Journal of Machine Learning Research 18 (2018) 1-32 Submitted 7/18; Revised 2/19; Published 3/19

Tunability: Importance of Hyperparameters of Machine Learning Algorithms

Philipp Probst PROBST@IBE.MED.UNI-MUENCHEN.DE
Institute for Medical Information Processing, Biometry and Epidemiology, LMU Munich
Marchioninistr. 15, 81377 München, Germany

Anne-Laure Boulesteix BOULESTEIX@IBE.MED.UNI-MUENCHEN.DE
Institute for Medical Information Processing, Biometry and Epidemiology, LMU Munich
Marchioninistr. 15, 81377 München, Germany

Bernd Bischl BERND.BISCHL@STAT.UNI-MUENCHEN
Department of Statistics, LMU Munich
Ludwigstraße 33, 80539 München, Germany

Editor: Ryan Adams

Abstract

Modern supervised machine learning algorithms involve hyperparameters that have to be set before running them. Options for setting hyperparameters are default values from the software package, manual configuration by the user or configuring them for optimal predictive performance by a tuning procedure. The goal of this paper is two-fold. Firstly, we formalize the problem of tuning from a statistical point of view, define data-based defaults and suggest general measures quantifying the tunability of hyperparameters of algorithms. Secondly, we conduct a large-scale benchmarking study based on 38 datasets from the OpenML platform and six common machine learning algorithms. We apply our measures to assess the tunability of their parameters. Our results yield default values for hyperparameters and enable users to decide whether it is worth conducting a possibly time consuming tuning strategy, to focus on the most important hyperparameters and to choose adequate hyperparameter spaces for tuning.

Keywords: machine learning, supervised learning, classification, hyperparameters, tuning, meta-learning

1. Introduction

Machine learning (ML) algorithms such as gradient boosting, random forest and neural networks for regression and classification involve a number of *hyperparameters* that have to be set before running them. In contrast to direct, first-level model parameters, which are determined during training, these second-level *tuning parameters* often have to be carefully optimized to achieve maximal performance. A related problem exists in many other algorithmic areas, e.g., control parameters in evolutionary algorithms (Eiben and Smit, 2011).

In order to select an appropriate hyperparameter *configuration* for a specific dataset at hand, users of ML algorithms can resort to default values of hyperparameters that are specified in implementing software packages or manually configure them, for example, based on recommendations from the literature, experience or trial-and-error.

Alternatively, one can use hyperparameter *tuning strategies*, which are data-dependent, second-level optimization procedures (Guyon et al., 2010), which try to minimize the expected generalization error of the inducing algorithm over a hyperparameter search space of considered candidate configurations, usually by evaluating predictions on an independent test set, or by running a resampling scheme such as cross-validation (Bischl et al., 2012). For a recent overview of tuning strategies, see, e.g., Luo (2016). These search strategies range from simple grid or random search (Bergstra and Bengio, 2012) to more complex, iterative procedures such as Bayesian optimization (Hutter et al., 2011; Snoek et al., 2012; Bischl et al., 2017b) or iterated F-racing (Birattari et al., 2010; Lang et al., 2017).

In addition to selecting an efficient tuning strategy, the set of tunable hyperparameters and their corresponding ranges, scales and potential prior distributions for subsequent sampling have to be determined by the user. Some hyperparameters might be safely set to default values, if they work well across many different scenarios. Wrong decisions in these areas can inhibit either the quality of the resulting model or at the very least the efficiency and fast convergence of the tuning procedure. This creates a burden for:

1. ML users—Which hyperparameters should be tuned and in which ranges?

2. Designers of ML algorithms—How do I define robust defaults?

We argue that many users, especially if they do not have years of practical experience in the field, here often rely on heuristics or spurious knowledge. It should also be noted that designers of fully automated tuning frameworks face at least very similar problems. It is not clear how these questions should be addressed in a data-dependent, automated, optimal and objective manner. In other words, the scientific community not only misses answers to these questions for many algorithms but also a systematic framework, methods and criteria, which are required to answer these questions.

With the present paper we aim at filling this gap and formalize the problem of parameter tuning from a statistical point of view, in order to simplify the tuning process for less experienced users and to optimize decision making for more advanced processes.

After presenting related literature in Section 2, we define theoretical measures for assessing the impact of tuning in Section 3. For this purpose we (i) define the concept of *default hyperparameters*, (ii) suggest measures for quantifiying the tunability of the whole algorithm and specific hyperparameters based on the differences between the performance of default hyperparameters and the performance of the hyperparameters when this hyperparameter is set to an optimal value. Then we (iii) address the tunability of hyperparameter combinations and joint gains, (iv) provide theoretical definitions for an appropriate hyperparameter space on which tuning should be executed and (v) propose procedures to estimate these quantities based on the results of a benchmark study with random hyperparameter configurations with the help of surrogate models. In sections 4 and 5 we illustrate these concepts and methods through an application. For this purpose we use benchmark results of six machine learning algorithms with different hyperparameters which were evaluated on 38 datasets from the OpenML platform. Finally, in the last Section 6 we conclude and discuss the results.

2. Related Literature

To the best of our knowledge, only a limited amount of articles address the problem of tunability and generation of tuning search spaces. Bergstra and Bengio (2012) compute the relevance of the hyperparameters of neural networks and conclude that some are important on all datasets, while others are only important on some datasets. Their conclusion is primarily visual and used as an argument for why random search works better than grid search when tuning neural networks.

A specific study for decision trees was conducted by Mantovani et al. (2016) who apply standard tuning techniques to decision trees on 102 datasets and calculate differences of accuracy between the tuned algorithm and the algorithm with default hyperparameter settings.

A different approach is proposed by Hutter et al. (2013), which aims at identifying the most important hyperparameters via forward selection. In the same vein, Fawcett and Hoos (2016) present an *ablation analysis* technique, which aims at identifying the hyperparameters that contribute the most to improved performance after tuning. For each of the considered hyperparameters, they compute the performance gain that can be achieved by changing its value from the initial value to the value specified in the target configuration which was determined by the tuning strategy. This procedure is iterated in a greedy forward search.

A more general framework for measuring the importance of single hyperparameters is presented by Hutter et al. (2014). After having used a tuning strategy such as sequential model-based optimization, a functional ANOVA approach is used for measuring the importance of hyperparameters.

These works concentrate on the importance of hyperparameters on single datasets, mainly to retrospectively explain what happened during an already concluded tuning process. Our main focus is the generalization across multiple datasets in order to facilitate better general understanding of hyperparameter effects and better decision making for future experiments. In a recent paper van Rijn and Hutter (2017) pose very similar questions to ours to assess the importance of hyperparameters across datasets. We compare it to our approach in Section 6.

Our framework is based on using surrogate models, also sometimes called empirical performance models, which allow estimating the performance of arbitrary hyperparameter configurations based on a limited number of prior experiments. The idea of surrogate models is far from new (Audet et al., 2000), as it constitutes the central idea of Bayesian optimization for hyperparameter search but is also used, for example, in Biedenkapp et al. (2017) for increasing the speed of an ablation analysis and by Eggensperger et al. (2018) for speeding up the benchmarking of tuning strategies.

3. Methods for Estimation of Defaults, Tunability and Ranges

In this section we introduce theoretical definitions for defaults, tunability and tuning ranges, then describe how to estimate them and finally discuss the topic of reparametrization.

3.1. General Notation

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 $\mathcal{T}$ of n observations. A machine learning (ML) algorithm now learns the functional relationship between X and Y by producing a prediction model $\hat{f}(X, \theta)$, controlled by the k-dimensional hyperparameter configuration $\theta = (\theta_1, ..., \theta_k)$ from the hyperparameter search space $\Theta = \Theta_1 \times ... \times \Theta_k$. In order to measure prediction performance pointwise between the true label Y and its prediction $\hat{f}(X, \theta)$, we define a loss function $L(Y, \hat{f}(X, \theta))$. We are naturally interested in estimating the expected risk of the inducing algorithm, w.r.t. θ on new data, also sampled from $\mathcal{P}$: $R(\theta) = E(L(Y, \hat{f}(X, \theta))|\mathcal{P})$. This mapping encodes, given a certain data distribution, a certain learning algorithm and a certain performance measure, the numerical quality for any hyperparameter configuration θ. Given m different datasets (or data distributions) $\mathcal{P}_1, ..., \mathcal{P}_m$, we arrive at m hyperparameter risk mappings

$$R^{(j)}(\theta) := E(L(Y, \hat{f}(X, \theta))|\mathcal{P}_j), \qquad j = 1, ..., m. \tag{1}$$

For now, we assume all $R^{(j)}(\theta)$ to be known, and show how to estimate them in Section 3.7.

3.2. Optimal Configuration per Dataset and Optimal Defaults

We first define the best hyperparameter configuration for dataset j as

$$\theta^{(j)\star} := \arg\min_{\theta \in \Theta} R^{(j)}(\theta). \tag{2}$$

Defaults settings are supposed to work well across many different datasets and are usually provided by software packages, in an often ad hoc or heuristic manner. We propose to define an optimal default configuration, based on an extensive number of empirical experiments on m different benchmark datasets, by

$$\theta^{\star} := \arg\min_{\theta \in \Theta} g(R^{(1)}(\theta), ..., R^{(m)}(\theta)). \tag{3}$$

Here, g is a summary function that has to be specified. Selecting the mean (or median as a more robust candidate) would imply minimizing the average (or median) risk over all datasets.

The measures $R^{(j)}(\theta)$ could potentially be scaled appropriately beforehand in order to make them more commensurable between datasets, e.g., one could scale all $R^{(j)}(\theta)$ to $[0, 1]$ by subtracting the result of a very simple baseline like a featureless dummy predictor and dividing this difference by the absolute difference between the risk of the best possible result (as an approximation of the Bayes error) and the result of the very simple baseline predictor. Or one could produce a statistical z-score by subtracting the mean and dividing by the standard deviation from all experimental results on the same dataset (Feurer et al., 2018).

The appropriateness of the scaling highly depends on the performance measure that is used. One could, for example, argue that the AUC does not have to be scaled by using the probabilistic interpretation of the AUC. Given a randomly chosen observation x belonging to class 1, and a randomly chosen observation x' belonging to class 0, the AUC is the probability that the evaluated classification algorithm will assign a higher score to x than to x'. Thus,

an improvement from 0.5 to 0.6 on one dataset could be seen as equally important to an improvement from 0.8 to 0.9 on another dataset. On the other hand, averaging the mean squared error on several datasets does not make a lot of sense, as the scale of the outcome of different regression problems can be very different. Then scaling or using another measure such as the R^2 is reasonable. As our main risk measure is the AUC, we do not use any scaling.[1]

3.3. Measuring Overall Tunability of a ML Algorithm

A general measure of the tunability of an algorithm per dataset can then be computed based on the difference between the risk of an overall reference configuration (e.g., either the software defaults or definition (3)) and the risk of the best possible configuration on that dataset:

$$d^{(j)} := R^{(j)}(\theta^\star) - R^{(j)}(\theta^{(j)\star}), \text{ for } j = 1, ..., m. \tag{4}$$

For each algorithm, this gives rise to an empirical distribution of performance differences over datasets, which might be directly visualized or summarized to an aggregated tunability measure d by using mean, median or quantiles.

