Data Analytics Seminar III
Hyperparameters Optimization

Salvador Rocha, Eduardo
ID 277584

Stiftung Universität Hildesheim
Information Systems and Machine Learning Lab (ISMLL)
International Master's Program in Data Analytics
M. Eng. Hadi Samer Jomaa

Abstract

In many applications and problems, it has been observed that an optimal solution can be achieved by just configuring better existing machine learning techniques rather than inventing new ones, and to get them to work well, hyper-parameter tuning is a crucial step. There is also a big number of machine learning practitioners for whom, with a lot of menial work involved, grid search and manual search are the de facto strategies to use for hyper-parameter optimization, only accelerated by domain knowledge. Automated hyper-parameter tuning can reduce the human effort necessary for applying an optimized machine learning algorithm. This paper is thus, an overview of the most prominent methods for hyper-parameter optimization (HPO), knowledge transference (meta-learning) and Neural Architecture Search.

1 Automated hyper-parameter optimization (HPO)

1.1 Introduction

Hyper-parameters (HP) are inherent features of every machine learning algorithm, and their optimization is typically carried out by hand, progressively refining a grid over the hyper-parameters space. Hyper-parameter optimization (HPO) aims to systematically find the best HP values in order to achieve a good performance, specially in cases where the algorithm at hand has a large number of HP to tune, e.g. recent deep neural networks which crucially depend on a wide range of choices like network's architecture, regularization, weight's initialization, optimizer, etc (BAKER *et al.*, 2016).

The problem of HPO date as back to the 1990s and from that early it has been established that different hyperparameter configurations tend to work best for different data sets, and as such HPO has several important use cases. In the simpler scenario HPO can reduce the human effort necessary for applying machine learning, but it should be noted that it's essential motivation is to improve the performance of the an algorithm. In several studies, HPO has contributed to obtain new state-of-the-art performances for well-known machine learning benchmarks (BERGSTRA; BENGIO, 2012) and since different methods can only be compared fairly if they all receive the same level of tuning for the problem at hand (BARDENET *et al.*, 2013) it also has improved the reproducibility and fairness of scientific studies.

In this line, companies also have increased the usage of machine learning, be it in company-internal tools, as part of machine learning cloud services (AMAZON, 2018)m (LI, 2018), or as a service by itself (SIGOPT, 2018), demonstrating in this way the commercial appeal that HPO can offer. Furthermore HPO plays an ever larger role as it can be used to adapt general-purpose pipelines to the companies' specific application domains.

On the other hand, HPO faces several challenges that make it a hard problem in practice as function evaluations are usually extremely expensive for large models (as in deep learning), complex pipelines, or large data sets, while in most cases, the HP space is often complex (comprising a mix of continuous, categorical and conditional features) and high-dimensional. Furthermore, it is not always clear which of an algorithm's HP need to be optimized, and in which ranges (ZOPH; LE, 2017).

Usually, researchers don't have access to a gradient of the loss function with respect to the hyperparameters. Furthermore, other properties of the target function often used in classical optimization do not typically apply, such as convexity and smoothness. One cannot directly optimize for generalization performance as training datasets are of limited size (HUTTER *et al.*, 2015).

1.2 HPO Problem Definition

Let A denote a machine learning algorithm with n hyperparameters θ. Yhe domain of the $n-th$ hyperparameter is denoted by Θ_n and the overall hyperparameter configuration space as $\mathbf{\Theta} = \Theta_1 \times \Theta_2 \times ...\Theta_n$

. A vector of hyperparameters is denoted by $\theta \in \mathbf{\Theta}$, and A with its hyperparameters represented by θ is denoted by A_θ. The domain of a hyperparameter can be real-valued (e.g., learning rate), integer-valued (e.g., number of layers), binary (e.g., whether to use early stopping or not), or categorical (e.g., choice of optimizer). For integer and real-valued hyperparameters, the domains are mostly bounded for practical reasons (HUTTER *et al.*, 2018).

Given a data set D, the goal is to find

$$\theta^* = \underset{\theta \in \mathbf{\Theta}}{\operatorname{argmin}} \mathbb{E}_{(D_{train}, D_{valid}) \sim D} \mathbf{V}(\mathcal{L}, A_\theta, D_{train}, D_{valid}) \tag{1}$$

where $\mathbf{V}(\mathcal{L}, A_\theta, D_{train}, D_{valid})$ measures the loss of a model generated by algorithm A with hyperparameters θ on training data D_{train} and evaluated on validation data D_{valid}. In practice, we only have access to finite data $\mathbf{D} \sim D$ and thus need to approximate the expectation in Equation 1. Popular choices for the validation protocol of $\mathbf{V}$ are the holdout and cross-validation error for a user-given loss function.

