

Write $s(w)$ as:

$$s(w) = -\frac{1}{2}(w^T - u^T)\Lambda(w-u) \tag{I.8}$$

$$= -\frac{1}{2}w^T\Lambda w + w^T\Lambda u - \frac{1}{2}u^T u \tag{I.9}$$

Matching the second order term of equation (I.7) and equation (I.9), we have

$$\Lambda = \lambda_y\Phi\Phi^T + \Sigma_p^{-1} \tag{I.10}$$

Matching the first order term of equation (I.7) and equation (I.9), we have $\lambda_y y^T\Phi^T w = w^T\Lambda u = u^T\Lambda^T w$, so

$$u^T = \lambda_y y^T\Phi^T(\Lambda^T)^{-1} \tag{I.11}$$

and

$$u = (u^T)^T = \lambda_y\Lambda^{-1}\Phi y \tag{I.12}$$

.

So the final posterior distribution of w is $p(w|y_{1:n}, x_{1:n}) = \mathcal{N}\left(w|\lambda_y\Lambda^{-1}\Phi y, (\lambda_y\Phi\Phi^T + \Sigma_p^{-1})^{-1}\right)$. Compared to the prior $\mathcal{N}(0, \Sigma_p)$, likelihood information is mixed in.

Predicative distribution Given test point $x^*_{1:n^*}$, we average the prediction of $y^*_{1:n^*}$ with the posterior distribution of w (learned from $x_{1:n}, y_{1:n}$):

$$p(y^*_{1:n^*}|x^*_{1:n^*}, x_{1:n}, y_{1:n}) = \int_w p(y^*_{1:n^*}, w|x^*_{1:n^*}, x_{1:n}, y_{1:n})dw \tag{I.13}$$

$$= \int_w p(y^*_{1:n^*}|w, x^*_{1:n})p(w|x_{1:n}, y_{1:n})dw \tag{I.14}$$

$$(\text{since } w \perp x^*_{1:n^*}|\{x_{1:n}, y_{1:n}\}, \text{ and } y^*_{1:n^*} \perp \{x_{1:n}, y_{1:n}\}|\{x^*_{1:n^*}, w\})$$

According to Bishop (2006) of equation (2.113) till (2.115), if

$$p(w|x_{1:n}, y_{1:n}) = \mathcal{N}(w|\mu, P^{-1}) \tag{I.15}$$

$$= \mathcal{N}\left(w|\lambda_y\Lambda^{-1}\Phi y,\ (\lambda_y\Phi\Phi^T + \Sigma_p^{-1})^{-1}\right) \tag{I.16}$$

and

$$p(y^*_{1:n^*}|w; x^*_{1:n^*}) = \mathcal{N}(y^*_{1:n^*}|Aw + b,\ L^{-1}) \text{ with } b = 0 \tag{I.17}$$

$$= \mathcal{N}((\phi^*)^T w,\ \lambda_y^{-1}I_n) \tag{I.18}$$

then

$$p(y^*_{1:n^*}|x_{1:n}, y_{1:n}; x^*_{1:n^*}) = \int_w p(y^*_{1:n^*}|w, x^*_{1:n^*})p(w|x_{1:n}, y_{1:n})dw \tag{I.19}$$

$$= \int_w \mathcal{N}(y^*|Aw + b, L^{-1})\mathcal{N}(w|\mu, P^{-1})dw \tag{I.20}$$

$$= \mathcal{N}(y|A\mu + b,\ L^{-1} + AP^{-1}A^T) \tag{I.21}$$

$$= \mathcal{N}(\phi_*^T\lambda_y\Lambda^{-1}\Phi y,\ \lambda_y^{-1}I_n + \phi_*^T\Lambda^{-1}\phi_*) \tag{I.22}$$

, where $P = \Lambda = \lambda_y\Phi\Phi^T + \Sigma_p^{-1}$ is defined by Equation (I.16) and (I.10).

As shown in Equation (I.22), the variance of the predictive distribution is partly a quadratic form of the test input point with the scale decided by $\Lambda = \lambda_y\Phi\Phi^T + \Sigma_p^{-1}$.

Avoid matrix inversion on the high dimensional space To avoid computing matrix inversion on the $\Lambda = \lambda_y\Phi\Phi^T + \Sigma_p^{-1}$ in Equation (I.22), which is high dimensional in the feature space, define

$$K = \Phi^T\Sigma_p\Phi =< \Sigma_p^{\frac{1}{2}}\Phi, \Sigma_p^{\frac{1}{2}}\Phi > \tag{I.23}$$