3.4. Measuring Tunability of a Specific Hyperparameter

The best hyperparameter value for one parameter i on dataset j, when all other parameters are set to defaults from $\theta^\star := (\theta_1^\star, ..., \theta_k^\star)$, is denoted by

$$\theta_i^{(j)\star} := \underset{\theta \in \Theta, \theta_l = \theta_l^\star \forall l \neq i}{\arg\min} R^{(j)}(\theta). \tag{5}$$

A natural measure for tunability of the i-th parameter on dataset j is then the difference in risk between the above and our default reference configuration:

$$d_i^{(j)} := R^{(j)}(\theta^\star) - R^{(j)}(\theta_i^{(j)\star}), \text{ for } j = 1, ..., m, i = 1, ..., k. \tag{6}$$

Furthermore, we define $d_i^{(j),\text{rel}} = \frac{d_i^{(j)}}{d^{(j)}}$ as the fraction of performance gain, when we only tune parameter i compared to tuning the complete algorithm, on dataset j. Again, one can calculate the mean, the median or quantiles of these two differences over the n datasets, to get a notion of the overall tunability d_i of this parameter.

3.5. Tunability of Hyperparamater Combinations and Joint Gains

Let us now consider two hyperparameters indexed as i_1 and i_2. To measure the tunability with respect to these two parameters, we define

$$\theta_{i_1,i_2}^{(j)\star} := \underset{\theta \in \Theta, \theta_l = \theta_l^\star \forall l \notin \{i_1,i_2\}}{\arg\min} R^{(j)}(\theta), \tag{7}$$

1. We also tried out normalization (z-score) and got qualitatively similar results to the non-normalized results presented in Section 5.

i.e., the θ-vector containing the default values for all hyperparameters other than i_1 and i_2, and the optimal combination of values for the i_1-th and i_2-th components of θ.

Analogously to the previous section, we can now define the tunability of the set (i_1, i_2) as the gain over the reference default on dataset j as

$$d^{(j)}_{i_1,i_2} := R^{(j)}(\theta^*) - R^{(j)}(\theta^{(j)\star}_{i_1,i_2}). \tag{8}$$

The joint gain which can be expected when tuning not only one of the two hyperparameters individually, but both of them jointly, on a dataset j, can be expressed by

$$g^{(j)}_{i_1,i_2} := \min\{(R^{(j)}(\theta^{(j)\star}_{i_1})), (R^{(j)}(\theta^{(j)\star}_{i_2}))\} - R^{(j)}(\theta^{(j)\star}_{i_1,i_2}). \tag{9}$$

Furthermore, one could be interested in whether this joint gain could simply be reached by tuning both parameters i_1 and i_2 in a univariate fashion sequentially, either in the order $i_1 \to i_2$ or $i_2 \to i_1$, and what order would be preferable. For this purpose one could compare the risk of the hyperparameter value that results when tuning them together $R^{(j)}(\theta^{(j)\star}_{i_1,i_2})$ with the risks of the hyperparameter values that are obtained when tuning them sequentially, that means $R^{(j)}(\theta^{(j)\star}_{i_1 \to i_2})$ or $R^{(j)}(\theta^{(j)\star}_{i_2 \to i_1})$, which is done for example in Waldron et al. (2011).

Again, all these measures should be summarized across datasets, resulting in d_{i_1,i_2} and g_{i_1,i_2}. Of course, these approaches can be further generalized by considering combinations of more than two parameters.

3.6. Optimal Hyperparameter Ranges for Tuning

A reasonable hyperparameter space $\Theta^\star$ for tuning should include the optimal configuration $\theta^{(j)\star}$ for dataset j with high probability. We denote the p-quantile of the distribution of one parameter regarding the best hyperparameters on each dataset $(\theta^{(1)\star})_i, ..., (\theta^{(m)\star})_i$ as $q_{i,p}$. The hyperparameter tuning space can then be defined as

$$\Theta^\star := \left\{\theta \in \Theta | \forall i \in \{1, ..., k\} : \theta_i \geq q_{i,p_1} \wedge \theta_i \leq q_{i,p_2}\right\}, \tag{10}$$

with p_1 and p_2 being quantiles which can be set for example to the 5 % quantile and the 95 % quantile. This avoids focusing too much on outlier datasets and makes the definition of the space independent from the number of datasets.

The definition above is only valid for numerical hyperparameters. In case of categorical variables one could use similar rules, for example only including hyperparameter values that were at least once or in at least 10 % of the datasets the best possible hyperparameter setting.

3.7. Practical Estimation

In order to practically apply the previously defined concepts, two remaining issues need to be addressed: a) We need to discuss how to obtain $R^{(j)}(\theta)$; and b) in (2) and (3) a multivariate optimization problem (the minimization) needs to be solved.[2]

2. All other previous optimization problems are univariate or two-dimensional and can simply be addressed by simple techniques such as a fine grid search.

For a) we estimate $R^{(j)}(\theta)$ by using surrogate models $\hat{R}^{(j)}(\theta)$, and replace the original quantity by its estimator in all previous formulas. Surrogate models for each dataset j are based on a meta dataset. This is created by evaluating a large number of configurations of the respective ML method. The surrogate regression model then learns to map a hyperparameter configuration to estimated performance. For b) we solve the optimization problem—now cheap to evaluate, because of the surrogate models—through black-box optimization.

3.8. Reparametrization

All tunability measures mentioned above can possibly depend on and be influenced by a reparametrization of hyperparameters. For example, in the case of the elastic net the parameters λ and α could be reparametrized as $\lambda_1 = \alpha\lambda$ and $\lambda_2 = (1-\alpha)\lambda$. Formally, such a reparametrization could be defined as a (bijective) function $\phi : \Theta \to \tilde{\Theta}$, such that $\phi(\theta)$ maps an original configuration θ to a new representation $\tilde{\theta} = \phi(\theta)$ from $\tilde{\Theta}$, in a one-to-one manner. Then defaults (calculated by the approach in Section 3.2) are naturally transformed via $\tilde{\theta}^\star = \phi(\theta^\star)$ into the new space $\tilde{\Theta}$, but will stay logically the same. Moreover, the general tunability of the algorithm does (obviously) not change. Depending on the parameters that are involved in the reparametrization, the tunability of the parameters can change. If, for example, only one parameter is involved, all tunabilities remain the same. If two or more parameters are involved, the single tunabilities of the parameters could change but the tunability of the set of the transformed parameters remains the same.

One might define a reparametrization as ideal (in the sense of simplified tuning) if the tunability is concentrated on one (or only few) hyperparameter(s), so that only this parameter has to be optimized and all remaining hyperparameters can remain at their (optimal) default values, reducing a multivariate optimization problem to a 1-dimensional or at least lower dimensional one. Using the definition above, this would imply that the joint gain of the new parameter(s) is (close to) 0. For example, imagine that the optimal hyperparameter values per dataset of two hyperparameters θ_1 and θ_2 lie on the line of equation $\theta_1 = \theta_2$. A useful reparametrization would then be $\tilde{\theta}_1 = \theta_1 + \theta_2$ and the orthogonal $\tilde{\theta}_2 = \theta_1 - \theta_2$. It would then only be necessary to tune $\tilde{\theta}_1$, while $\tilde{\theta}_2$ would be set to the default value 0.

A more general formulation is possible if we use the definition of 3.4. We could, for example, search for a bijective and invertible function $\phi^\star(.)$, across a certain parameterized function space, such that the mean tunability is concentrated on and therefore maximal for the first parameter and minimal for the other parameters, i.e.:

$$\phi^\star := \arg\min_{\phi \in \Phi} \frac{1}{m} \sum_{j=1}^{m} \min_{\tilde{\theta} \in \tilde{\Theta}, \tilde{\theta}_l = \tilde{\theta}_l^\star \forall l \neq 1} R^{(j)} \left(\phi^{-1} \left(\tilde{\theta} \right) \right). \tag{11}$$

We could select a restricted function space for ϕ, e.g., restrict ourselves to the space of all linear (invertible) transformations $\{\phi : \mathbb{R}^k \to \mathbb{R}^k | \phi(x) = Ax, A \in \mathbb{R}^{k \times k}, \det(A) \neq 0\}$. If concentrating the whole tunability on only one parameter is not possible, we could try a similar approach by concentrating it on a combination of two hyperparameters.

Note that such a reparametrization is not always helpful. For example, imagine we have two binary parameters and transform them such that (i) one of them has 4 levels that correspond to all possible combinations of these two parameters and (ii) the other parameter is set to a fixed constant. This reparametrization would not be useful: all the tunability

is contained in the first parameter, but there is no real advantage, as still four evaluations have to be executed in the tuning process to get the best hyperparameter combination.

Finally, note that it can also be useful to reparametrize a single hyperparameter for the purpose of tuning. Imagine, for example, that most of the optimal parameters on the different datasets are rather small and only a few are large. A transformation of this parameter such as a log-transformation may then be useful. This is very similar to using prior probabilities for tuning (based on results on previous datasets) which could be seen as a useful alternative to a reparametrization and which is already proposed in van Rijn and Hutter (2017).

4. Experimental Setup

In this section we give an overview about the experimental setup that is used for obtaining surrogate models, tunability measures and tuning spaces.

4.1. Datasets from the OpenML Platform

Recently, the OpenML project (Vanschoren et al., 2013) has been created as a flexible online platform that allows ML scientists to share their data, corresponding tasks and results of different ML algorithms. We use a specific subset of carefully curated classification datasets from the OpenML platform called *OpenML100* (Bischl et al., 2017a). For our study we only use the 38 binary classification tasks that do not contain any missing values.

4.2. ML Algorithms

The algorithms considered in this paper are common methods for supervised learning. We examine elastic net (`glmnet` R package), decision tree (`rpart`), k-nearest neighbors (`kknn`), support vector machine (`svm`), random forest (`ranger`) and gradient boosting (`xgboost`). For more details about the used software packages see Kühn et al. (2018b). An overview of their considered hyperparameters is displayed in Table 1, including respective data types, box-constraints and a potential transformation function.

In the case of `xgboost`, the underlying package only supports numerical features, so we opted for a dummy feature encoding for categorical features, which is performed internally by the underlying packages for `svm` and `glmnet`.

Some hyperparameters of the algorithms are dependent on others. We take into account these dependencies and, for example, only sample a value for `gamma` for the support vector machine if the radial kernel was sampled beforehand.

4.3. Performance estimation

Several measures are regarded throughout this paper, either for evaluating our considered classification models that should be tuned, or for evaluating our surrogate regression models. As no optimal measure exists, we will compare several of them. In the classification case, we consider AUC, accuracy and Brier score. In the case of surrogate regression, we consider R^2, which is directly proportional to the regular mean squared error but scaled to [0,1] and explains the gain over a constant model estimating the overall mean of all data points. We also compute Kendall's tau as a ranking based measure for regression.