In practice, the HPO problem can also be addressed as:

$$\theta^{(*)} \approx \underset{\theta \in \mathbf{\Theta}}{\operatorname{argmin}} \underset{x \in \mathcal{X}^{(valid)}}{\operatorname{mean}} \mathcal{L}(x, A_\theta(\mathcal{X}^{(train)})) \tag{2}$$

$$\equiv \underset{\theta \in \mathbf{\Theta}}{\operatorname{argmin}} \mathbf{\Psi}(\theta) \tag{3}$$

$$\approx \underset{\theta \in \{\theta^{(1)}, \ldots, \theta^{(s)}\}}{\operatorname{argmin}} \mathbf{\Psi}(\theta) \equiv \hat{\theta} \tag{4}$$

Equation 3 expresses the hyper-parameter optimization problem in terms of a hyper-parameter response function, Ψ. Hyper-parameter optimization is the minimization of $\Psi(\theta)$ over $\theta \in \Theta$. This function is sometimes called the response surface. Different data sets, tasks, and learning algorithm families give rise to different sets Θ and functions Ψ. Knowing in general very little about the response surface Ψ or the search space Θ, the dominant strategy for finding a good θ is to choose some number (S) of trial points $\{\theta^1, ..., \theta^S\}$, to evaluate $\Psi(\theta)$ for each one, and return the θ_i that worked the best as θ^*. This strategy is made explicit by Equation 4 (BERGSTRA; BENGIO, 2012).

1.3 HPO Methods

1.3.1 Grid Search

Grid search is the most basic HPO model-free method, also known as full factorial design [110]. In this approach, a finite set of values are defined for each HP and the Cartesian product of these sets is evaluated. The major downside of this method is that the required number of function evaluations grows exponentially with the dimensionality of the configuration space (a.k.a. the curse of dimensionality). As additional problem, increasing the resolution of discretization substantially increases the required number of function evaluations. Albeit it simplicity in implementation, trivial parallelization and reliable in low dimensional spaces (e.g., 1-d, 2-d), in (BERGSTRA; BENGIO, 2012) it is shown the theoretical and empirical inefficiencies of this method, mainly because functions Ψ of interest are more sensitive to changes in some dimensions than others.

1.3.2 Random Search

A simple alternative to grid search is random search [13]. As the name suggests, random search samples configurations at random until a certain threshold for the search is exhausted. The performance is specially superior than grid search when some HPs are much more important than others (a property that can be exploited to deliver faster solutions (van Rijn; Hutter, 2017)).

Random Search also serves as a useful baseline because it makes no assumptions on the algorithm being optimized, it's easily parallelizable (considering that workers do not need to communicate between each other and a failing worker do not harm the overall design), it accounts for flexible resource allocation (since an arbitrary number of random points can be added at any time to still yield a random search design; the equivalent does not hold for grid search).

Interleaving random search with more complex optimization strategies allows to guarantee a minimal rate of convergence towards the global minimum, and also adds exploration that can improve model-based search [59]. Random search is also a useful method for initializing the search process, as it explores the entire configuration space and thus often finds settings with reasonable performance. However, one of it's major inconveniences is that it often takes far longer time than guided search methods to identify one of the best performing HP configurations (HUTTER *et al.*, 2018).

1.3.3 Bayesian Optimization and Surrogate Models

Bayesian optimization (BO) is a well known framework for the global optimization of expensive functions, and has seen success in HPO by achieving new state-of-the-art results in tuning a wide variety of deep neural networks (ZOPH; LE, 2017), (BAKER *et al.*, 2016). In a nutshell BO is an iterative algorithm comprised of two main components: a probabilistic surrogate model and an acquisition function that decide which point to evaluate next. At each iteration, the surrogate model tries to emulate the response surface Ψ by fitting all the sets of HP and its evaluations over the the target function made so far. Then the acquisition function make use of the predictive distribution of the probabilistic model to determine the adequacy of different candidate points, trading off between exploration and exploitation (BROCHU *et al.*, 2010). In this sense, the motivation behind the use of an acquisition function is that in general it's cheaper to compute and can therefore be thoroughly optimized compared to evaluating the expensive function.

