Learning Multiple Defaults for Machine Learning Algorithms

Florian Pfisterer* **and Jan N. van Rijn†** **and Philipp Probst*** **and Andreas Müller†** **and Bernd Bischl***
*Ludwig Maximilian University of Munich, Germany
†Columbia University, New York, U.S.A.

Abstract

The performance of modern machine learning methods highly depends on their *hyperparameter configurations*. One simple way of selecting a configuration is to use *default settings*, often proposed along with the publication and implementation of a new algorithm. Those default values are usually chosen in an ad-hoc manner to work *good enough* on a wide variety of datasets. To address this problem, different automatic hyperparameter configuration algorithms have been proposed, which select an optimal configuration per dataset. This principled approach usually improves performance, but adds additional algorithmic complexity and computational costs to the training procedure. As an alternative to this, we propose learning a set of complementary default values from a large database of prior empirical results. Selecting an appropriate configuration on a new dataset then requires only a simple, efficient and embarrassingly parallel search over this set. We demonstrate the effectiveness and efficiency of the approach we propose in comparison to random search and Bayesian Optimization.

Introduction

The performance of most machine learning algorithms highly depends on their hyperparameter settings. Various methods exist to automatically optimize hyperparameters, including random search (Bergstra and Bengio 2012), Bayesian optimization (Snoek, Larochelle, and Adams 2012; Hutter, Hoos, and Leyton-Brown 2011), meta-learning (Brazdil et al. 2008) and bandit-based methods (Li et al. 2017). Depending on the algorithm, properly tuning the hyperparameters yields a considerable performance gain (Lavesson and Davidsson 2006).

Despite this acknowledged importance of tuning the hyperparameters, in many practical cases it is often neglected.

Possible reasons for this are the additional run time, code complexity and experimental design questions. It has indeed been pointed out that properly deploying a hyperparameter tuning strategy requires expert knowledge (Probst, Boulesteix, and Bischl 2019; van Rijn and Hutter 2018).

When parameters are not tuned, they are often set to a default value provided by the software authors. While not tuning parameters at all can be detrimental, defaults provide a fallback for cases, where no additional knowledge is available. Wistuba, Schilling, and Schmidt-Thieme (2015b) proposed to extend the notion of pre-specified defaults to ordered sets of defaults, combining the prior knowledge encoded in default values with the flexibility of optimization procedures. This work directly builds upon this notion. Our ordered sets of defaults are diverse lists of parameter settings for a particular algorithm, ordered by their performance across datasets. This can be seen as an extension of the classical exhaustive grid-search: Instead of searching all possible combinations in the grid, we keep only those configurations that historically (on a collection of benchmark datasets) performed well. Given that we eliminate most candidates using prior data, we can then afford to start with a very fine grid, approximating the results of a continuous optimization procedure.

A different perspective on multiple defaults is as a special case of meta-learning: we build a model using a collection of benchmark datasets, that allows us to predict good candidate parameters for a new dataset. Only we do not use any properties of the new dataset, and always predict the same ordered set of candidates.

Compared with more complex optimization procedures, multiple defaults have several benefits.

Ease of implementation Sets of defaults can be easily computed in advance and implemented as look-up tables. Only simple resampling is required to select the best configuration from the set.

Ease of use The concept of multiple defaults is easy to understand and does not introduce any additional parameters or specification of parameter ranges. The number of default configurations to be evaluated is determined by computational constraints.

Strong anytime performance Defaults can achieve high performances even if only few evaluations can be performed. If additional computational resources are available, they can be used in combination with other optimization methods, e.g., as good initial values for conventional tuning methods.

Embarrassingly parallel Evaluation of the ordered set of defaults can be arbitrarily parallelized.

Robustness Defaults do not suffer from the problems asso-

ciated with optimizing over high-dimensional, mixed-type input spaces nor from potential crashes or failures of the optimization method.

We conjecture that a small set of well-performing configurations can perform quite well on a broad set of datasets. We will leverage a large set of historic datasets, and the performance results of prior experiments that are readily available on OpenML (Vanschoren et al. 2014; van Rijn 2016). While proper hyperparameter tuning techniques remain preferable when the resources and expertise are available, simply iterating over an orderd set of defaults might be a viable alternative when this is not the case.

We define a model-agnostic approach of learning not only a single configuration parameter configuration, but instead a set of configurations, that *together* perform well on new datasets. That is, any given set of defaults should contain at least one parameter configuration that performs well on a given dataset. These defaults can be written down and hard coded into software implementations, and thus easily be adapted by users.

Our contributions are the following. 1) We describe two methods, an exact exhaustive method and a greedy method, that acquire a list of defaults, based on predictions from surrogate models. In particular, the surrogate models allow to scale the method in a realistic setting to arbitrary algorithms and sizes of hyperparameter spaces. 2) we show that solving the underlying problem in an exact manner is NP-hard, 3) due to this NP-hardness, we conduct a small experiment comparing the greedy and exact exhaustive approach, and 4) empirically evaluate the defaults obtained through the greedy approach in a large benchmark using 6 configurable state-of-the-art ML algorithms, containing 2–10 hyperparameters, on a wide range of datasets. In this experiment we compare defaults found with the described method against random search as well as Bayesian Optimization. We show that the method we propose requires about 4 times fewer model evaluations to achieve similar performances than random search or Bayesian optimization.