Algorithm	Hyperparameter	Type	Lower	Upper	Trafo
glmnet					
(Elastic net)	alpha	numeric	0	1	-
	lambda	numeric	-10	10	2^x
rpart					
(Decision tree)	cp	numeric	0	1	-
	maxdepth	integer	1	30	-
	minbucket	integer	1	60	-
	minsplit	integer	1	60	-
kknn	-	-			
(k-nearest neighbor)	k	integer	1	30	-
svm					
(Support vector machine)	kernel	discrete	-	-	-
	cost	numeric	-10	10	2^x
	gamma	numeric	-10	10	2^x
	degree	integer	2	5	-
ranger					
(Random forest)	num.trees	integer	1	2000	-
	replace	logical	-	-	-
	sample.fraction	numeric	0.1	1	-
	mtry	numeric	0	1	$x \cdot p$
	respect.unordered.factors	logical	-	-	-
	min.node.size	numeric	0	1	n^x
xgboost					
(Gradient boosting)	nrounds	integer	1	5000	-
	eta	numeric	-10	0	2^x
	subsample	numeric	0.1	1	-
	booster	discrete	-	-	-
	max_depth	integer	1	15	-
	min_child_weight	numeric	0	7	2^x
	colsample_bytree	numeric	0	1	-
	colsample_bylevel	numeric	0	1	-
	lambda	numeric	-10	10	2^x
	alpha	numeric	-10	10	2^x

Table 1: Hyperparameters of the algorithms. p refers to the number of variables and n to the number of observations. The columns *Lower* and *Upper* indicate the regions from which samples of these hyperparameters are drawn. The transformation function in the trafo column, if any, indicates how the values are transformed according to this function. The exponential transformation is applied to obtain more candidate values in regions with smaller hyperparameters because for these hyperparameters the performance differences between smaller values are potentially bigger than for bigger values. The `mtry` value in `ranger` that is drawn from $[0, 1]$ is transformed for each dataset separately. After having chosen the dataset, the value is multiplied by the number of variables and afterwards rounded up. Similarly, for the `min.node.size` the value x is transformed by the formula $[n^x]$ with n being the number of observations of the dataset, to obtain a positive integer values with higher probability for smaller values (the value is finally rounded to obtain integer values).

The performance estimation for the different hyperparameter experiments is computed through 10-fold cross-validation. For the comparison of surrogate models 10 times repeated 10-fold cross-validation is used.

4.4. Random Bot sampling strategy for meta data

To reliably estimate our surrogate models we need enough evaluated configurations per classifier and dataset. We sample these points from independent uniform distributions where the respective support for each parameter is displayed in Table 1. Here, *uniform* refers to the untransformed scale, so we sample uniformly from the interval [*Lower*, *Upper*] of Table 1.

In order to properly facilitate the automatic computation of a large database of hyperparameter experiments, we implemented a so called OpenML bot. In an embarrassingly parallel manner it chooses in each iteration a random dataset, a random classification algorithm, samples a random configuration and evaluates it via cross-validation. A subset of 500000 experiments for each algorithm and all datasets are used for our analysis here.[3] More technical details regarding the random bot, its setup and results can be obtained in Kühn et al. (2018b), furthermore, for simple and permanent access the results of the bot are stored in a figshare repository (Kühn et al., 2018a).

4.5. Optimizing Surrogates to Obtain Optimal Defaults

Random search is also used for our black-box optimization problems in Section 3.7. For the estimation of the defaults for each algorithm we randomly sample 100000 points in the hyperparameter space as defined in Table 1 and determine the configuration with the minimal average risk. The same strategy with 100000 random points is used to obtain the best hyperparameter setting on each dataset that is needed for the estimation of the tunability of an algorithm. For the estimation of the tunability of single hyperparameters we also use 100000 random points for each parameter, while for the tunability of combination of hyperparameters we only use 10000 random points to reduce runtime as this should be enough to cover 2-dimensional hyperparameter spaces.

Of course one has to be careful with overfitting here, as our optimal defaults are chosen with the help of the same datasets that are used to determine the performance. Therefore, we also evaluate our approach via a "10-fold cross-validation across datasets". Here, we repeatedly calculate the optimal defaults based on 90% "training datasets" and evaluate the package defaults and our optimal defaults—the latter induced from the training datasets—on the surrogate models of the remaining 10% "test datasets", and compare their difference in performance.

4.6. The Problem of Hyperparameter Dependency

Some parameters are dependent on other superordinate hyperparameters and are only relevant if the parameter value of this superordinate parameter was set to a specific value. For example `gamma` in `svm` only makes sense if the `kernel` was set to "radial" or `degree` only makes sense if the kernel was set to "polynomial". Some of these subordinate parameters might be invalid/inactive in the reference default configuration, rendering it impossible to

3. Only 30 experiments are used for each dataset for `kknn`, because we only consider the parameter k.

univariately tune them in order to compute their tunability score. In such a case we set the superordinate parameter to a value which makes the subordinate parameter active, compute the optimal defaults for the rest of the parameters and compute the tunability score for the subordinate parameter with these defaults.

4.7. Software Details

All our experiments are executed in R and are run through a combination of custom code from our random bot (Kühn et al., 2018b), the `OpenML` R package (Casalicchio et al., 2017), `mlr` (Bischl et al., 2016) and `batchtools` (Lang et al., 2017) for parallelization. All results are uploaded to the OpenML platform and there publicly available for further analysis. `mlr` is also used to compare and fit all surrogate regression models. The fully reproducible R code for all computations and analyses of our paper can be found on the github page: `https://github.com/PhilippPro/tunability`. We also provide an interactive shiny app under `https://philipppro.shinyapps.io/tunability/`, which displays all results of the following section in a potentially more convenient, interactive fashion and which can simply be accessed through a web browser.

5. Results and Discussion

We calculate all results for AUC, accuracy and Brier score but mainly discuss AUC results here. Tables and figures for the other measures can be accessed in the appendix and in our interactive shiny application.

5.1. Surrogate Models

We compare different possible regression models as candidates for our surrogate models: the linear model (`lm`), a simple decision tree (`rpart`), k nearest-neighbors (`kknn`) and random forest (`ranger`)[4] All algorithms are run with their default settings. We calculate 10 times repeated 10-fold cross-validated regression performance measures R^2 and Kendall's tau per dataset, and average these across all datasets.[5] Results for AUC are displayed in Figure 1. A good overall performance is achieved by `ranger` with qualitatively similar results for other classification performance measures (see Appendix). In the following we use random forest as surrogate model because it performs reasonably well and is already an established algorithm for surrogate models in the literature (Eggensperger et al., 2014; Hutter et al., 2013).

5.2. Optimal Defaults and Tunability

Table 2 displays our mean tunability results for the algorithms as defined in formula (4) w.r.t. package defaults (`Tun.P` column) and our optimal defaults (`Tun.O`). The distribution of the tunability values of the optimal defaults can be seen in Figure 2 in the modified

4. We also tried `cubist` (Kuhn et al., 2016), which provided good results but the algorithm had some technical problems for some combinations of datasets and algorithms. We did not include gaussian process which is one of the standard algorithms for surrogate models as it cannot handle categorical variables.
5. In case of `kknn` four datasets did not provide results for one of the surrogate models and were not used.

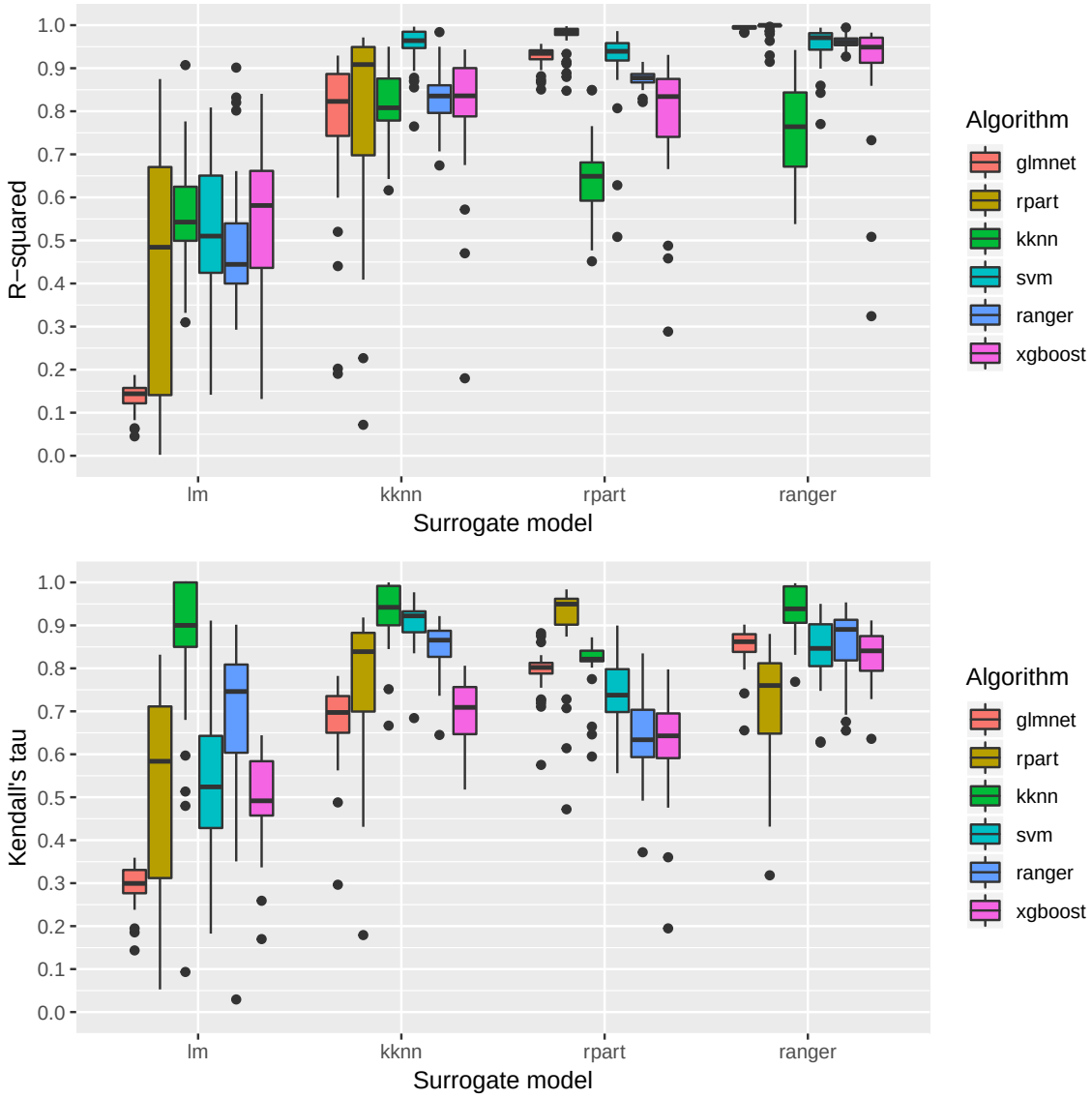

Figure 1: Average performances over the datasets of different surrogate models (target: AUC) for different algorithms (that were presented in 4.2). For an easier comparison of the surrogate models the same graph with exchanged x-axis and legend is available in the appendix in Figure 5.