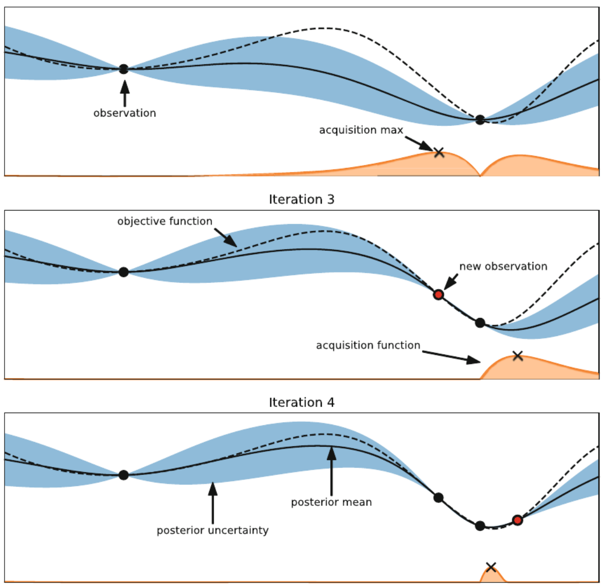

Figure 1: Illustration of Bayesian optimization on a 1-d function.

Although there are a broad range of acquisition functions to choose from, the expected improvement (EI) (Equation 5) is a common choice since it can be computed in closed from if the model prediction $\hat{y}$ at a HP configuration θ follows a normal distribution. In Equation 6, $\phi()$ and $\Phi()$ are the standard normal density and standard normal distribution funcion, and f_{min} is the best observed value so far:

$$\mathbb{E}[\mathbb{I}(\theta)] = \mathbb{E}[\max(f_{min} - \hat{y}, 0)] \tag{5}$$

$$\begin{aligned}\mathbb{E}[\mathbb{I}(\theta)] = (f_{min} - \mu(\theta))\Phi(\frac{f_{min} - \mu(\theta)}{\sigma}) \\ +\sigma\phi(\frac{f_{min} - \mu(\theta)}{\sigma})\end{aligned} \tag{6}$$

An illustration of a BO of a 1-dimensional toy function is presented in Figure 1 (reproduced from (HUTTER *et al.*, 2018)). The objective is to minimize the dashed line using a Gaussian process surrogate (predictions shown as black line, with blue tube representing the uncertainty) by maximizing the acquisition function represented by the lower orange curve. In the top frame, the acquisition value is low around observations, and the highest acquisition value is at a point where the predicted function value is low and the predictive uncertainty is relatively high. In the Middle frame, while there is still a lot of variance to the left of the new observation, the predicted mean to the right is much lower and the next observation is conducted there. At the last frame can be noted that although there is almost no uncertainty left around the location of the true maximum, the next evaluation is done there due to its expected improvement over the best point so far.

Traditionally, BO employs Gaussian processes (GP) to model the target function because of their expressiveness, smooth and well-calibrated uncertainty estimates and closed-form computability of the predictive distribution. A Gaussian process $G(m(\theta), k(\theta, \theta')$ is fully specified by a mean $m(\theta)$ and a covariance function $k(\theta, \theta')$, although the mean function is usually assumed to be constant in BO (RASMUSSEN; WILLIAMS, 2005). The performance of the GP is determined entirely by the covariance function and a common choice is the Mátern 5/2 kernel although there is an extensive list of kernel options. The two biggest downsides of standard GP is that they scale cubically in the number of data points, and have poor scalability to high dimensions and since some other machine learning models are more scalable and flexible than Gaussian processes, there is also a large body of research on adapting these models to Bayesian optimization.

For example, (deep) neural networks are a very flexible and scalable models. The simplest way to apply them to Bayesian optimization is as a feature extractor to preprocess inputs and then use the outputs of the final hidden layer as basis functions for Bayesian linear regression. In this sense, Neural networks tend to be faster than Gaussian processes for Bayesian optimization after 250 function evaluations, which also allows for large-scale parallelism (BAKER *et al.*, 2016).

Another alternative model for Bayesian optimization are random forests [59]. While GPs perform better than random forests on small, numerical configuration spaces [29], random forests natively handle larger, categorical and conditional configuration spaces where standard GPs do not work well [29, 70, 90]. Furthermore, the computational complexity of random forests scales far better to many data points: while the computational complexity of fitting and predicting variances with GPs for n data points scales as $O(n^3)$ and $O(n^2)$, respectively, for random forests, the scaling in n is only $O(nlog(n))$ and $O(log(n))$, respectively.

Instead of modeling the probability $p(y|\theta)$ of observations y given the configurations θ, the Tree Parzen Estimator (TPE [12, 14]) models density functions $p(\theta|y < \alpha)$ and $p(\theta|y \geq \alpha)$. Given a percentile α (usually set to 15%), the observations are divided in good observations and bad observations and simple $p(\theta|y < \alpha)$ 1-d Parzen windows are used to model the two distributions. The ratio $\frac{p(\theta|y<\alpha)}{p(\theta|y\geq\alpha)}$ related to the expected improvement acquisition function and is used to propose new hyperparameter configurations. TPE uses a tree of Parzen estimators for conditional hyperparameters and demonstrated good performance on such structured HPO tasks [12, 14, 29, 33, 143, 149, 160], is conceptually simple, and parallelizes naturally [91].