, then

$$\sigma_y^{-2}\Phi(K + \sigma_y^2 I_n) = \sigma_y^{-2}\Phi(\Phi^T\Sigma_p\Phi + \sigma_y^2 I_n) \tag{I.24}$$

$$= \sigma_y^{-2}\Phi(\Phi^T\Sigma_p\Phi + \sigma_y^2 I_n\Sigma_p^{-1}\Sigma_p) \tag{I.25}$$

$$= \sigma_y^{-2}\Phi\Phi^T\Sigma_p\Phi + \Phi\Sigma_p^{-1}\Sigma_p \tag{I.26}$$

$$= \sigma_y^{-2}\Phi\Phi^T\Sigma_p\Phi + \Sigma_p^{-1}\Sigma_p\Phi \tag{I.27}$$

$$= (\sigma_y^{-2}\Phi\Phi^T + \Sigma_p^{-1})\Sigma_p\Phi \tag{I.28}$$

$$= \Lambda\Sigma_p\Phi \tag{I.29}$$

Left multiply each side of equation (I.29) by Λ^{-1} and right multiply both sides by $(K + \sigma_y^2 I)^{-1}$, we have

$$\Lambda^{-1}\sigma_y^{-2}\Phi = \Sigma_p\Phi(K + \sigma_y^2 I)^{-1} \tag{I.30}$$

which shows the inversion of the high dimensional precision matrix Λ could be replaced by the inversion of the low dimensional kernel matrix $K_{n\times n}$. So $\phi_*^T\lambda_y\Lambda^{-1}\Phi y$ in Equation (I.22) becomes $\phi_*^T\Sigma_p\Phi(K + \sigma_y^2 I)^{-1}y$.

General definition of kernels Let $f(x) = w^T\phi(x)$,

$$\mathbb{E}[f(x)] = \mathbb{E}[w^T\phi(x)] = \mathbb{E}[\phi(x)^T w] = 0 \tag{I.31}$$

$$\mathbb{E}\ [f(x) - \mathbb{E}[f(x)]]\left[f(x^{'}) - \mathbb{E}[f(x^{'})]\right] = \mathbb{E}[\phi(x)^T w w^T\phi(x)] \tag{I.32}$$

$$= \phi(x)^T E[ww^T]\phi(x) = \phi(x)^T\Sigma_p\phi(x) = k(x, x^{'}) \tag{I.33}$$

Equation (I.31) and (I.33) defines the mean and covariance of any pair of observations $f(x), f(x^{'})$ which can be extended any number of observations, then, $f(x) \sim \mathcal{GP}\left(m(x), k(x, x^{'})\right)$.

Extending to non-linear Kernel Function The covariance function or kernel for Bayesian Linear Regression is $K = \Phi^T \Sigma_p \Phi$ defined in Equation (I.23. Extending to arbitrary kernel function like $k(x, x^{'}) = \exp(-\frac{1}{2}|x - x^{'}|^2)$ enables modeling more complicated interactions.

Divide the data into observed pairs $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ and test input $\{x^*, y^*\}$ with n_* pairs, we are interested in the posterior distribution

$$p(y^*|x_1, \ldots, x_n, y_1, \ldots, y_n, x^*) \tag{I.34}$$

For Gaussian process, the response jointly follows Gaussian distribution with mean function and covariance function dependent on the covariate.

I.2 Basic Information Theory

In this section, we use H to represent entropy and I to represent information. Capital letter (e.g.X) represents a Random Variable, the corresponding lower case represents the realization of the corresponding Random Variable. p and q are used to represent probabilistic distribution functions. For simplicity, $H(X)$, $H(p)$, where $p(x)$ is the p.d.f of X, both represent Entropy.

The Information of a random event $X = x$ can be defined as $I(X = x) = -\log p(x)$. Entropy $H(X) = \int -p(x) \log p(x) dx = E_{x \sim p(x)}\left[I(x)\right]$, where $I(x) = -\log p(x)$. Entropy is always non-negative and measure the amount of uncertainty of a Random Variable, following distribution $p(x)$. Entropy as expected information $H(X) = -\sum p(x) \log p(x)$ can be interpreted as the expected number of binary search needed to locate x (number of bits).

$H(Y|X = x) = \sum_{y \in \mathcal{Y}}[-p(y|x) \log(p(y|x))]$ is conditional expectation of information, integrating over $x \in \mathcal{X}$, results in Conditional Entropy defined as

$$\begin{aligned} H(Y|X) &= E_{x \sim p(x)}\left[H(Y|X = x)\right] && \text{(I.35)} \\ &= E_{x \sim p(x)}\left[E_{y \sim p(y|x)} - \log p(y|x)\right] && \text{(I.36)} \\ &= \sum_x \sum_y -p(y|x) \log p(y|x) && \text{(I.37)} \end{aligned}$$

Conditional Entropy $H(Y \mid X)$ measures the reduced uncertainty of Y given observation

of X w.r.t. Joint Entropy $H(X,Y)$, defined as

$$H(X,Y) = \int\int -p(x,y)\log p(x,y)dxdy \tag{I.38}$$
$$= \int\int -p(x)p(y \mid x)\log[p(x)p(y \mid x)]dxdy \tag{I.39}$$
$$= \int\int -p(x)p(y \mid x)\log p(x)dxdy + \int\int -p(x)p(y \mid x)\log p(y \mid x)dxdy \tag{I.40}$$
$$= H(X) + H(Y \mid X) \tag{I.41}$$

which is the chain rule of entropy: The joint entropy of X and Y is the entropy of X plus the conditional entropy of $Y|X$.