Algorithm	Tun.P	Tun.O	Tun.O-CV	Improv	Impr-CV
glmnet	0.069 ± 0.019	0.024 ± 0.013	0.037 ± 0.015	0.045 ± 0.015	0.032 ± 0.015
rpart	0.038 ± 0.006	0.012 ± 0.004	0.016 ± 0.004	0.025 ± 0.006	0.022 ± 0.006
kknn	0.031 ± 0.006	0.006 ± 0.004	0.006 ± 0.004	0.025 ± 0.008	0.025 ± 0.008
svm	0.056 ± 0.011	0.042 ± 0.007	0.048 ± 0.008	0.014 ± 0.005	0.008 ± 0.007
ranger	0.010 ± 0.003	0.006 ± 0.001	0.007 ± 0.001	0.004 ± 0.003	0.003 ± 0.003
xgboost	0.043 ± 0.006	0.014 ± 0.006	0.017 ± 0.007	0.029 ± 0.003	0.026 ± 0.003

Table 2: Mean tunability (regarding AUC) with the package defaults (Tun.P) and the optimal defaults (Tun.O) as reference, cross-validated tunability (Tun.O-CV), average improvement (Improv) and cross-validated average improvement (Impr-CV) obtained by using optimal defaults compared to package defaults. The (cross-validated) improvement can be calculated by the (rounded) difference between Tun.P and Tun.O (Tun.O-CV). Standard error of the mean (SEM) is given behind the "±"-sign.

boxplots. Table 2 also displays the average improvement per algorithm when moving from package defaults to optimal defaults (`Improv`), which was positive overall. This also holds for `svm` and `ranger` although the package defaults are data dependent, which we currently cannot model (`gamma` $= 1/p$ for `svm` and `mtry` $= \sqrt{p}$ for `ranger`). As our optimal defaults are calculated based on all datasets, there is a risk of overfitting. So we perform a 5-fold cross-validation on dataset level, always calculating optimal defaults on $\frac{4}{5}$ of datasets and evaluating them on $\frac{1}{5}$ of the datasets. The results in the column `Impr-CV` in Table 2 show that the improvement compared to the package defaults is less pronounced but still positive for all algorithms.

From now on, when discussing tunability, we will only do this w.r.t. our optimal defaults.

Clearly, some algorithms such as `glmnet` and `svm` are much more tunable than the others, while `ranger` is the algorithm with the smallest tunability, which is in line with common knowledge in the web community. In the boxplots in Figure 2 for each ML algorithm, some values that are much bigger than the others are visible, which indicates that tuning has a much higher impact on some specific datasets.

5.3. Tunability of Specific Hyperparameters

In Table 3 the mean tunability (regarding the AUC) of single hyperparameters as defined in Equation (6) in Section 3.4 can be seen. Moreover, in Figure 3 the distributions of the tunability values of the hyperparameters are depicted in boxplots, which makes it possible to detect outliers and to examine skewness. The same results for the Brier score and accuracy can be found in the appendix. In the following analysis of our results, we will refer to tunability only with respect to optimal defaults.

For `glmnet` `lambda` seems to be more tunable than `alpha` regarding the AUC, especially for two datasets tuning seems to be crucial. For accuracy we observe the same pattern, while for Brier score `alpha` seems to be more tunable than `lambda` (see Figure 11 and Figure 13 in the appendix). We could not find any recommendation in the literature for

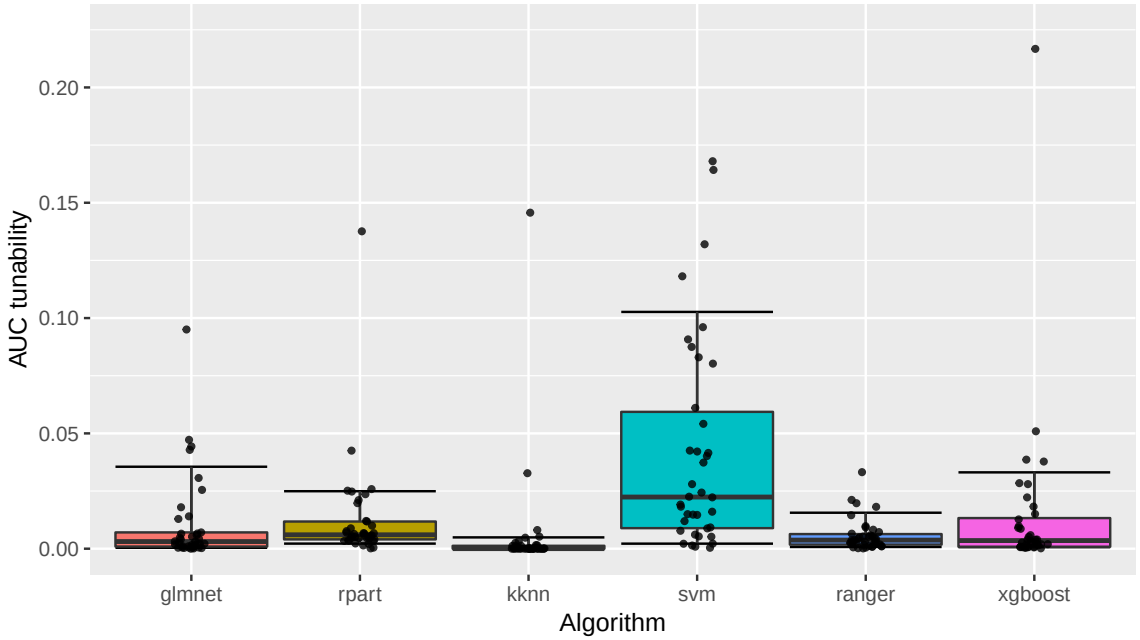

Figure 2: Boxplots of the tunabilities (AUC) of the different algorithms with respect to optimal defaults. The upper and lower whiskers (upper and lower line of the boxplot rectangle) are in our case defined as the 0.1 and 0.9 quantiles of the tunability scores. The 0.9 quantile indicates how much performance improvement can be expected on at least 10% of datasets. One outlier of `glmnet` (value 0.5) is not shown.

these two parameters. In `rpart` the `minbucket` and `minsplit` parameters seem to be the most important ones for tuning which is in line with the analysis of Mantovani et al. (2018). `k` in the `kknn` algorithm is very tunable w.r.t. package defaults, but not regarding optimal defaults. Note that the optimal default is 30 which is at the boundary of possible values, so possibly bigger values can provide further improvements. A classical suggestion in the literature (Lall and Sharma, 1996) is to use $\sqrt{n}$ as default value. This is in line with our results, as the number of observations is bigger than 900 in most of our datasets.

In `svm` the biggest gain in performance can be achieved by tuning the `kernel`, `gamma` or `degree`, while the `cost` parameter does not seem to be very tunable. To the best of our knowledge, this has not been noted in the literature yet. In `ranger` `mtry` is the most tunable parameter which is already common knowledge and is implemented in software packages such as `caret` (Kuhn, 2008). For `xgboost` there are two parameters that are quite tunable: `eta` and `booster`. The tunability of `booster` is highly influenced by an outlier as can be seen in Figure 3. The 5-fold cross-validated results can be seen in Table 10 of the appendix: they are quite similar to the non cross-validated results and for all parameters slightly higher.

Parameter	Def.P	Def.O	Tun.P	Tun.O	$q_{0.05}$	$q_{0.95}$
glmnet			0.069	0.024		
alpha	1	0.403	0.038	0.006	0.009	0.981
lambda	0	0.004	0.034	0.021	0.001	0.147
rpart			0.038	0.012		
cp	0.01	0	0.025	0.002	0	0.008
maxdepth	30	21	0.004	0.002	12.1	27
minbucket	7	12	0.005	0.006	3.85	41.6
minsplit	20	24	0.004	0.004	5	49.15
kknn			0.031	0.006		
k	7	30	0.031	0.006	9.95	30
svm			0.056	0.042		
kernel	radial	radial	0.030	0.024		
cost	1	682.478	0.016	0.006	0.002	920.582
gamma	$1/p$	0.005	0.030	0.022	0.003	18.195
degree	3	3	0.008	0.014	2	4
ranger			0.010	0.006		
num.trees	500	983	0.001	0.001	206.35	1740.15
replace	TRUE	FALSE	0.002	0.001		
sample.fraction	1	0.703	0.004	0.002	0.323	0.974
mtry	$\sqrt{p}$	$p \cdot 0.257$	0.006	0.003	0.035	0.692
respect.unordered.factors	TRUE	FALSE	0.000	0.000		
min.node.size	1	1	0.001	0.001	0.007	0.513
xgboost			0.043	0.014		
nrounds	500	4168	0.004	0.002	920.7	4550.95
eta	0.3	0.018	0.006	0.005	0.002	0.355
subsample	1	0.839	0.004	0.002	0.545	0.958
booster	gbtree	gbtree	0.015	0.008		
max_depth	6	13	0.001	0.001	5.6	14
min_child_weight	1	2.06	0.008	0.002	1.295	6.984
colsample_bytree	1	0.752	0.006	0.001	0.419	0.864
colsample_bylevel	1	0.585	0.008	0.001	0.335	0.886
lambda	1	0.982	0.003	0.002	0.008	29.755
alpha	1	1.113	0.003	0.002	0.002	6.105

Table 3: Defaults (package defaults (Def.P) and optimal defaults (Def.O)), tunability of the hyperparameters with the package defaults (Tun.P) and our optimal defaults (Tun.O) as reference and tuning space quantiles ($q_{0.05}$ and $q_{0.95}$) for different parameters of the algorithms.