1.3.4 Multi-Fidelity Methods

As mentioned in the previous section, an the increasing sizes of the data sets along the development of further complex models are a major hurdle in HPO since they make blackbox performance evaluation more expensive, making even training a single HP configuration θ on a large data set easily exceed several hours and take up to several days [85]. A frequent approach to hasten manual tuning is therefore to execute an algorithm/HP configuration on a small subset of the data, by training it only for a few iterations, by running it on a subset of HP, or by only using one or a few of the cross-validation folds [120]. These kind of methods are usually denominated as multi-fidelity approximations, which in turn cast manual heuristics into formal algorithms using a so-called low fidelity approximations of the actual loss function to minimize. These approximations introduce a trade-off between optimization performance and runtime, but in practice, the obtained speedups often counterbalance the approximation error (FALKNER *et al.*, 2018).

A straightforward approximation is to just model and extrapolate the algorithm's learning curves within the initial steps of the training of a given hyperparameter configuration in order to decide whether to add further resources or stop the training procedure in case the configuration is predicted to not reach the performance of the best model trained so far in the optimization process [reference].

The successive halving algorithm is another extremely simple, yet powerful, and therefore popular strategy for multi-fidelity algorithm selection introduced for HPO [69]: for a given initial budget, query all algorithms for that budget; then, remove the half that performed worst, double the budget and successively repeat until only a single algorithm is left. This process is illustrated in Figure 2 (reproduced from (HUTTER *et al.*, 2018)). Successive halving performs well both in terms of the number of required iterations and in the required computation time, and theoretically outperforms a uniform budget allocation strategy if the algorithms converge favorably.

Despite successive halving algorithm's efficiency it enclose the budget-vs-number of configurations trade off. Given a total budget, the number of configurations has to be established in advance as well as the amount of budget designated to it. In one hand, assigning a small budget can result in prematurely terminating good configurations while on the other, assigning a large a budget can result in running poor configurations for too long and thereby wasting resources.

HyperBand is a hedging strategy designed to combat this problem when selecting from randomly sampled configurations. It divides the total budget into several combinations of number of configurations vs. budget for each, to then call successive halving as a subroutine on each set of random configurations. Due to the hedging strategy which includes running some configurations only on the maximal budget, in the worst case, HyperBand takes at most a constant factor more time than vanilla random search on the maximal budget [90]. In practice, due to its use of cheap low-fidelity evaluations, HyperBand has been shown to improve over vanilla random search and blackbox Bayesian optimization for data subsets, feature subsets and iterative algorithms. As an upgraded version of Hyperband, BOHB combines Bayesian optimization and HyperBand to achieve the best of both worlds: strong anytime performance (quick improvements in

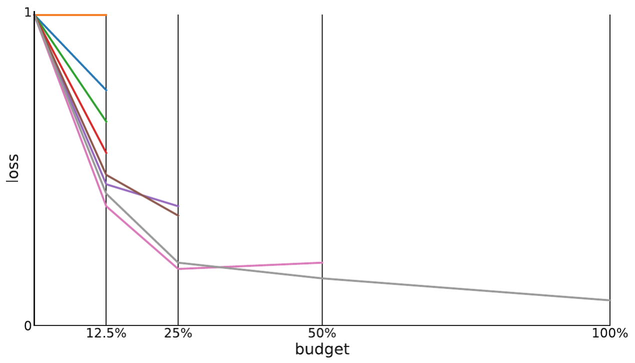

Figure 2: Illustration of successive halving for eight algorithms/configurations.

the beginning by using low fidelities in HyperBand) and strong final performance (good performance in the long run by replacing HyperBand's random search by Bayesian optimization). BOHB also uses parallel resources effectively and deals with problem domains ranging from a few to many dozen hyperparameters. BOHB's Bayesian optimization component resembles TPE [12], but differs by using multidimensional kernel density estimators. BOHB's first model is fitted on the lowest fidelity, and over time models trained on higher fidelities take over, while still using the lower fidelities in successive halving. Empirically, BOHB was shown to outperform several state-of-the-art HPO methods for tuning support vector machines, neural networks and reinforcement learning algorithms (FALKNER *et al.*, 2018) (HUTTER *et al.*, 2018).