Related work

There are various openly available Machine Learning workbenches, implementing many algorithms. Some popular examples in scientific communities are Weka (Hall et al. 2009), scikit-learn (Pedregosa et al. 2011) and mlr (Bischl et al. 2016). Most algorithms have hyperparameters, that in turn have default values. It has often been noted that using default values does not yield adequate performance, and can be improved by hyperparameter optimization (Bergstra et al. 2011; Bergstra and Bengio 2012; Hutter, Hoos, and Leyton-Brown 2011; Snoek, Larochelle, and Adams 2012; Li et al. 2017). Lavesson and Davidsson (2006) investigate for a given algorithm how strong the impact of different hyperparameter configurations can be across datasets. They infer that the importance of *good* settings varies between algorithms, and that parameter tuning can be more important than the choice of an algorithm.

Many techniques have been proposed for performing hyperparameter optimization. Bergstra and Bengio (2012) compare grid search and random search, and concluded that although both approaches are rather simple, they already yield great performance gains compared to using a single default setting. Furthermore, they noted that given the same number of iterations, random search is in many practical cases preferable over grid search. We will therefore use random search as the baseline in all our experiments. Successive Halving (Jamieson and Talwalkar 2016) and Hyperband (Li et al. 2017) are full-exploration bandit-based methods, using initially small but increasing budgets to prioritize evaluation of particular hyperparameter settings. The field of model based optimization (also often referred to as MBO or Bayesian Optimization) uses an internal *empirical performance model*, which tries to learn a surrogate model of the objective function while optimizing it (Bergstra et al. 2011; Snoek, Larochelle, and Adams 2012; Hutter, Hoos, and Leyton-Brown 2011; Bischl et al. 2018). It focuses the search towards regions that are promising according to the model.

Alternatively, the field of meta-learning attempts to leverage knowledge obtained from experiments on prior datasets to a new dataset (Brazdil et al. 2008). The underlying principle is to represent each dataset as a vector of numerical attributes. The assumption is that on datasets, similar algorithms that work well on these. A so-called meta-model can be trained to predict the performance of arbitrary configurations on new, unseen datasets (Gomes et al. 2012; Leite, Brazdil, and Vanschoren 2012). Several approaches attempt to combine the paradigms of meta-learning and hyperparameter optimization, for example by warm starting hyperparameter optimization techniques (Feurer et al. 2015; Feurer, Springenberg, and Hutter 2015; Wistuba, Schilling, and Schmidt-Thieme 2015a), or in a streaming setting (Yogatama and Mann 2014). An approach very close to ours is investigated in (Wistuba, Schilling, and Schmidt-Thieme 2015b). From a perspective of finding good initial points for warm-stating Bayesian optimization, they propose to *greedily* find a set of configurations, that minimizes the sum of risk across several datasets. This approach has severe limitations, we intend to alleviate in our work. First, the procedure requires hyperparameters evaluated on a grid across several datasets. This will scale exponentially with hyperparameter space dimensionality, and thus be practically infeasible for algorithms that require multiple hyperparameters. Additionally, it has been noted in (Bergstra and Bengio 2012), that grid search, especially when evaluated on coarse grid, often emphasizes regions that do not matter and suffers from poor coverage in important dimensions.

While all these methods yield convincing results and generated a considerable amount of scientific follow-up, they are by no means easy to deploy. Methods from the search paradigm require knowledge of which hyperparameters are important as well as suitable ranges to optimize over (Probst, Boulesteix, and Bischl 2019; van Rijn and Hutter 2018). Methods from the meta-learning paradigm require a set of historic datasets and meta-features to train on. Finding an

informative set of training data and meta-features is still an open scientific question (Pinto, Soares, and Mendes-Moreira 2016; van Rijn 2016).

Most similar to the approach that we introduce are the works of Wistuba, Schilling, and Schmidt-Thieme (2015a) and Wistuba, Schilling, and Schmidt-Thieme (2015b). Additionally, the work of Feurer et al. (2018) also involves lists of defaults, although they do not detail on how to construct these.

Wistuba, Schilling, and Schmidt-Thieme (2015b) propose a method that selects defaults based on meta-data from different algorithms. As the first reference to the multiple defaults, this work was a great contribution, but it also came with drawbacks. It requires a full grid of hyperparameters, evaluated on all tasks that have been encountered in the past. As this is not a realistic requirement, this makes applying this method unpractical, and it does not scale beyond a few hyperparameters. In fact, in the experimental evaluation only 2 algorithms with no more than 4 hyperparameters were considered. Parameters suggested by this method can only stem from the pre-specified grid.

Alternatively, Wistuba, Schilling, and Schmidt-Thieme (2015a) propose a method that is able to warm-start Bayesian Optimization procedures, which can essentially also be seen as a set of defaults. The approach they propose requires a differentiable model, which in theory means that it can only work with numeric and unconditional hyperparameters. The method does not provide a fixed list of parameters, but instead adaptively learns them for a new problem instance. The methods we propose do not require any of these. Additionally, the approach we propose does not rely on a predetermined budget of optimizations. For each different budget in terms of function evaluations, a different set of defaults will be recommended. It can therefore not successfully be applied in the more realistic case where the budget is run time.

Additionally, both works only experiment with 2 algorithms and do not apply a *nested* cross-validation procedure, which is required to make plausible conclusions when hyperparameter optimization is involved (Cawley and Talbot 2010).

Method

Consider a target variable y, a feature vector X, and an unknown joint distribution P on (X, y), from which we have sampled a dataset $\mathcal{D}$ containing $|\mathcal{D}|$ observations.

A machine learning (ML) algorithm tries to approximate the functional relationship between X and y by producing a prediction model $\hat{f}_\theta(X)$, controlled by a multi-dimensional hyperparameter configuration $\theta \in \Theta$. In order to measure prediction performance pointwise between a true label y and its prediction $\hat{f}(X)$, we define a loss function $L(y, \hat{f}(X))$. We are naturally interested in estimating the expected risk of the inducing algorithm, w.r.t. θ on new data, also sampled from $\mathcal{P}$: $R_\mathcal{P}(\theta) = E(L(y, \hat{f}(X))|\mathcal{P})$. Thus, $R_\mathcal{P}(\theta)$ quantifies the expected predictive performance associated with a hyperparameter configuration θ for a given data distribution, learning algorithm and performance measure.