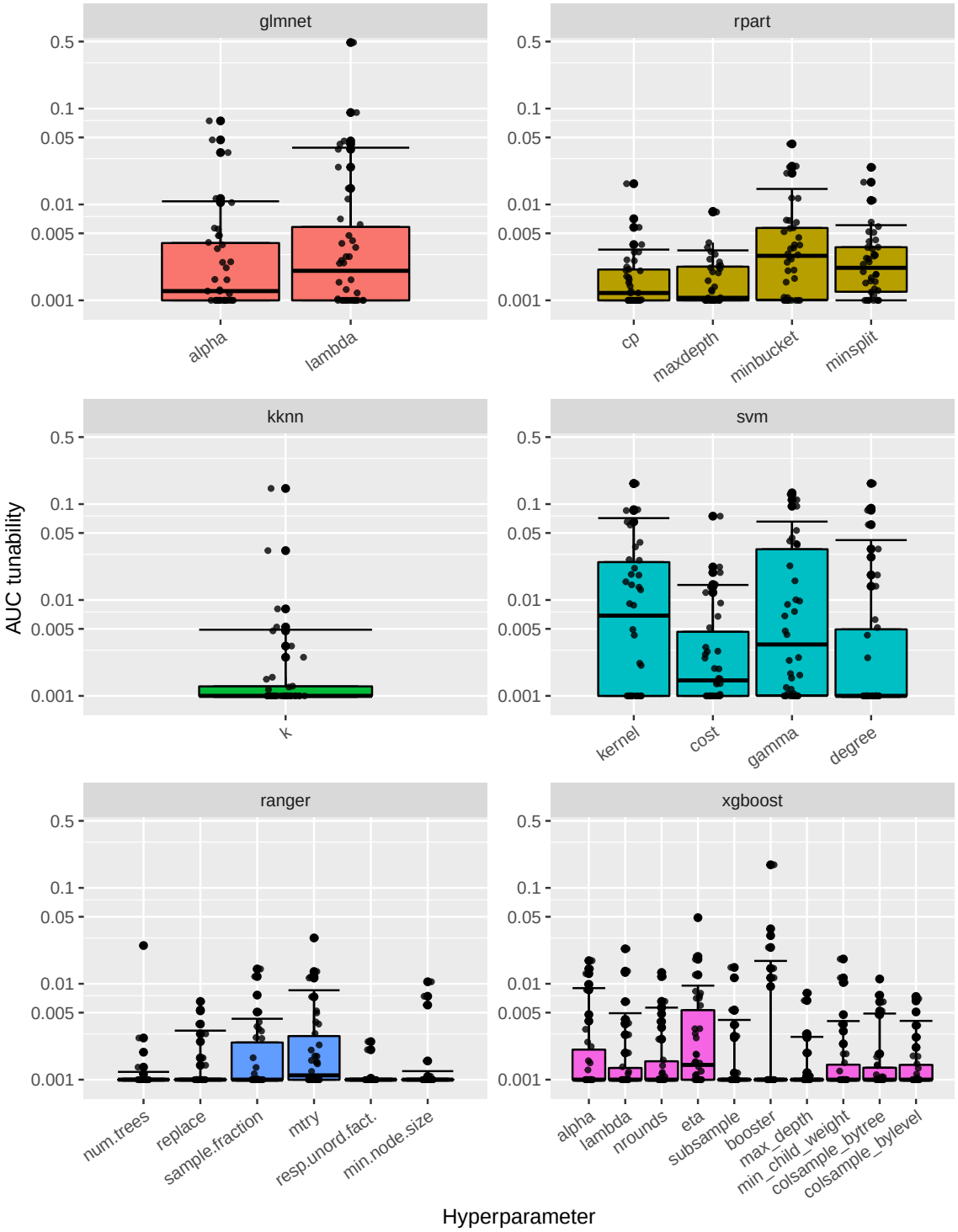

Figure 3: Boxplots of the tunabilities of the hyperparameters of the different algorithms with respect to optimal defaults. The y-axis is on a logarithmic scale. All values below 10^{-3} were set to 10^{-3} to be able to display them. Same definition of whiskers as in Figure 2.

	cp	maxdepth	minbucket	minsplit
cp	0.002	0.003	0.006	0.004
maxdepth		0.002	0.007	0.005
minbucket			0.006	0.011
minsplit				0.004

Table 4: Tunability d_{i_1,i_2} of hyperparameters of `rpart`, diagonal shows tunability of the single hyperparameters.

	maxdepth	minbucket	minsplit
cp	0.0007	0.0005	0.0004
maxdepth		0.0014	0.0019
minbucket			0.0055

Table 5: Joint gain g_{i_1,i_2} of tuning two hyperparameters instead of the most important in `rpart`.

5.4. Tunability of Hyperparameter Combinations and Joint Gains

As an example, Table 4 displays the average tunability d_{i_1,i_2} of all 2-way hyperparameter combinations for `rpart`. Obviously, the increased flexibility in tuning a 2-way combination enables larger improvements when compared with the tunability of one of the respective individual parameters. In Table 5 the joint gain of tuning two hyperparameters g_{i_1,i_2} instead of only the best as defined in Section 3.5 can be seen. The parameters `minsplit` and `minbucket` have the biggest joint effect, which is not very surprising, as they are closely related: `minsplit` is the minimum number of observations that must exist in a node in order for a split to be attempted and `minbucket` the minimum number of observations in any terminal leaf node. If a higher value of `minsplit` than the default performs better on a dataset it is possibly not enough to set it higher without also increasing `minbucket`, so the strong relationship is quite clear. Again, further figures for other algorithms are available through the shiny app. Another remarkable example is the combination of `sample.fraction` and `min.node.size` in `ranger`: the joint gain is very low and tuning `sample.fraction` only seems to be enough, which is concordant to the results of Scornet (2018). Moreover, in `xgboost` the joint gain of `nrounds` and `eta` is relatively low, which is not surprising, as these parameters are highly connected with each other (when setting `nrounds` higher, `eta` should be set lower and vice versa).

5.5. Hyperparameter Space for Tuning

The hyperparameter space for tuning, as defined in Equation (10) in Section 3.6 and based on the 0.05 and 0.95 quantiles, is displayed in Table 3. All optimal defaults are contained in this hyperparameter space while some of the package defaults are not.

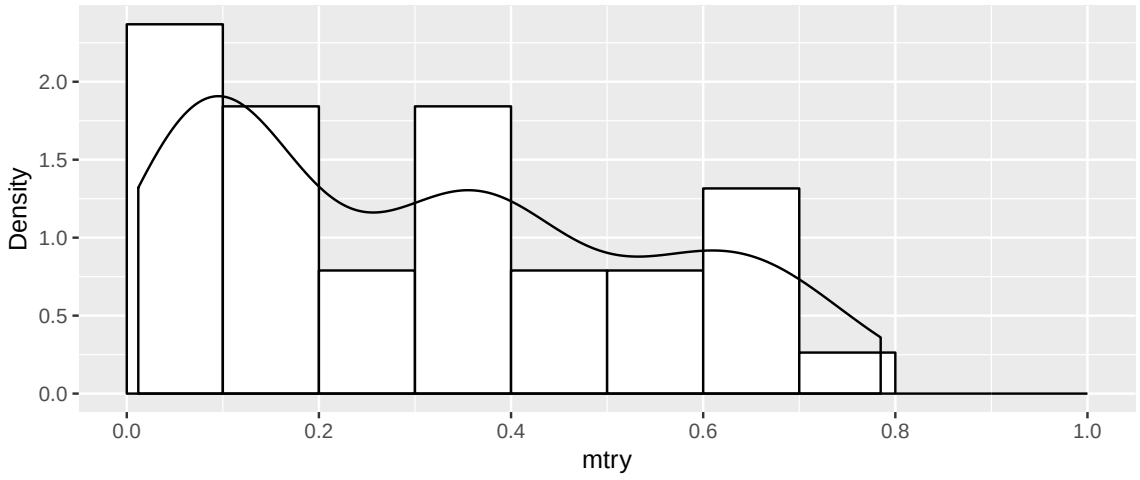

Figure 4: Density and histogram of best parameter values for `mtry` of random forest over all considered datasets.

As an example, Figure 4 displays the full histogram of the best values of `mtry` of the random forest over all datasets. Note that for quite a few datasets much higher values than the package defaults seem advantageous.

6. Conclusion and Discussion

Our paper provides concise and intuitive definitions for optimal defaults of ML algorithms and the impact of tuning them either jointly, tuning individual parameters or combinations, all based on the general concept of surrogate empirical performance models. Tunability values as defined in our framework are easily and directly interpretable as *how much performance can be gained by tuning this hyperparameter?*. This allows direct comparability of the tunability values across different algorithms.

In an extensive OpenML benchmark, we computed optimal defaults for elastic net, decision tree, k-nearest neighbors, SVM, random forest and xgboost and quantified their tunability and the tunability of their individual parameters. This—to the best of our knowledge—has never been provided before in such a principled manner. Our results are often in line with common knowledge from literature and our method itself now allows an analogous analysis for other or more complex methods.

Our framework is based on the concept of default hyperparameter values, which can be seen both as an advantage (default values are a valuable output of the approach) and as an inconvenience (the determination of the default values is an additional analysis step and needed as a reference point for most of our measures).

We now compare our method with van Rijn and Hutter (2017). In contrast to us, they apply the functional ANOVA framework from Hutter et al. (2014) on a surrogate random forest to assess the importance of hyperparameters regarding empirical performance of a support vector machine, random forest and adaboost, which results in numerical importance

scores for individual hyperparameters. Their numerical scores are - in our opinion - less directly interpretable, but they do not rely on defaults as a reference point, which one might see as an advantage. They also propose a method for calculating hyperparameter priors, combine it with the tuning procedure hyperband, and assess the performance of this new tuning procedure. In contrast, we define and calculate ranges for all hyperparameters. Setting ranges for the tuning space can be seen as a special case of a prior distribution - the uniform distribution on the specified hyperparameter space. Regarding the experimental setup, we compute more hyperparameter runs (around 2.5 million vs. 250000), but consider only the 38 binary classification datasets of OpenML100 while van Rijn and Hutter (2017) use all the 100 datasets which also contain multiclass datasets. We evaluate the performance of different surrogate models by 10 times repeated 10-fold cross-validation to choose an appropriate model and to assure that it performs reasonably well.

Our study has some limitations that could be addressed in the future: a) We only considered binary classification, where we tried to include a wider variety of datasets from different domains. In principle this is not a restriction as our methods can easily be applied to multiclass classification, regression, survival analysis or even algorithms not from machine learning whose empirical performance is reliably measurable on a problem instance. b) Uniform random sampling of hyperparameters might not scale enough for very high dimensional spaces, and a smarter sequential technique might be in order here, see Bossek et al. (2015) for an potential approach of sampling across problem instances to learn optimal mappings from problem characteristics to algorithm configurations. c) We currently are learning static defaults, which cannot depend on dataset characteristics (like number of features, or further statistical measures). Doing so might improve performance results of optimal defaults considerably, but would require a more complicated approach. A recent paper regarding this topic was published by van Rijn et al. (2018). d) Our approach still needs initial ranges to be set, in order to run our sampling procedure. Only based on these wider ranges we can then compute more precise, closer ranges.

Acknowledgments

We would like to thank Joaquin Vanschoren for support regarding the OpenML platform and Andreas Müller, Jan van Rijn, Janek Thomas and Florian Pfisterer for reviewing and useful comments. Thanks to Jenny Lee for language editing. This work has been partially funded by grant BO3139/2-3 to ALB from the German Research Foundation and 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.