2 Meta-Learning

2.1 Introduction

Following the natural way of human learning (i.e. by experience), many machine learning models are often build on past knowledge from previous tasks; it rarely starts from scratch. Meta-learning is then a machine learning area where learning from prior experience in a systematic, data-driven way is seek. Meta-data is thus defined as information that describe prior learning tasks (in most cases, *task* can be understood as a *datasets*, but the term *task* will be used for the rest of this writing for generalization purposes) and previously learned models, and comprises the exact algorithm configurations used to train the models, HP settings, pipeline compositions, network architectures, the resulting model evaluations (e.g. accuracy and training time), the learned model parameters (e.g. weights of a neural net), as well as measurable properties of the task itself (properties also known as meta-features). After gathering this meta-data, the challenge now is to learn how to extract and transfer knowledge that guides the search for optimal models for new tasks from this prior information.

Any type of learning based on prior experience with other tasks is considered to be part of meta-learning.

The more similar those previous tasks are, the more types of meta-data we can leverage, and defining task similarity will be a key overarching challenge, but it should be noted that when a new task represents completely unrelated phenomena, or random noise, leveraging prior experience will not be effective.

As remark, while ensemble learning (multiple models on the same task) and multi-task learning (multiple related tasks simultaneously), can often be meaningfully combined with meta-learning systems, they do not in themselves involve learning from prior experience on other tasks.

2.2 Problem Definition

Let define a prior task as $t_j \in T$, where T is the set of all known tasks, as well as a set of learning algorithms, fully defined by their configurations $\theta_i \in \Theta$; here Θ represents a discrete, continuous, or mixed configuration space which can cover hyper-parameter settings, pipeline components and/or network architecture components. P is the set of all prior scalar evaluations $P_{i,j} = P(\theta_i, t_j)$ of configuration θ_i on task t_j , according to a predefined evaluation measure, e.g. accuracy, and model evaluation technique, e.g. cross-validation. P_{new} is the set of known evaluations P_i (usually gathered beforehand),new on a new task t_{new}.

A meta-learner L is now trained on meta-data $P \cup P_{new}$ to predict recommended configurations Θ^*_{new} for a new task t_{new}. P_{new} is learned by the meta-learning technique itself in an iterative fashion, sometimes warm-started with an initial P'_{new} generated by another method (VANSCHOREN, 2018).

2.3 Knowledge Transference from previous models

2.3.1 Hyper parameter Importance: Configuring the search space

When processing a new task t_{new} it is expected to not have any kind of evaluation about the response surface Ψ (Equation 3), hence $P_{new} = \emptyset$; nevertheless it is still possible to learn a function $f : \Theta \times T \rightarrow \theta^*_k, k = 1, ..., K$ (a set of hyper-parameters given past tasks) yielding a set of recommended configurations independent of t_{new}. These θ^*_k can be evaluated on t_{new} to select the best one. Such approaches often produce an aggregated ranking (where equally good but faster algorithms are ranked higher), i.e. an ordered set θ^*_k, typically done by discretizing Θ into a set of candidate configurations θ_i , also called a portfolio, evaluated on a large number of tasks t_j (HUTTER *et al.*, 2018).

To find the best θ^* for a task t_{new} , never before seen, a simple anytime method is to select the top-K configurations [21], going down the list and evaluating each configuration on t_{new} in turn. This evaluation can be halted after a predefined value for K, a time budget, or when a sufficiently accurate model is found (YOGATAMA; MANN, 2014).

Prior evaluations can also be used to learn a better configuration space Θ^* (i.e. relevant HP). While independent from t_{new}, exploring only the most relevant regions of the configuration space can radically speed up the search for optimal models.

This approach is explored in (van Rijn; Hutter, 2017), where a functional ANOVA rule a set of HP as important if they can explain most of the variance in algorithm performance on a given task (in Rijin's work,this was explored using 250,000 OpenML experiments with 3 algorithms across 100 datasets). An alternative approach is to first learn an optimal HP default setting, and then define HP importance as the performance gain that can be achieved by tuning the HP instead of leaving it at that default value. Indeed, even though a HP may cause a lot of variance, it may also have one specific setting that always results in good performance. Finally, the tunability of each HP is estimated by observing how much improvement can still be obtained by optimizing it.

2.3.2 Configuration Transference

To find a good set of Hyper-parameters for a specific task t_{new}, the new configuration can be reproduced from configurations found in previous tasks t_j by just finding the similarity between them. One option to calculate this similarity is to evaluate a number either recommended or random configurations on t_{new}, creating therefore new information P_{new}. In case that the evaluations $P_{i,new}$ are similar to $P_{i,j}$, then t_j and t_{new} can be considered intrinsically similar, based on empirical evidence. This knowledge can be used to train a meta-learner that predicts a recommended set of configurations Θ^*_{new} for t_{new}. Moreover, every selected θ^*_{new} can be evaluated and included in P_{new}, repeating the cycle and collecting more empirical evidence to learn which tasks are similar to each other.