Given a certain data distribution, a certain learning algorithm and a certain performance measure, this mapping encodes the numerical quality for any hyperparameter configuration θ.

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

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

For a set of M configurations $\Theta_M = \{\theta_1, \ldots, \theta_M\}$ and with a slight abuse of notation, we could define and visualize

$$R(\Theta_M) = (R_k(\theta_m))_{k=1,...,K;m=1,...,M}$$

as a matrix of dimensions $K \times M$ of risks for different configurations and datasets. Here, the k-th row-vector of $R(\Theta_M)$ contains the risks of all configurations in Θ_M, evaluated on dataset k; while the m-column contains the empirical distribution of risk for θ_m across all datasets.

Defining a set of optimal defaults

Hyperparameter optimization methods usually try to find an optimal θ for a given dataset. In this work on the other hand, we try to find a fixed-size set Θ_{def} that works well over a wide variety of datasets, in the sense that Θ_{def} contains at least one configuration that works well on any given dataset (and in that case we do not really care about the performance of the other configurations on that dataset). In order for this to be feasible in practice, the individual datasets need to have at least some common structure from which we can generalize. This patterns can in general stem from algorithm properties, such as combinations of individual hyperparameters that work well together, or alternatively from similar data situations. By using a large number of datasets, we hope to find defaults that are less tailored to specific datasets, but generalize well. This allows focusing on the first kind of patterns. If patterns can be transferred from a set of datasets to a new dataset, one would assume that, given there exists a common structure, learned configurations perform significantly better than a set of randomly drawn data points on the same held out dataset.

In practical terms, given our set Θ_{def}, we would trivially evaluate all configurations in parallel (e.g., by cross-validation), and then simply select the best one to obtain the final fit for our ML algorithm.

Algorithm 1: ML algorithm with multiple defaults

Input: Dataset $\mathcal{D}$, Inducer $\mathcal{A}$, candidate configurations Θ_{def} of size n
Result: Model induced by $\mathcal{A}$ on data $\mathcal{D}$
Cross-validate $\mathcal{A}$ on $\mathcal{D}$ with all $\theta_i \in \Theta_{def}$;
Select best θ^* from Θ_{def};
Fit $\mathcal{A}$ on complete $\mathcal{D}$ with θ^* and obtain ML model;

Hence, an optimal set of defaults Θ_{def} should contain complementary defaults, i.e., some configurations $\theta \in \Theta_{def}$

can cover for shortcomings of other configurations from the same set. This can be achieved by jointly optimizing over the complete set.

Risk of a set of configurations The risk of a set of configurations $\Theta_{def} \subset \Theta$, for datasets $1, \ldots, K$ is given by:

$$G(\Theta_{def}) = h\left(\min_{j=1,\ldots,n} R_1(\theta_j), \ldots, \min_{j=1,\ldots,n} R_K(\theta_j))\right)$$

We aggregate this to a single scalar performance value by using a function h, e.g., the median. For aggregation to be sensible, we assume that performances across all K datasets are commensurable, which is a strong assumption.

The optimal set of defaults Θ_{def} of size n is then given by

$$\underset{\Theta_{def} \subset \Theta, |\Theta_{def}|=n}{\arg\min} \quad G(\Theta_{def})). \tag{1}$$

We compare two methods, namely an *exact discretized* and a *greedy* search approach, that allow us to obtain such sets of defaults.

Computational Complexity This problem is a generalization of the *Maximum coverage problem*, which was proven NP-hard by Nemhauser, Wolsey, and Fisher (1978). The original maximum coverage problem assumes Boolean input variables, s.t. each set covers covers certain elements, whereas the formulation in our context assumes scalar input variables. This adds additional complexity to the Exact Discretized Formulation.

Exact Discretized Optimization

A discrete version of this problem can be formulated as a Mixed Integer Programming problem. The solution we propose is specific to the aggregation functions sum and mean. Other aggregation function can be incorporated as well, at the cost of introducing more variables and constraints. Given a discrete set of M configurations $\{\theta_1, \ldots, \theta_M\} \subset \Theta$, we first define

$$Q(k, m) = \{s : R_k(\theta_s) < R_k(\theta_m)\} \tag{2}$$

for each $m \in \{1, \ldots, M\}$ and given k, where $R_k(\theta_s)$ is the empirical risk of θ_s on $\mathcal{P}_k$.

Intuitively, $Q(k, m)$ now is a set of integer indices such that for each $q \in Q(k, m)$ holds that the risk of θ_q is lower than the risk of θ_m on dataset k. The definition of Q assumes no ties. Ties may be broken arbitrarily, but must be consistent. For example, comparing $R_k(\theta_s) < R_k(\theta_m)$ can be replaced by the lexicographical comparison $(R_k(\theta_s), s) < (R_k(\theta_m), m)$.