Appendix A. Additional Graphs and Tables

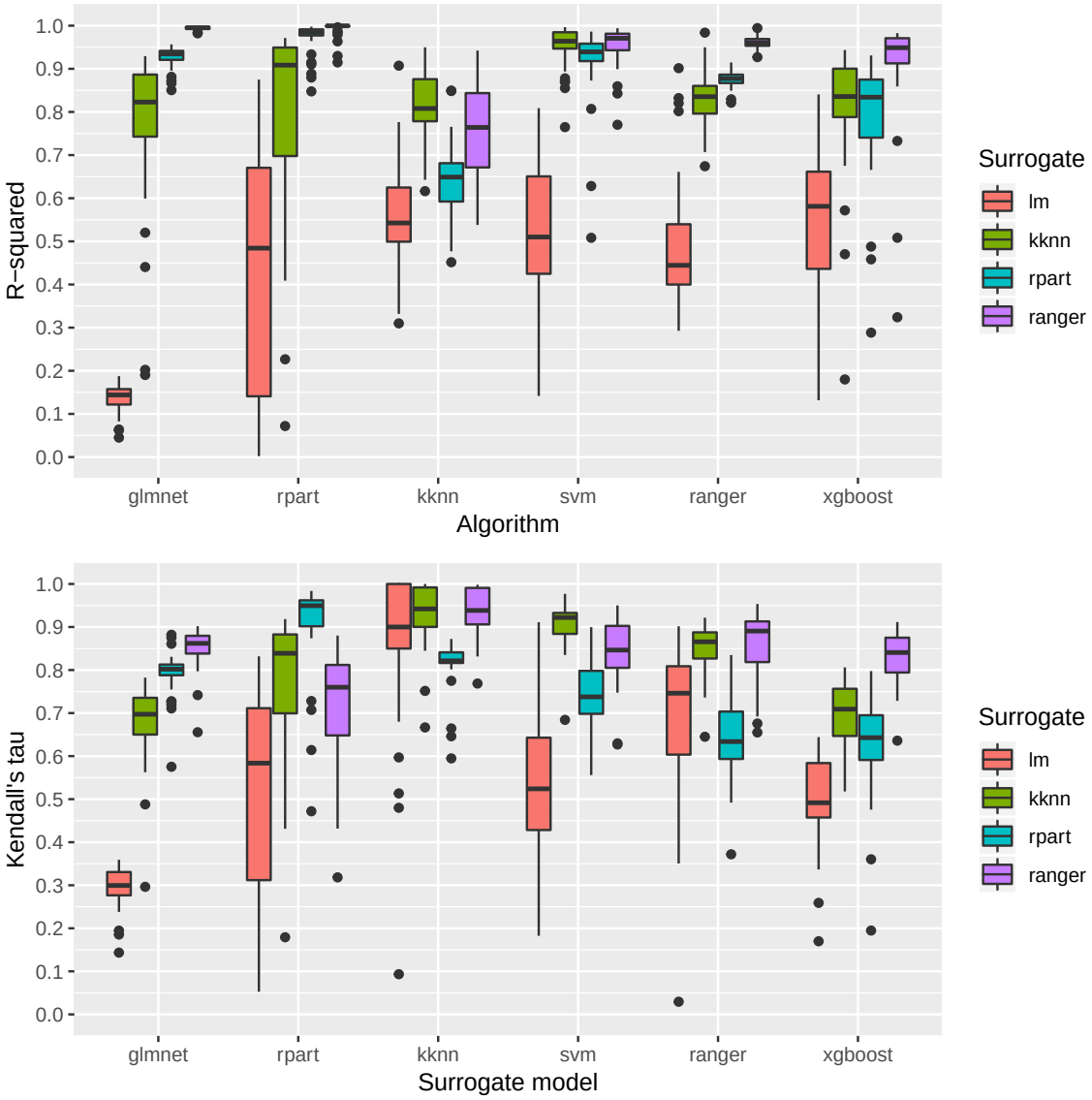

Figure 5: Same as Figure 1 but with exchanged x-axis and legend. Average performances over the datasets of different surrogate models (target: AUC) for different algorithms (that were presented in 4.2).

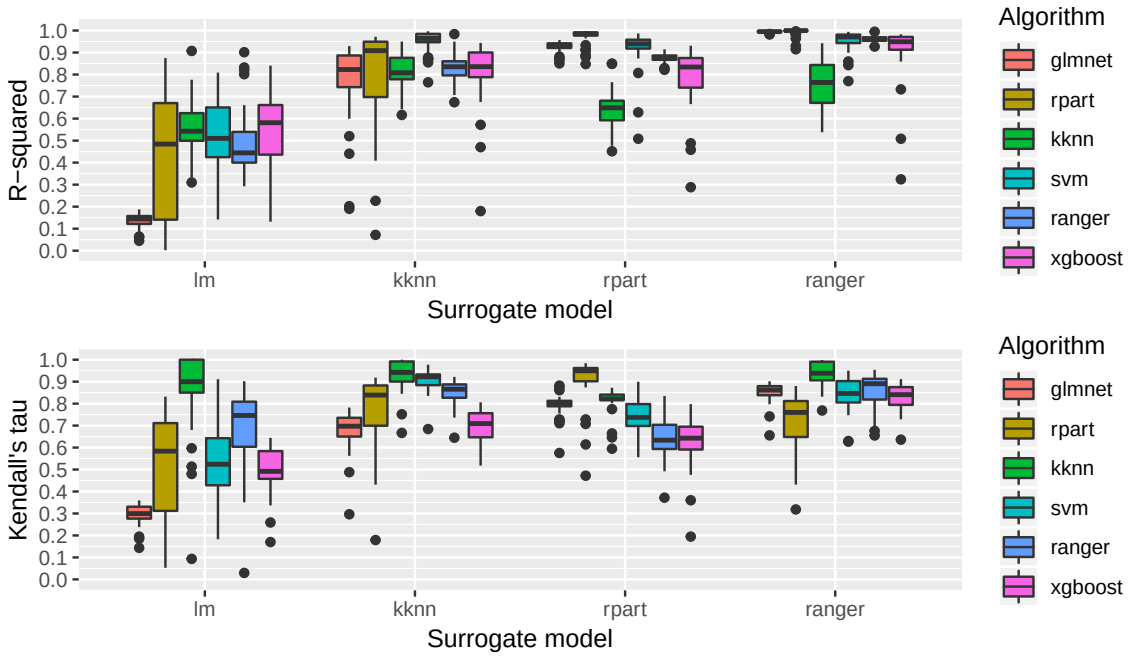

Figure 6: Surrogate model comparison as in Figure 1 but with accuracy as target measure.

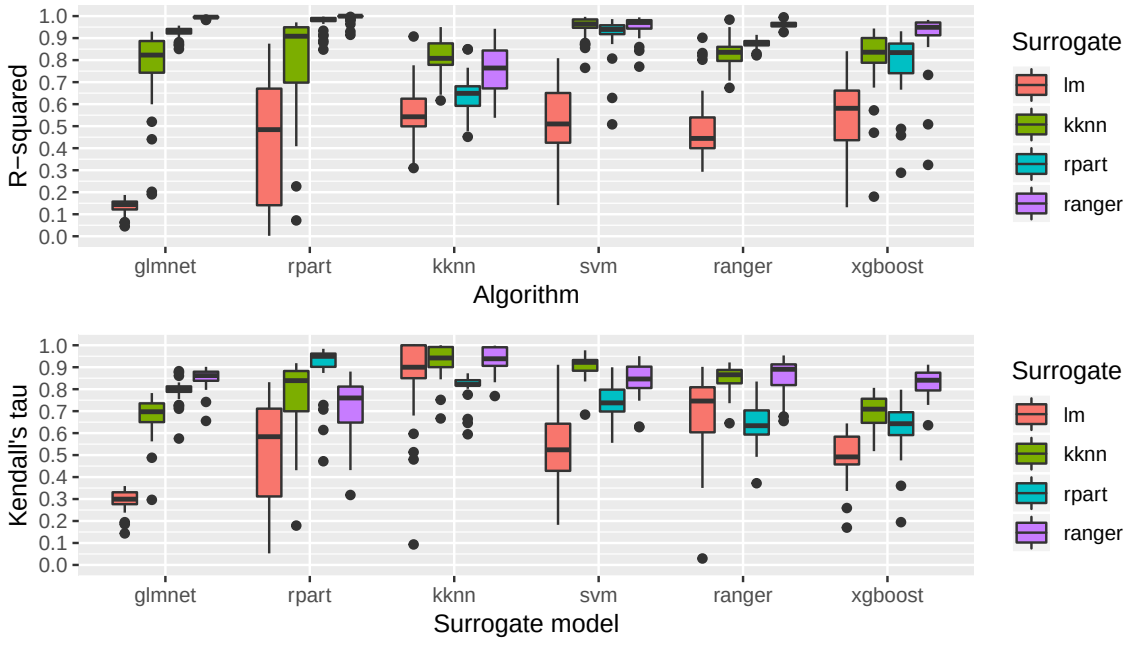

Figure 7: Surrogate model comparison as in Figure 6 (target: accuracy) but with exchanged x-axis and legend.

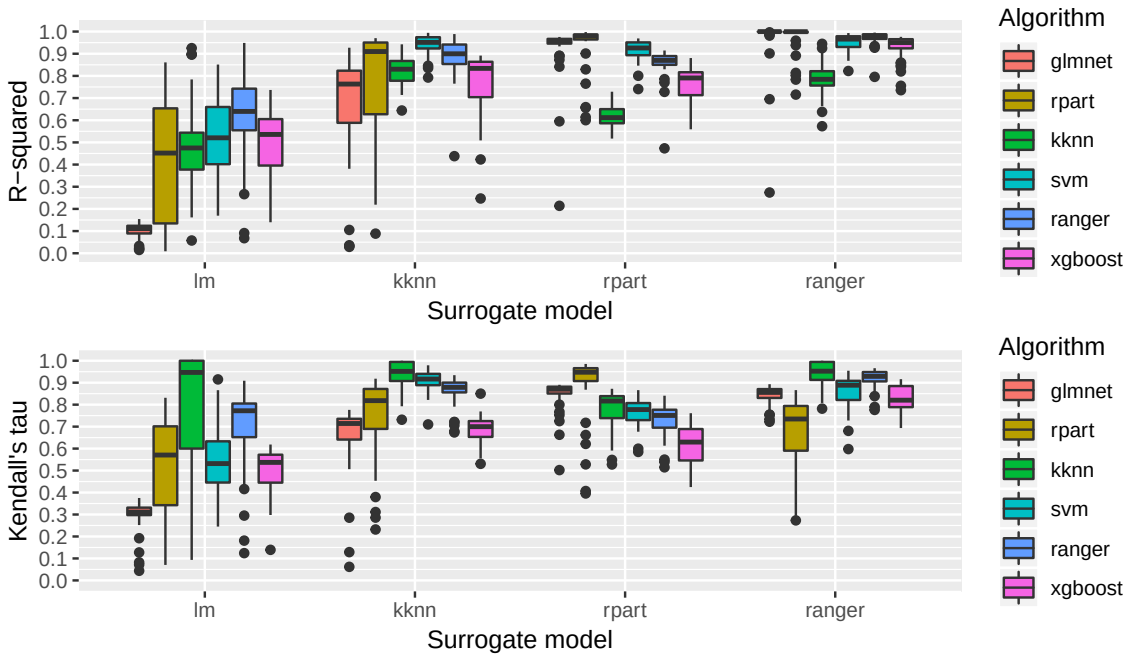

Figure 8: Surrogate model comparison as in Figure 1 but with Brier score as target measure.

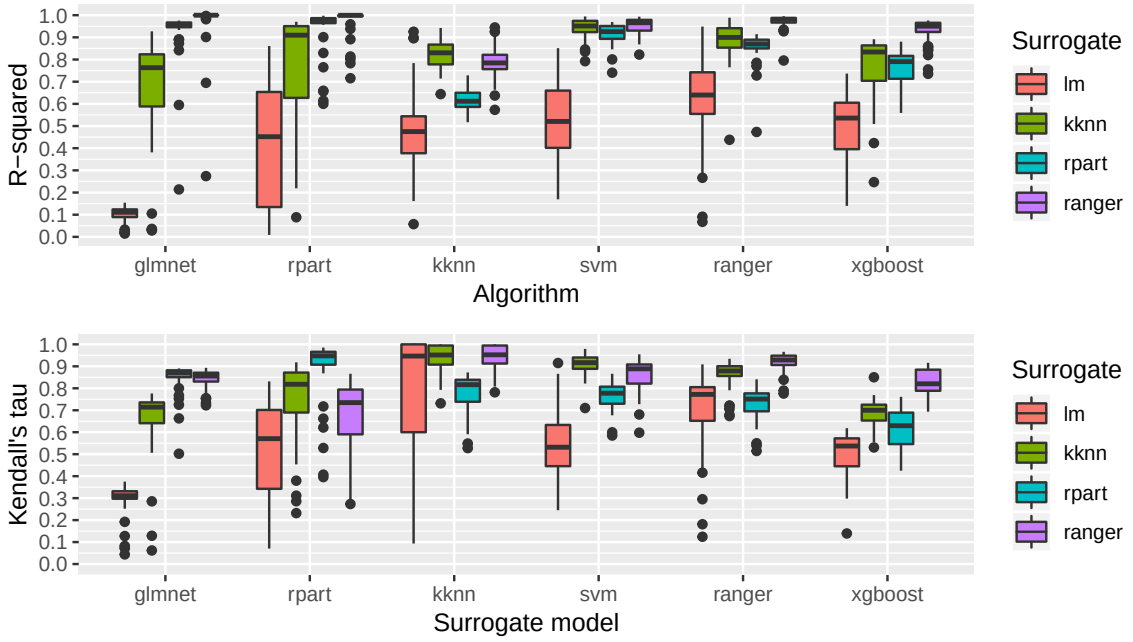

Figure 9: Surrogate model comparison as in Figure 8 but with exchanged x-axis and legend.