Knowledge transference can also be done by generating $surrogatemodels s_j(\theta_i) = P_{i,j}$ for all previous tasks t_j, trained using all available $\mathbf{P}$. Then, similarity can be calculated in terms of the error between $s_j(\theta_i)$ and $P_{i,new}$; if the surrogate model created for t_j generate accurate predictions for t_new, then those tasks are intrinsically similar. This is usually done in combination with Bayesian optimization to determine the next θ_i .

In Wistuba et al. (WISTUBA *et al.*, 2016), surrogate models are trained based on Gaussian Processes (GPs) for every prior task, plus one for t_new , and combine them into a weighted, normalized sum, with the (new) predicted mean μ defined as the weighted sum of the individual μ_j's (obtained from prior tasks t_j). The weights of the μ_j's are computed using the Nadaraya-Watson kernel-weighted average, where each task is represented as a vector of relative Pairwise Hyper-parameter Performance Rankings, and the Epanechnikov quadratic kernel is used to measure the similarity between the vectors of t_j and t_new. The more similar tj is to t_new , the larger the weight s_j , increasing the influence of the surrogate model for t_j. The intuition behind using Pairwise Hyper-parameter Performance

Rankings is to select all paired combinations of hyperparameter configurations (θ_i, θ_j) evaluated on the new task t_{new}, and estimate how often two previous tasks t_{old}, t'_{old} agree on the configuration ranking. Usually, it is assumed that the HP configurations evaluated on t_{new} have also been evaluated on all previous task from the meta-data set T, P, Θ, although in a general context this is likely not the case. Witsuba et.al. proposed the use of the predictors obtained in the first step to approximate the performances and therefore to overcome this limitation while also comparing the resulting rank from a similarity comparison of task's meta-features. Along this line, in [45] is proposed a combination of the predictive distributions of the individual Gaussian processes, but the weights are computed following the agnostic Bayesian ensemble instead, which weights predictors according to an estimate of their generalization performance.

In contrast, meta-data can also be transferred in the acquisition function rather than the surrogate model. The surrogate model is only trained on $P_{i,new}$, but the next θ_i to evaluate is provided by an acquisition function which weights the expected improvement on P_i, new and the predicted improvements on all prior $P_{i,j}$. The weights of the prior tasks can again be defined via the accuracy of the surrogate model or via relative landmarks. The weight of the expected improvement component is gradually increased with every iteration as more evidence $P_{i,new}$ is collected. [69] [187]

Meta-data can also be extracted by characterizing the task at hand (meta-features). Each task $t_j \in T$ is described with a vector $m(t_j) = (m_j, 1, ..., m_{j,K})$ of K meta-features $m_{j,k} \in M$, the set of all known meta-features. This can be used to define a task similarity measure based on, for instance, the Euclidean distance between $m(t_i)$ and $m(t_j)$, so that we can transfer information from the most similar tasks to the new task t_new. Moreover, together with prior evaluations P, we can train a meta-learner L to predict the performance $P_{i,new}$ of configurations θ_i on a new task t_new.

References

AMAZON. **Amazon: Automatic model tuning**. 2018. Disponível em: ⟨https://docs.aws.amazon.com/sagemaker/latest/dg/automatic-model-tuning.html⟩.

BAKER, B.; GUPTA, O.; NAIK, N.; RASKAR, R. Designing neural network architectures using reinforcement learning. **CoRR**, abs/1611.02167, 2016. Disponível em: ⟨http://arxiv.org/abs/1611.02167⟩.

BARDENET, R.; BRENDEL, M.; KéGL, B.; SEBAG, M. Collaborative hyperparameter tuning. In: DASGUPTA, S.; MCALLESTER, D. (Ed.). **Proceedings of the 30th International Conference on Machine Learning**. Atlanta, Georgia, USA: PMLR, 2013. (Proceedings of Machine Learning Research, 2), p. 199–207. Disponível em: ⟨http://proceedings.mlr.press/v28/bardenet13.html⟩.

BERGSTRA, J.; BENGIO, Y. Random search for hyper-parameter optimization. **J. Mach. Learn. Res.**, JMLR.org, v. 13, n. 1, p. 281–305, fev. 2012. ISSN 1532-4435. Disponível em: ⟨http://dl.acm.org/citation.cfm?id=2503308.2188395⟩.

BROCHU, E.; CORA, V. M.; FREITAS, N. de. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. **CoRR**, abs/1012.2599, 2010. Disponível em: ⟨http://arxiv.org/abs/1012.2599⟩.