In order to obtain a set of n defaults, the goal is to minimize

$$\sum_{k=1}^{K} \sum_{m=1}^{M} \Psi_{k,m} \cdot R_k(\theta_m) \tag{3}$$

subject to

$$\sum_{m=1}^{M} \phi_m = n \tag{4}$$

$$\forall k : \forall m : \Psi_{k,m} \geq \phi_m - \sum_{s \in Q(k,m)} \phi_s \tag{5}$$

$$\forall k : \forall m : \Psi_{k,m} \geq 0 \tag{6}$$

$$\forall k : \sum_{m=1}^{M} \Psi_{k,m} = 1 \tag{7}$$

The free variables are Ψ (a matrix of size $K \times M$) and ϕ (a vector of size M, containing booleans). After the optimization procedure, $\phi_m = 1$ if and only if θ_m is part of the optimal set of defaults. Eq. 4 ensures that exactly the required number of defaults will be selected. Matrix Ψ does not need to be restricted to a specific type, but will only contain values $\{0, 1\}$ (we will see why further on). After the optimization procedure, element $\Psi_{k,m}$ will be 1 if and only if configuration θ_m has the lowest risk on distribution i out of all the configurations that are in the set of defaults. Formally, $\Psi_{k,m} = 1$ if and only if $\forall j : \phi_j = 0 \vee R_k(\theta_j) > R_k(\theta_i)$ (this is enforced by Eq. 5). The optimization criterion presented in Eq. 3 is the Hadamard product between the matrix Ψ and the matrix of risks R. The outcome of this formula is equal to the definition of the risk of a set of configurations, with h being the sum. The aggregation functions sum and mean lead to the same set of defaults.

We will show that the constraints ensure the correct behaviour as described above. For an element $\Psi_{k,m}$, there are two factors that influence the minimum value: i) whether ϕ_i is part of the selected set of defaults, and ii) whether there are other configurations in the selected set of defaults that have a lower risk on distribution k. If either or both conditions hold, $\Psi_{k,m} \geq 0$. If neither of the conditions hold, $\Psi_{k,m} \geq 1$. Given that the risk is always positive, and matrix Ψ can not contain negative numbers (Eq. 6), the optimizer will aim for as low as possible values in Ψ, in this case either 0 or 1. The constraint presented in Eq. 7 is formally not necessary, but removes the requirement that all values of R need to be positive.

Greedy Search

A computationally more feasible solution is to instead iteratively add defaults to the set in a greedy forward fashion, starting from the empty set as follows:

for i = $1, \ldots, n$:

$$\theta_{def,i} := \underset{\theta \in \Theta}{\arg\min} \quad G(\{\theta\} \cup \Theta_{def,i-1}) \tag{8}$$

$$\Theta_{def,i} := \{\theta_{def,1}, \ldots, \theta_{def,i}\} \tag{9}$$

where $\Theta_{def,i} = \emptyset$, and the final solution $\Theta_{def} = \Theta_{def,n}$.

An advantage of a greedy approach is that it results in a growing sequence of default subsets for increasing budgets. So if we compute a set of size $n = 100$ through the above approach, e.g., in practice a user might opt to only evaluate the first 10 configurations of the sequence (due to budget constraints) or to sequentially run through in them in an iterative process and stop when a desired performance is reached; in other words, it is an anytime algorithm. A possible disadvantage is, that for a given size n the set of parameters might not be optimal according to Equation 1.

Surrogate Models

The exact discretized approach requires evaluating $R_k(\theta)$ on a fine discretization of the search space θ, while greedy search even requires optimizing G, a complex function of $R_k(\theta)$. It is possible to estimate $R_k(\theta)$ empirically using cross-validation. In this case evaluation of $R_k(\theta)$ corresponds to evaluating a particular hyperparameter setting with cross-validation, which involves building several models. While the proposed method only requires us to do this *once* to obtain multiple defaults to be used in the future, building models on a fine grid is not tractable, especially when the number of hyperparameters is high. Therefore we employ *surrogate models* that predict the outcome of a given performance measure and algorithm for a given hyperparameter configuration. We train one model for each dataset on underlying performance data (Eggensperger et al. 2015). This provides us with a fast approximate way to evaluate the performance of any given configuration, without the requirement of costly training and evaluating models using cross-validation.

Additionally, because we can not practically evaluate every $\theta \in \Theta$ on each dataset, as $|\Theta|$ can be infinitely big depending on the algorithm, we instead only evaluate a large random sample from Θ. Cheap approximations can then be obtained via surrogate models.

Standardizing results We mitigate the problem of lacking commensurability between datasets by normalizing performance results on a per-dataset basis to mean 0 and standard deviation 1 before training surrogate models. A drawback to this is, that some information with regards to the absolute performance of the algorithm and the spread across different configurations is lost.

On the choice of an aggregation function Considering the fact that performances on different datasets are usually not commensurable (Demšar 2006), an appropriate aggregation function or scaling is required to obtain sensible defaults. One approach can be using quantiles. Depending on the choice of quantile, this either emphasizes low risks, i.e., datasets that are relatively easy anyways (by choosing the *minimum*), or high risks, i.e., hard datasets (when choosing the *maximum*). This corresponds to optimizing an optimistic case (i.e., optimizing a best case scenario) or a pessimistic scenario (i.e., hedging against the worst case). Several other methods from decision theory literature, such as the Hodges-Lehmann criterion (Hodges and Lehmann 1952) could also be used. From a theoretical point of view, it is not immediately clear which aggregation functions benefit the method most. From a small experiment, excluded for brevity, we concluded that in practice, the choice of an aggregation function has negligible impact on the performance of the set of defaults across datasets. Hence, we use the median over datasets as aggregation function.

Defaults across algorithms In addition to providing a list of defaults for a given machine learning algorithm, the proposed method can also generate sets of defaults across a set of different learners. In this case, surrogate models need to be trained on results across all learners standardized per task. In this setting, the parameter space Θ is defined by a hyperparameter that reflects the choice of an algorithm and the conjunction of the parameter spaces of all learners.

Experimental Setup

We will perform two experiments. We first present an experiment on small scale to compare the defaults obtained from the exact discretized approach against the greedy approach. As the exact discretized approach is presumably intractable, we can only perform a direct comparison on a small dataset, using a small number of defaults. In the second experiment, which is one of the main contributions of this work, we compare defaults obtained from the greedy approach against random search and Bayesian Optimization. This section describes the setup of these experiments. Due to the presumable intractability, the experiment comparing the greedy and exact discretized approach slightly deviates from this, in the sense that it operates on a discretized version of the problem, and uses a lower number of defaults (at most 6).