Algorithm	Tun.P	Tun.O	Tun.O-CV	Improv	Impr-CV
glmnet	0.042 ± 0.020	0.019 ± 0.010	0.042 ± 0.018	0.023 ± 0.021	0.001 ± 0.013
rpart	0.020 ± 0.004	0.012 ± 0.002	0.014 ± 0.004	0.008 ± 0.003	0.005 ± 0.002
kknn	0.021 ± 0.006	0.008 ± 0.002	0.010 ± 0.004	0.013 ± 0.005	0.010 ± 0.006
svm	0.041 ± 0.009	0.030 ± 0.008	0.041 ± 0.012	0.011 ± 0.004	-0.001 ± 0.011
ranger	0.016 ± 0.004	0.007 ± 0.001	0.009 ± 0.002	0.009 ± 0.004	0.006 ± 0.004
xgboost	0.034 ± 0.005	0.011 ± 0.004	0.012 ± 0.004	0.023 ± 0.004	0.022 ± 0.004

Table 6: Mean tunability as in Table 2, but calculated for the accuracy. Overall tunability (regarding accuracy) with the package defaults (Tun.P) and the optimal defaults (Tun.O) as reference points, cross-validated tunability (Tun.O-CV), average improvement (Improv) and cross-validated average improvement (Impr-CV) obtained by using optimal defaults compared to package defaults. The (cross-validated) improvement can be calculated by the (rounded) difference between Tun.P and Tun.O (Tun.O-CV). Standard error of the mean (SEM) is given behind the "±"-sign.

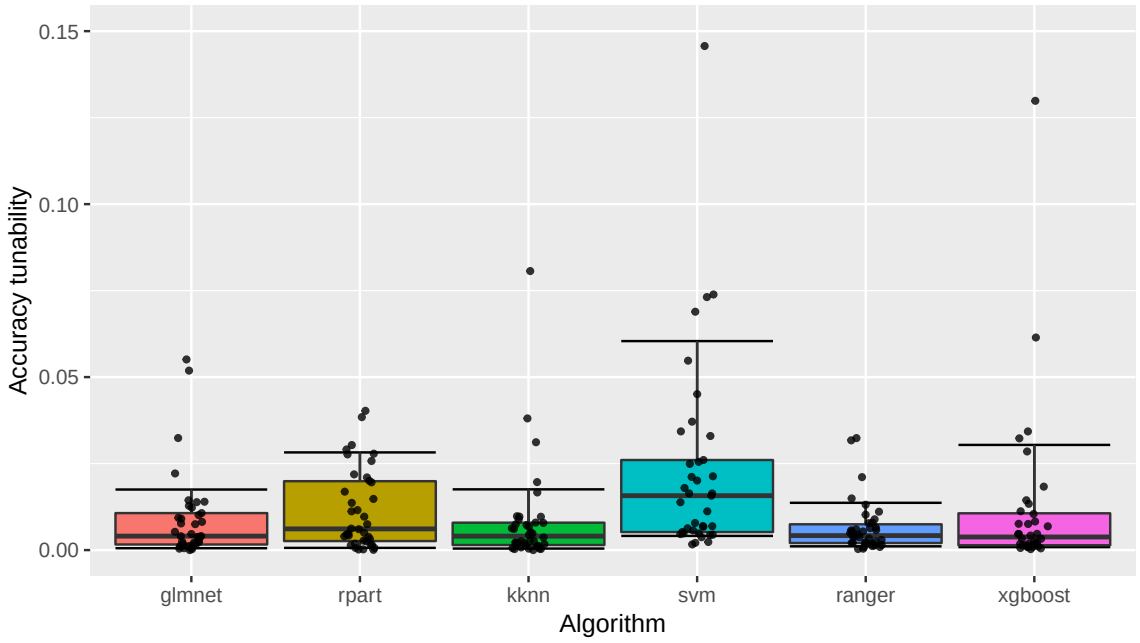

Figure 10: Boxplots of the tunabilities (accuracy) of the different algorithms with respect to optimal defaults.

Parameter	Def.P	Def.O	Tun.P	Tun.O	$q_{0.05}$	$q_{0.95}$
glmnet			0.042	0.019		
alpha	1	0.252	0.022	0.010	0.015	0.979
lambda	0	0.005	0.029	0.017	0.001	0.223
rpart			0.020	0.012		
cp	0.01	0.002	0.013	0.008	0	0.528
maxdepth	30	19	0.004	0.004	10	28
minbucket	7	5	0.005	0.006	1.85	43.15
minsplit	20	13	0.002	0.003	6.7	47.6
kknn			0.021	0.008		
k	7	14	0.021	0.008	2	30
svm			0.041	0.030		
kernel	radial	radial	0.019	0.018		
cost	1	936.982	0.019	0.003	0.025	943.704
gamma	$1/p$	0.002	0.024	0.020	0.007	276.02
degree	3	3	0.005	0.014	2	4
ranger			0.016	0.007		
num.trees	500	162	0.001	0.001	203.5	1908.25
replace	TRUE	FALSE	0.004	0.001		
sample.fraction	1	0.76	0.003	0.003	0.257	0.971
mtry	$\sqrt{p}$	$p \cdot 0.432$	0.010	0.003	0.081	0.867
respect.unordered.factors	TRUE	TRUE	0.001	0.000		
min.node.size	1	1	0.001	0.002	0.009	0.453
xgboost			0.034	0.011		
nrounds	500	3342	0.004	0.002	1360	4847.15
eta	0.3	0.031	0.005	0.005	0.002	0.445
subsample	1	0.89	0.003	0.002	0.555	0.964
booster	gbtree	gbtree	0.008	0.005		
max_depth	6	14	0.001	0.001	3	13
min_child_weight	1	1.264	0.009	0.002	1.061	7.502
colsample_bytree	1	0.712	0.005	0.001	0.334	0.887
colsample_bylevel	1	0.827	0.006	0.001	0.348	0.857
lambda	1	2.224	0.002	0.002	0.004	5.837
alpha	1	0.021	0.003	0.002	0.003	2.904

Table 7: Tunability measures for single hyperparameters and tuning spaces as in Table 3, but calculated for the accuracy. Defaults (package defaults (Def.P) and own optimal defaults (Def.O)), tunability of the hyperparameters with the package defaults (Tun.P) and our optimal defaults (Tun.O) as reference and tuning space quantiles ($q_{0.05}$ and $q_{0.95}$) for different parameters of the algorithms.

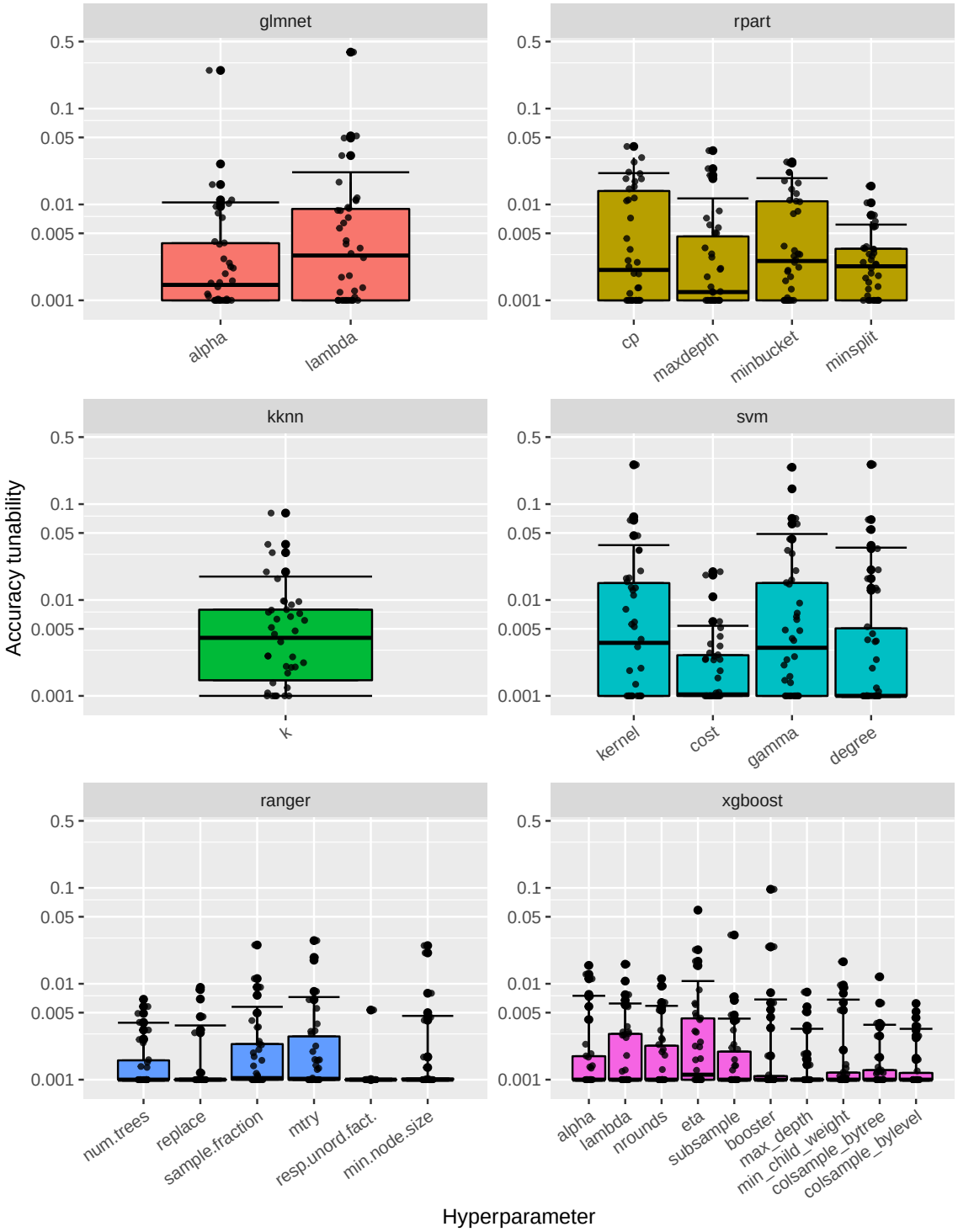

Figure 11: Boxplots of the tunabilities (accuracy) of the hyperparameters of the different algorithms with respect to optimal defaults. The y-axis is on a logarithmic scale. All values below 10^{-3} were set to 10^{-3} to be able to display them. Same definition of whiskers as in Figure 2.