FALKNER, S.; KLEIN, A.; HUTTER, F. BOHB: Robust and efficient hyperparameter optimization at scale. In: DY, J.; KRAUSE, A. (Ed.). **Proceedings of the 35th International Conference on Machine Learning**. Stockholmsmässan, Stockholm Sweden: PMLR, 2018. (Proceedings of Machine Learning Research, v. 80), p. 1437–1446. Disponível em: ⟨http://proceedings.mlr.press/v80/falkner18a.html⟩.

HUTTER, F.; KOTTHOFF, L.; VANSCHOREN, J. (Ed.). **Automated Machine Learning: Methods, Systems, Challenges**. [S.l.]: Springer, 2018. In press, available at http://automl.org/book.

HUTTER, F.; LüCKE, J.; SCHMIDT-THIEME, L. Beyond manual tuning of hyperparameters. **KI**, v. 29, n. 4, p. 329–337, 2015. Disponível em: ⟨http://dblp.uni-trier.de/db/journals/ki/ki29.html#HutterLS15⟩.

LI, F.-F. **Cloud AutoML: Making AI accessible to every business**. 2018. Disponível em: ⟨https://www.blog.google/products/google-cloud/cloud-automl-making-ai-accessible-every-business/⟩.

RASMUSSEN, C. E.; WILLIAMS, C. K. I. **Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)**. [S.l.]: The MIT Press, 2005. ISBN 026218253X.

SIGOPT. **SIGOPT: Improve ML models 100x faster**. 2018. Disponível em: ⟨https://sigopt.com/⟩.

van Rijn, J. N.; Hutter, F. Hyperparameter Importance Across Datasets. **arXiv e-prints**, p. arXiv:1710.04725, Oct 2017.

VANSCHOREN, J. Meta-learning: A survey. **CoRR**, abs/1810.03548, 2018. Disponível em: ⟨http://arxiv.org/abs/1810.03548⟩.

WISTUBA, M.; SCHILLING, N.; SCHMIDT-THIEME, L. Two-stage transfer surrogate model forźautomatic hyperparameter optimization. In: **European Conference on Machine Learning and Knowledge Discovery in Databases - Volume 9851**. Berlin, Heidelberg: Springer-Verlag, 2016. (ECML PKDD 2016), p. 199–214. ISBN 978-3-319-46127-4. Disponível em: ⟨https://doi.org/10.1007/978-3-319-46128-1_13⟩.