Estimations of the performance on future datasets can be obtained by evaluating a set of n defaults Θ_{def} using Leave-One-Dataset-Out Cross-validation over K datasets. As a baseline, we compare to *random search* with several budgets and *Bayesian Optimization* with 32 iterations. The latter simulates scenarios where the number of available evaluations is limited, for example due to computational constraints.

For each $k \in \{1, \ldots, K\}$, we repeat the following steps:

- **Defaults**
  for $n \in \{1, 2, 4, 8, 16, 32\}$:
  - Learn a set of n defaults Θ_{def} on datasets $\{\{1, \ldots, K\} \setminus k\}$
  - Run the proposed *greedy* algorithm with Θ_{def} on OpenML Task k – embedded in nested CV.
- **Random search**
  for $i \in \{4, 8, 16, 32, 64\}$:
  - Run random search with budget i on OpenML Task k – embedded in nested CV.
- **Bayesian Optimization**
  Run Bayesian Optimization budget 32 on OpenML Task k – embedded in nested CV.

Performance estimates across all evaluated methods are obtained from a fixed outer 10-fold cross-validation loop on each left out dataset. Evaluation of a set of configurations is done using *nested* 5-fold cross-validation. For each configuration in the set, we obtain an estimation of the performance from the nested cross-validation loop. This allows us to select a best configuration from the set, which is then evaluated on the outer test-set. We use either *mlrMBO* or *scikit-optimize* as Bayesian Optimization frameworks.

Datasets, Algorithms and Hyperparameters

We use experimental results available on OpenML (Vanschoren et al. 2014; van Rijn 2016) to evaluate the sets of defaults. In total, we evaluate the proposed method on six algorithms, coming from `mlr` (Bischl et al. 2016) and `scikit-learn` (Pedregosa et al. 2011). We use the 100 datasets from the OpenML100 (Bischl et al. 2017). These contain between 500 and 100,000 observations, up to 5,000 features and are not imbalanced. We evaluate the `mlr` algorithms on the 38 binary class datasets; we evaluate the `scikit-learn` algorithms on all 100 datasets. This decision was made based on the availability of meta-data.

For `mlr`, we evaluate the method on `glmnet` (Friedman, Hastie, and Tibshirani 2010) (elastic net implementation, 2 hyperparameters), `rpart` (Therneau and Atkinson 2018) (decision tree implementation, 4 hyperparameters) and `xgboost` (Chen and Guestrin 2016) (gradient boosting implementation, 10 hyperparameters). The optimization criterion was Area under the ROC curve. We obtained in the order of a million results for randomly selected parameters on the 38 binary datasets.

As of `scikit-learn`, we evaluate the method using Adaboost (5 hyperparameters), SVM (6 hyperparameters) and random forest (6 hyperparameters). The optimization criterion was predictive accuracy. We obtained approximatelly 137.000 results for randomly selected parameters of the 3 algorithms on the 100 datasets. The hyperparameters and their respective ranges are the same as by van Rijn and Hutter (2018).

All following results are obtained by computing defaults using Leave-One-Dataset-Out cross-validation. This means we iteratively learn defaults using $K - 1$ datasets, and evaluate using the held-out dataset. Defaults thus have not been learned from the dataset that they are evaluated on. For any given configuration, we can either obtain an approximation of the risk, by resorting to trained surrogate models, or estimate the true performance using cross-validation.

Exact Discretized vs. Greedy Defaults

We compare the computationally expensive exact discretized approach to the greedy approach in a small-scale experiment, to understand their relative performance in terms of hold-out accuracy. In this experiment, we generate $n = \{1, 2, 3, 4, 5, 6\}$ defaults using both the greedy approach and the exact discretized search, on a subset of the SVM hyperparameter space, for the 100 datasets from the OpenML100. We aim to optimize the gamma ($[2^{-15}, 2^3]$, log-scale) and complexity ($[2^{-5}, 2^{15}]$, log-scale) hyperparameter for the RBF kernel, as van Rijn and Hutter (2018) found these to be most important. We discretized the problem to have 16 choices for both hyperparameters. Note that in order to obtain 6 defaults, already $\binom{256}{6} \approx 3.7 \cdot 10^{11}$ possible options need to be evaluated, over 100 datasets.