Cross Entropy $H_p(q)$ is defined between two distributions p and q as

$$H_p(q) = \int -p(x)\log q(x)dx = E_p\left[I_q\right] \tag{I.42}$$

$H_p(q)$ is the expectation of the information of q with respect to p.

Mutual Information is defined as

$$\begin{aligned} I(X;Y) &= \int\int p(x,y)\log\frac{p(x,y)}{p(x)p(y)}dxdy \\ &= \int\int p(x)p(y \mid x)\log\frac{p(x)p(y \mid x)}{p(x)p(y)}dxdy = \int\int p(x)p(y \mid x)\log\frac{p(y \mid x)}{p(y)}dxdy \\ &= \int\int p(x)p(y \mid x)\log p(y \mid x)dxdy - \int\int p(x)p(y \mid x)\log p(y)dxdy \\ &= -H(Y \mid X) + H(Y) \geq 0 \end{aligned} \tag{I.43}$$

. KL Divergence measures the uncertainty change or average information gain, with respect to $H(p)$

$$D_{KL}(p,q) = \int p(x)\log\frac{p(x)}{q(x)}dx = H_p(q) - H(p) \geq 0 \tag{I.44}$$

Relationship between KL Divergence and Mutual Information is

$$I(X;Y) = \int\int p(x,y)\log\left[\frac{p(x,y)}{p(x)p(y)}\right]dxdy = \int\int p(y)p(x|y)\log\left[\frac{p(x,y)}{p(x)p(y)}\right]dxdy$$
$$= \int\int p(y)p(x|y)\log\left[\frac{p(x \mid y)}{p(x)}\right]dxdy = E_Y\left[p(x|y)\log\left[\frac{p(x \mid y)}{p(x)}\right]\right] \tag{I.45}$$
$$= E_Y\left[D_{KL}(p(x|y),p(x))\right] \tag{I.46}$$
$$= E_X\left[D_{KL}(p(y|x),p(y))\right] \tag{I.47}$$

Equation (I.46) is the average KL Divergence between $p(x \mid y)$, the conditional probability of x, upon observation of variable y, and the original $p(x)$. $E_X\left[D_{KL}(p(y|x),p(y))\right]$ is the information gain on Y once feature X is observed, which reduces the uncertainty about Y.

Chain Rule of Mutual Information

Take X, Z as a concatenated variable, one can calculate the mutual information between X, Z and Y as

$$\begin{aligned}
&I(X,Z;Y)\\
=&\sum_{x,y,z} p_{x,z,y}(x,z,y)\log\frac{p_{x,z,y}(x,z,y)}{p_{x,z}(x,z)p_y(y)} = \sum_{x,y,z} p_{x,y,z}(x,y,z)\log\frac{p_z(z)p_{x,y|z}(x,y|z)}{p_{x|z}(x|z)p_z(z)p_y(y)}\\
=&\sum_{x,y,z} p_{x,y,z}(x,y,z)\log\frac{p_{x,y|z}(x,y|z)}{p_{x|z}(x|z)p_y(y)} = \sum_{x,y,z} p_{x,y,z}(x,y,z)\log\frac{p_{x,y|z}(x,y|z)}{p_{x|z}(x|z)p_{y|z}(y|z)}\frac{p_{y|z}(y|z)}{p_y(y)}\\
=&I(X,Y|Z)+\sum_{x,y,z} p(x,y,z)\log\frac{p_{y|z}(y|z)}{p_y(y)} = I(X,Y|Z)+I(Z,Y)
\end{aligned} \tag{I.48}$$

where the mutual information $I(Z,Y)=\sum_z\sum_y p_{z,y}(z,y)\log\frac{p_{z,y}(z,y)}{p_z(z)p_y(y)}$ and the conditional mutual information is

$$I(X,Y|Z)=\sum_z p(z)\sum_{x,y} p_{x,y|z}(x,y|z)\log\frac{p_{x,y|z}(x,y|z)}{p_x(x|z)p_y(y|z)} = \sum_{x,y,z} p(x,y,z)\log\frac{p_{x,y|z}(x,y|z)}{p_x(x|z)p_y(y|z)} \tag{I.49}$$

The chain rule of mutual information $I(X,Z;Y)=I(X,Y|Z)+I(Z,Y)$ where Z is the d-separation node. The conditional mutual information and $I(z,y)$ are trade offs which sum up to the total mutual information.

Due to the symmetry $I(X,Y|Z)=I(X,Z;Y)-I(Y,Z)=I(Y,Z;X)-I(X,Z)$, there is $I(X,Y|Z)-I(X,Y)=I(Y,Z;X)-I(X,Z)-I(X,Y)$, which leads to the identity

$$I(X,Y|Z)-I(X,Y)=I(X,Z|Y)-I(X,Z) \tag{I.50}$$