Name	Formula	Rationale	Variants
Nr instances	n	Speed, Scalability (Michie et al., 1994)	p/n, $log(n)$, log(n/p)
Nr features	p	Curse of dimensionality (Michie et al., 1994)	$log(p)$, % categorical
Nr classes	c	Complexity, imbalance (Michie et al., 1994)	ratio min/maj class
Nr missing values	m	Imputation effects (Kalousis, 2002)	% missing
Nr outliers	o	Data noisiness (Rousseeuw and Hubert, 2011)	o/n
Skewness	$\frac{E(X-\mu_X)^3}{\sigma_X^3}$	Feature normality (Michie et al., 1994)	min,max,μ,σ,q_1,q_3
Kurtosis	$\frac{E(X-\mu_X)^4}{\sigma_X^4}$	Feature normality (Michie et al., 1994)	min,max,μ,σ,q_1,q_3
Correlation	$\rho_{X_1X_2}$	Feature interdependence (Michie et al., 1994)	min,max,μ,σ,ρ_{XY}
Covariance	$cov_{X_1X_2}$	Feature interdependence (Michie et al., 1994)	min,max,μ,σ,cov_{XY}
Concentration	$\tau_{X_1X_2}$	Feature interdependence (Kalousis and Hilario, 2001)	min,max,μ,σ,τ_{XY}
Sparsity	sparsity(X)	Degree of discreteness (Salama et al., 2013)	min,max,μ,σ
Gravity	gravity(X)	Inter-class dispersion (Ali and Smith-Miles, 2006a)	
ANOVA p-value	$p_{val}{}_{\mathtt{X}_1X_2}$	Feature redundancy (Kalousis, 2002)	$p_{val}{}_{XY}$(Soares et al., 2004)
Coeff. of variation	$\frac{\sigma_Y}{\mu_Y}$	Variation in target (Soares et al., 2004)	
PCA ρ_{λ_1}	$\sqrt{\frac{\lambda_1}{1+\lambda_1}}$	Variance in first PC (Michie et al., 1994)	$\frac{\lambda_1}{\sum_i \lambda_i}$(Michie et al., 1994)
PCA skewness		Skewness of first PC (Feurer et al., 2014)	PCA kurtosis
PCA 95%	$\frac{dim_{95\%var}}{p}$	Intrinsic dimensionality (Bardenet et al., 2013)	
Class probability	$P(\mathtt{C})$	Class distribution (Michie et al., 1994)	min,max,μ,σ
Class entropy	$H(\mathtt{C})$	Class imbalance (Michie et al., 1994)	
Norm. entropy	$\frac{H(\mathtt{X})}{log_2 n}$	Feature informativeness (Castiello et al., 2005)	min,max,μ,σ
Mutual inform.	$MI(\mathtt{C},\mathtt{X})$	Feature importance (Michie et al., 1994)	min,max,μ,σ
Uncertainty coeff.	$\frac{MI(\mathtt{C},\mathtt{X})}{H(\mathtt{C})}$	Feature importance (Agresti, 2002)	min,max,μ,σ
Equiv. nr. feats	$\frac{H(C)}{\overline{MI(C,X)}}$	Intrinsic dimensionality (Michie et al., 1994)	
Noise-signal ratio	$\frac{\overline{H(X)}-\overline{MI(C,X)}}{\overline{MI(C,X)}}$	Noisiness of data (Michie et al., 1994)	
Fisher's discrimin.	$\frac{(\mu_{c1}-\mu_{c2})^2}{\sigma_{c1}^2-\sigma_{c2}^2}$	Separability classes c_1, c_2 (Ho and Basu, 2002)	See Ho:2002
Volume of overlap		Class distribution overlap (Ho and Basu, 2002)	See Ho and Basu (2002)
Concept variation		Task complexity (Vilalta and Drissi, 2002)	See Vilalta (1999)
Data consistency		Data quality (Köpf and Iglezakis, 2002)	See Köpf and Iglezakis (2002)
Nr nodes, leaves	$\|\eta\|, \|\psi\|$	Concept complexity (Peng et al., 2002)	Tree depth
Branch length		Concept complexity (Peng et al., 2002)	min,max,μ,σ
Nodes per feature	$\|\eta_X\|$	Feature importance (Peng et al., 2002)	min,max,μ,σ
Leaves per class	$\frac{\|\psi_c\|}{\|\psi\|}$	Class complexity (Filchenkov and Pendryak, 2015)	min,max,μ,σ
Leaves agreement	$\frac{n_{\psi_i}}{n}$	Class separability (Bensusan et al., 2000)	min,max,μ,σ
Information gain		Feature importance (Bensusan et al., 2000)	min,max,μ,σ, gini
Landmarker(1NN)	$P(\theta_{1NN}, t_j)$	Data sparsity (Pfahringer et al., 2000)	See Pfahringer et al. (2000)
Landmarker(Tree)	$P(\theta_{Tree}, t_j)$	Data separability (Pfahringer et al., 2000)	Stump,RandomTree
Landmarker(Lin)	$P(\theta_{Lin}, t_j)$	Linear separability (Pfahringer et al., 2000)	Lin.Disciminant
Landmarker(NB)	$P(\theta_{NB}, t_j)$	Feature independence (Pfahringer et al., 2000)	See Ler et al. (2005)
Relative LM	$P_{a,j} - P_{b,j}$	Probing performance (Fürnkranz and Petrak, 2001)	
Subsample LM	$P(\theta_i, t_j, s_t)$	Probing performance (Soares et al., 2001)	

Figure 3: Reproduced from (VANSCHOREN, 2018). Overview of commonly used meta-features. Groups from top to bottom: simple, statistical, information-theoretic, complexity, model-based, and landmarkers. Continuous features X and target Y have mean μ_X , stdev σ_X , variance σ_X^2. Categorical features X and class C have categorical valuesπ_i, conditional probabilities $\pi_{i|j}$, joint probabilities $\pi_{i,j}$, marginal probabilities $\pi_{i+} = \sum_j \pi_{ij}$, entropy $H(X) = -\sum_i \pi_{i+} log_2(\pi_{i+})$.

YOGATAMA, D.; MANN, G. Efficient Transfer Learning Method for Automatic Hyperparameter Tuning. In: KASKI, S.; CORANDER, J. (Ed.). **Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics**. Reykjavik, Iceland: PMLR, 2014. (Proceedings of Machine Learning Research, v. 33), p. 1077–1085. Disponível em: ⟨http://proceedings.mlr.press/v33/yogatama14.html⟩.

ZOPH, B.; LE, Q. V. Neural architecture search with reinforcement learning. In: . [s.n.], 2017. Disponível em: ⟨https://arxiv.org/abs/1611.01578⟩.

References