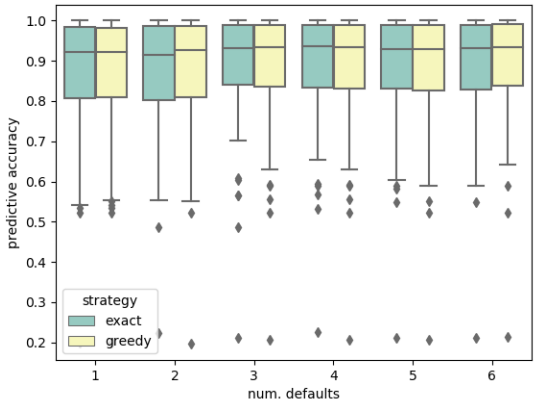


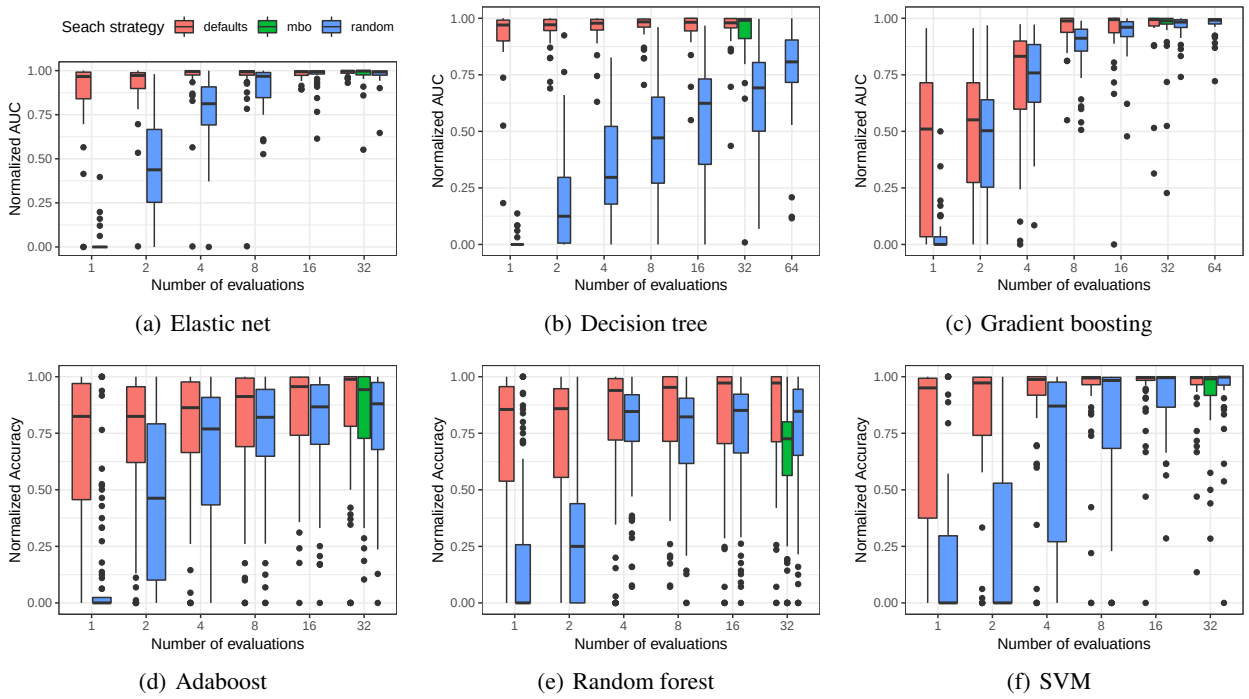
Figure 1: Performance of defaults obtained by exact discretized vs. the greedy.

Figure 1 shows the results. Note that by definition, the results for 1 default should be approximately equal, and can only deviate when a tie is broken in a different way. Furthermore, even though intuitively the exact discretized approach should come up with better set defaults, this might not hold in practice, as the defaults are evaluated on datasets that were not considered when calculating the defaults. The results reveal that the sets of defaults from both strategies perform approximately the same. As the greedy defaults have the benefit of being computationally much cheaper and provide anytime capabilities, we will use the greedy method for the remainder of the paper.

Greedy Defaults

Figure 2 presents the results of the set of defaults obtained by the greedy approach and the baselines. Each subfigure shows the results for a given algorithm. The boxplots represent how the algorithm performed across the 38 (`mlR` algorithms) or 100 (`scikit-learn` algorithms) datasets that it was ran on. Results are normalized to $[0, 1]$ per algorithm and task using the best and worst result on each dataset across all settings.

The results reveal some expected trends. For both the defaults and the random search, having more iterations strictly improves performance. As might be expected, random search with only 1 or 2 iterations does not seem to be a compelling strategy. Bayesian Optimization is often among the highest ranked strategies (which can also be seen in Figure 3). We further observe that using only a few defaults is already competitive with Bayesian Optimization and random search strategies with higher budget. In many



(a) Elastic net (b) Decision tree (c) Gradient boosting

(d) Adaboost (e) Random forest (f) SVM

Figure 2: Boxplots for different algorithms, comparing *sets of defaults*, *random search* and *Bayesian Optimization (mbo)* across several budgets. The y-axis depicts normalized Area under the Curve (upper) and normalized Accuracy (lower) across each learner and task.

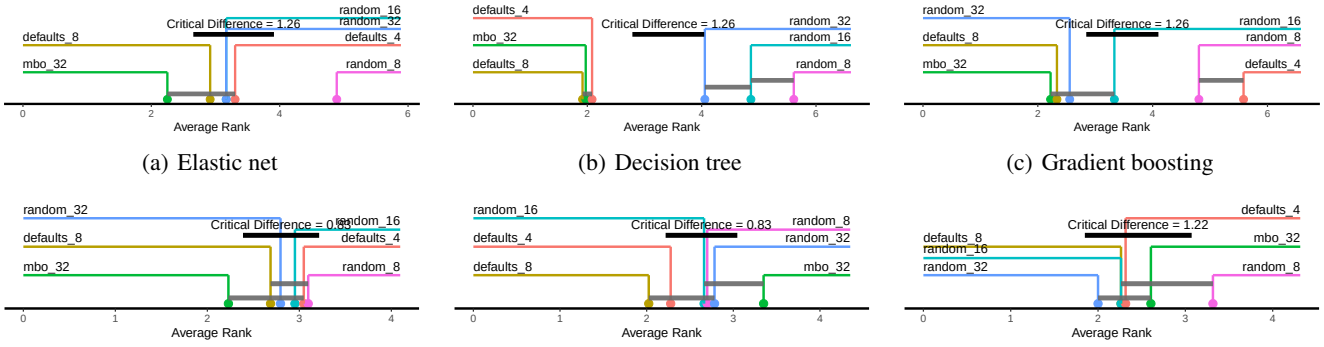
cases, the defaults are competitive with random search procedures that have four to eight times more budget. This is in particular clear for decision trees and elastic net, where 4 defaults already outperform random search significantly. In some cases, e.g., for random forest, advantages of using defaults over random search with multiples of the budgets seem negligible. A reason for this could be, that random forests in general appear more robust with regards to selection of hyperparameters (Probst, Boulesteix, and Bischl 2019), and thus do not profit as much from optimal defaults. We can also see, that random search seems to stagnate much quicker than the set of defaults, which suggests that defaults can still be a viable alternative. It can also be seen, that defaults perform particularly well when the budget is low. When the budget increases, the potential gains decrease. This can be observed in Figures 2 d)-e), where performance only increases marginally after 8 defaults. This can be intuitively understood from the fact, that defaults are learned from a limited number of datasets. Thus when the number of defaults approaches the number of datasets, defaults more and more adapt to a small set of datasets, rather than generalizing to many datasets.

In order to further analyze the results, we perform the Friedman statistical test (with post-hoc Nemenyi test) on the results (Demšar 2006). Again, per classifier and task combination, each strategy gets assigned a rank. The ranks are averaged over all datasets, and reported in Figures 3. If the difference is bigger than the critical distance, there is statistical proof that this difference in performance was not due to random chance. Each pair of strategies that is connected by a gray line, is considered statistically equivalent at $\alpha = 0.05$. We observe the same trends as in the boxplots. Strategies that employ defaults are usually ranked better than random search with 2-4 times the budget, and significantly better then using the same budget. Discrepancies between learners in 2 a) - c) and d)-f) can stem for example from the fact that fewer experimental results were available for the latter, hampering the performance of trained surrogate models.

Numbers of datasets used in the different comparisons arise due to computational constraints. For the evaluation of elastic net, decision trees, the largest two datasets have been excluded from the evaluation, thus allowing for an evaluation on 36 datasets. For Adaboost, random forest and SVM, only datasets where all evaluations finished are included.

Conclusions

We explored the potential of using sets of defaults. Single defaults usually give poor performance, whereas a more complex optimization procedure can be hard to implement and often needs many iterations. Instead, we show that using a sequence of defaults can be a robust and viable alternative to obtain a good hyperparameter configuration. When having access to large amounts of historic results, we can infer short lists of 4-8 candidate configurations that are competitive to random search or Bayesian Optimization with much larger budgets.



(a) Elastic net (b) Decision tree (c) Gradient boosting

(d) Adaboost (e) Random forest (f) SVM

Figure 3: Critical Differences plots comparing 4 and 8 defaults with random search with (8, 16 and 32) and Bayesian Optimization with 32 iterations. The x-axis contains average ranks across all datasets for which all stategies terminated.

Finding the defaults in itself is an NP-hard search problem. We proposed a Mixed Integer Programming solution and a greedy solution, the latter running in polynomial time. Both strategies seem to obtain comparable results, which validates the use of the greedy strategy. An additional benefit of a greedy strategy is its *anytime* performance, which allows selecting an arbitrarily sized, optimal subset.

We performed an extensive evaluation across 6 algorithms, 2 workbenches and 2 performance measures (accuracy and area under the curve). We compared the optimization over sets of defaults against random search with a a much larger budget, and Bayesian Optimization. Experiments showed that defaults consistently outperform the other strategies, even when given less budget.

We note that using sets of defaults is especially worthwhile when either computation time or expertise on hyperparameter optimization is lacking. Especially in the regime of few function evaluations, sets of defaults seem to work particularly well and are statistically equivalent with state-of-the-art techniques. A potential drawback of our method is that our defaults are optimal with respect to a single metric such as accuracy or AUC, and thus might need to be used separately for different evaluation metrics. Identifying whether is actually the case requires further investigation.

However, when fixing the metric, our results can readily be implemented in machine learning software as simple, hard-coded lists of parameters. These will require less knowledge of hyperparameter optimization from the users than current methods, and lead to faster results in many cases.

In future work, we aim to incorporate multi-objective measures for determining the defaults. This includes, but is not limited to computational time, as focusing on defaults that are expected to run fast, might improve the anytime performance even further.

In order to improve performance even more, models trained for a set of defaults can be ensembled. Finally, we aim to combine the sets of defaults with other hyperparameter optimization techniques, e.g., by using it to warm-start Bayesian Optimization procedures. Successfully combining these two paradigms might be the key to convincingly push forward the state-of-the-art for Automated Machine Learning.

Acknowledgements. We would like to thank Michael Hennebry for the discussion leading to the formulation of the 'exact discretized' approach. Furthermore, we would like to thank Fabian Scheipl for constructive criticism of the manuscript. This work has been funded by the German Federal Ministry of Education and Research (BMBF) under Grant No. 01IS18036A. The authors of this work take full responsibilities for its content.

References

[Bergstra and Bengio 2012] Bergstra, J., and Bengio, Y. 2012. Random search for hyper-parameter optimization. *JMLR* 13(Feb):281–305.

[Bergstra et al. 2011] Bergstra, J.; Bardenet, R.; Bengio, Y.; and Kégl, B. 2011. Algorithms for hyper-parameter optimization. In *Proc. of NIPS*. Curran Associates, Inc. 2546–2554.

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

[Bischl et al. 2017] Bischl, B.; Casalicchio, G.; Feurer, M.; Hutter, F.; Lang, M.; Mantovani, R. G.; van Rijn, J. N.; and Vanschoren, J. 2017. OpenML Benchmarking Suites and the OpenML100. *arXiv preprint arXiv:1708.03731v1*.

[Bischl et al. 2018] Bischl, B.; Richter, J.; Bossek, J.; Horn, D.; Thomas, J.; and Lang, M. 2018. mlrMBO: A Modular Framework for Model-Based Optimization of Expensive Black-Box Functions. *arXiv preprint arXiv:1703.03373*.

[Brazdil et al. 2008] Brazdil, P.; Giraud-Carrier, C.; Soares, C.; and Vilalta, R. 2008. *Metalearning: Applications to*

Data Mining. Springer Publishing Company, Incorporated, 1 edition.

[Cawley and Talbot 2010] Cawley, G. C., and Talbot, N. L. 2010. On over-fitting in model selection and subsequent selection bias in performance evaluation. *J. Mach. Learn. Res.* 11:2079–2107.

[Chen and Guestrin 2016] Chen, T., and Guestrin, C. 2016. XGBoost: A scalable tree boosting system. In *Proc. of KDD*, KDD '16, 785–794. New York, NY, USA: ACM.

[Demšar 2006] Demšar, J. 2006. Statistical Comparisons of Classifiers over Multiple Data Sets. *JMLR* 7:1–30.

[Eggensperger et al. 2015] Eggensperger, K.; Hutter, F.; Hoos, H.; and Leyton-Brown, K. 2015. Efficient benchmarking of hyperparameter optimizers via surrogates. In *Proc. AAAI*, 1114–1120.

[Feurer et al. 2015] Feurer, M.; Klein, A.; Eggensperger, K.; Springenberg, J. T.; Blum, M.; and Hutter, F. 2015. Efficient and robust automated machine learning. In *Proc. of NIPS*. Curran Associates, Inc. 2962–2970.

[Feurer et al. 2018] Feurer, M.; Eggensperger, K.; Falkner, S.; Lindauer, M.; and Hutter, F. 2018. Practical automated machine learning for the automl challenge 2018. In *ICML 2018 AutoML Workshop*.

[Feurer, Springenberg, and Hutter 2015] Feurer, M.; Springenberg, J. T.; and Hutter, F. 2015. Initializing bayesian hyperparameter optimization via meta-learning. In *Proc. AAAI*, 1128–1135. AAAI Press.

[Friedman, Hastie, and Tibshirani 2010] Friedman, J.; Hastie, T.; and Tibshirani, R. 2010. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software* 33(1):1–22.

[Gomes et al. 2012] Gomes, T. A. F.; Prudêncio, R. B. C.; Soares, C.; Rossi, A. L. D.; and Carvalho, A. 2012. Combining meta-learning and search techniques to select parameters for support vector machines. *Neurocomputing* 75(1):3–13.

[Hall et al. 2009] Hall, M.; Frank, E.; Holmes, G.; Pfahringer, B.; Reutemann, P.; and Witten, I. H. 2009. The WEKA Data Mining Software: An Update. *ACM SIGKDD explorations newsletter* 11(1):10–18.

[Hodges and Lehmann 1952] Hodges, J. L., and Lehmann, E. L. 1952. The use of previous experience in reaching statistical decisions. *Ann. Math. Statist.* 23(3):396–407.

[Hutter, Hoos, and Leyton-Brown 2011] Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2011. Sequential model-based optimization for general algorithm configuration. In *Proc. of LION*, 507–523. Springer.

[Jamieson and Talwalkar 2016] Jamieson, K., and Talwalkar, A. 2016. Non-stochastic best arm identification and hyperparameter optimization. In *Proc. of AISTATS*, volume 51, 240–248. PMLR.

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

[Leite, Brazdil, and Vanschoren 2012] Leite, R.; Brazdil, P.; and Vanschoren, J. 2012. Selecting Classification Algorithms with Active Testing. In *Proc. of MLDMPR*. Springer. 117–131.

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

[Nemhauser, Wolsey, and Fisher 1978] Nemhauser, G. L.; Wolsey, L. A.; and Fisher, M. L. 1978. An analysis of approximations for maximizing submodular set functions—i. *Mathematical programming* 14(1):265–294.

[Pedregosa et al. 2011] Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; Vanderplas, J.; Passos, A.; Cournapeau, D.; Brucher, M.; Perrot, M.; and Duchesnay, E. 2011. Scikit-learn: Machine learning in Python. *JMLR* 12:2825–2830.

[Pinto, Soares, and Mendes-Moreira 2016] Pinto, F.; Soares, C.; and Mendes-Moreira, J. 2016. Towards automatic generation of metafeatures. In *Proc. of PAKDD*, 215–226. Springer.

[Probst, Boulesteix, and Bischl 2019] Probst, P.; Boulesteix, A.-L.; and Bischl, B. 2019. Tunability: Importance of hyperparameters of machine learning algorithms. *JLMR* 20(53):1–32.

[Snoek, Larochelle, and Adams 2012] Snoek, J.; Larochelle, H.; and Adams, R. P. 2012. Practical bayesian optimization of machine learning algorithms. In *Proc. of NIPS*, NIPS'12, 2951–2959. USA: Curran Associates Inc.

[Therneau and Atkinson 2018] Therneau, T., and Atkinson, B. 2018. *rpart: Recursive Partitioning and Regression Trees*. R package version 4.1-13.

[van Rijn and Hutter 2018] van Rijn, J. N., and Hutter, F. 2018. Hyperparameter importance across datasets. In *Proc. of KDD*, 2367–2376. ACM.

[van Rijn 2016] van Rijn, J. N. 2016. *Massively Collaborative Machine Learning*. Ph.D. Dissertation, Leiden University.

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

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

[Wistuba, Schilling, and Schmidt-Thieme 2015b] Wistuba, M.; Schilling, N.; and Schmidt-Thieme, L. 2015b. Sequential model-free hyperparameter tuning. In *Proc. of ICDM*, 1033–1038.

[Yogatama and Mann 2014] Yogatama, D., and Mann, G. 2014. Efficient Transfer Learning Method for Automatic Hyperparameter Tuning. In *Proc. of AISTATS*, volume 33, 1077–1085. PMLR.