Algorithm	Tun.P	Tun.O	Tun.O-CV	Improv	Impr-CV
glmnet	0.022 ± 0.007	0.010 ± 0.004	0.020 ± 0.014	0.011 ± 0.006	0.001 ± 0.012
rpart	0.015 ± 0.002	0.009 ± 0.002	0.011 ± 0.003	0.006 ± 0.002	0.004 ± 0.002
kknn	0.012 ± 0.003	0.003 ± 0.001	0.003 ± 0.001	0.009 ± 0.003	0.009 ± 0.003
svm	0.026 ± 0.005	0.018 ± 0.004	0.023 ± 0.006	0.008 ± 0.003	0.003 ± 0.005
ranger	0.015 ± 0.004	0.005 ± 0.001	0.006 ± 0.001	0.010 ± 0.004	0.009 ± 0.004
xgboost	0.027 ± 0.003	0.009 ± 0.002	0.011 ± 0.003	0.018 ± 0.002	0.016 ± 0.002

Table 8: Mean tunability as in Table 2, but calculated for the Brier score. Overall tunability (regarding Brier score) with the package defaults (Tun.P) and the optimal defaults (Tun.O) as reference points, cross-validated tunability (Tun.O-CV), average improvement (Improv) and cross-validated average improvement (Impr-CV) obtained by using optimal defaults compared to package defaults. The (cross-validated) improvement can be calculated by the (rounded) difference between Tun.P and Tun.O (Tun.O-CV). Standard error of the mean (SEM) is given behind the "±"-sign.

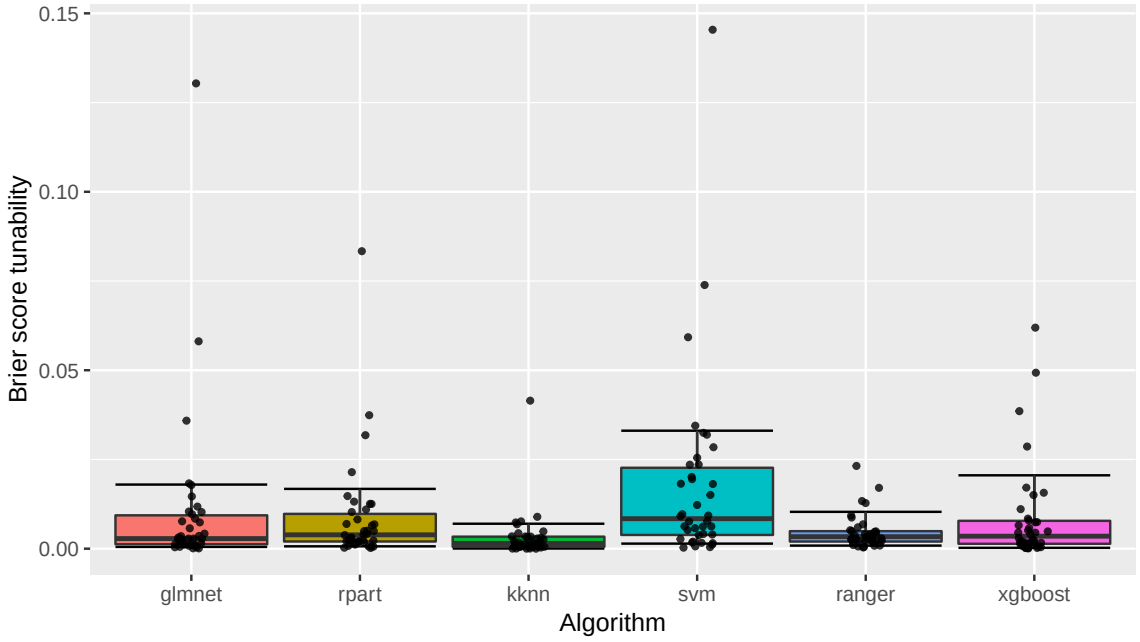

Figure 12: Boxplots of the tunabilities (Brier score) of the different algorithms with respect to optimal defaults.

Parameter	Def.P	Def.O	Tun.P	Tun.O	$q_{0.05}$	$q_{0.95}$
glmnet			0.022	0.010		
alpha	1	0.997	0.009	0.005	0.003	0.974
lambda	0	0.004	0.014	0.007	0.001	0.051
rpart			0.015	0.009		
cp	0.01	0.001	0.009	0.003	0	0.035
maxdepth	30	13	0.002	0.002	9	27.15
minbucket	7	12	0.004	0.006	1	44.1
minsplit	20	18	0.002	0.002	7	49.15
kknn			0.012	0.003		
k	7	19	0.012	0.003	4.85	30
svm			0.026	0.018		
kernel	radial	radial	0.013	0.011		
cost	1	950.787	0.012	0.002	0.002	963.81
gamma	$1/p$	0.005	0.015	0.012	0.001	4.759
degree	3	3	0.003	0.009	2	4
ranger			0.015	0.005		
num.trees	500	198	0.001	0.001	187.85	1568.25
replace	TRUE	FALSE	0.002	0.001		
sample.fraction	1	0.667	0.002	0.003	0.317	0.964
mtry	$\sqrt{p}$	$p \cdot 0.666$	0.010	0.002	0.072	0.954
respect.unordered.factors	TRUE	TRUE	0.000	0.000		
min.node.size	1	1	0.001	0.001	0.008	0.394
xgboost			0.027	0.009		
nrounds	500	2563	0.004	0.002	2018.55	4780.05
eta	0.3	0.052	0.004	0.005	0.003	0.436
subsample	1	0.873	0.002	0.002	0.447	0.951
booster	gbtree	gbtree	0.009	0.004		
max_depth	6	11	0.001	0.001	2.6	13
min_child_weight	1	1.75	0.007	0.002	1.277	5.115
colsample_bytree	1	0.713	0.004	0.002	0.354	0.922
colsample_bylevel	1	0.638	0.004	0.001	0.363	0.916
lambda	1	0.101	0.002	0.003	0.006	28.032
alpha	1	0.894	0.003	0.004	0.003	2.68

Table 9: Tunability measures for single hyperparameters and tuning spaces as in Table 3, but calculated for the Brier score. Defaults (package defaults (Def.P) and own optimal defaults (Def.O)), tunability of the hyperparameters with the package defaults (Tun.P) and our optimal defaults (Tun.O) as reference and tuning space quantiles ($q_{0.05}$ and $q_{0.95}$) for different parameters of the algorithms.

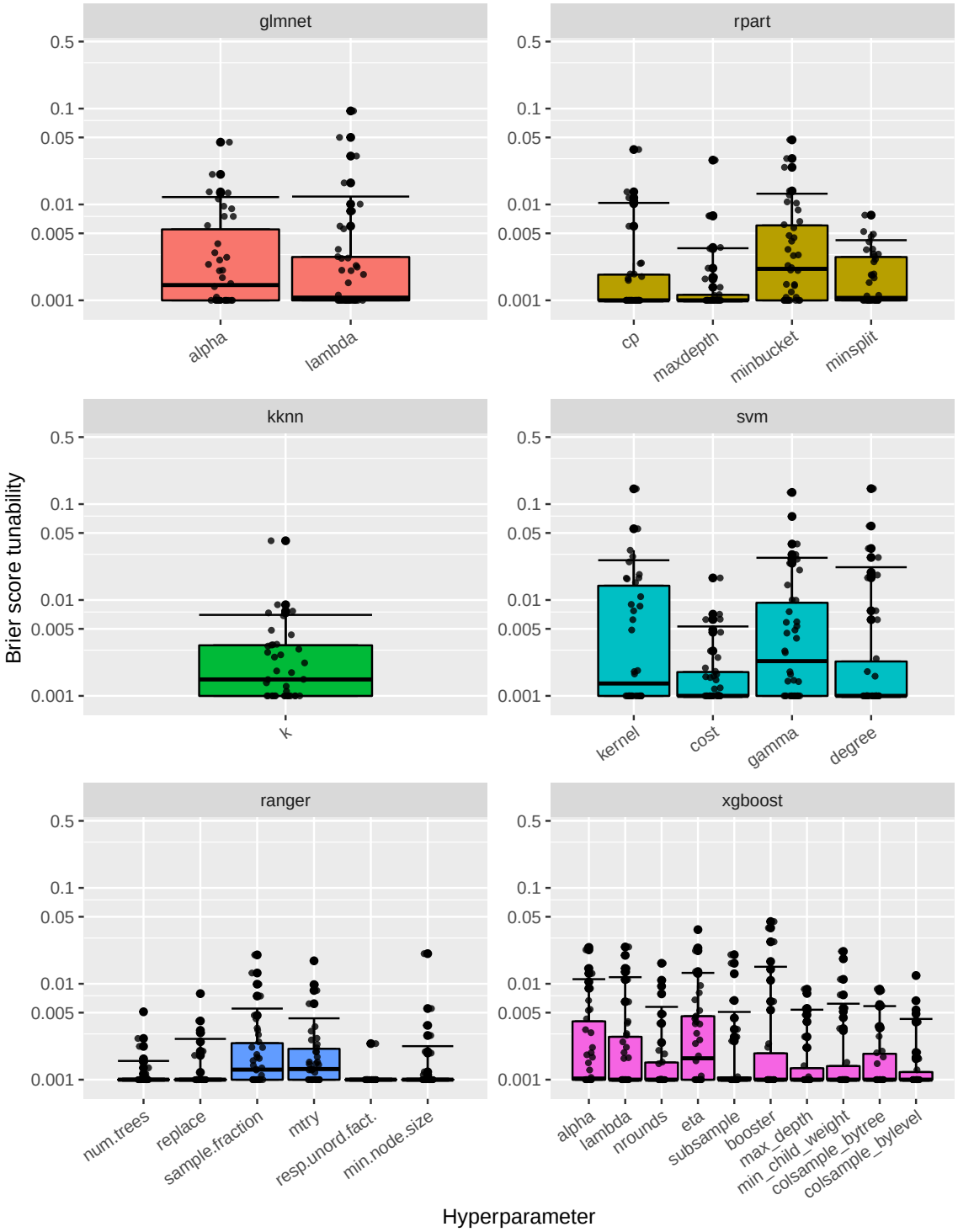

Figure 13: Boxplots of the tunabilities (Brier score) of the hyperparameters of the different algorithms with respect to optimal defaults. The y-axis is on a logarithmic scale. All values below 10^{-3} were set to 10^{-3} to be able to display them. Same definition of whiskers as in Figure 2.

Measure	**AUC**		**Accuracy**		**Brier score**	
Parameter	Tun.O	Tun.O-CV	Tun.O	Tun.O-CV	Tun.O	Tun.O-CV
glmnet	0.024	0.037	0.019	0.042	0.010	0.020
alpha	0.006	0.006	0.010	0.026	0.005	0.015
lambda	0.021	0.034	0.017	0.039	0.007	0.018
rpart	0.012	0.016	0.012	0.014	0.009	0.011
cp	0.002	0.002	0.008	0.008	0.003	0.005
maxdepth	0.002	0.002	0.004	0.004	0.002	0.003
minbucket	0.006	0.009	0.006	0.007	0.006	0.006
minsplit	0.004	0.004	0.003	0.003	0.002	0.003
kknn	0.006	0.006	0.008	0.010	0.003	0.003
k	0.006	0.006	0.008	0.010	0.003	0.003
svm	0.042	0.048	0.030	0.041	0.018	0.023
kernel	0.024	0.030	0.018	0.031	0.011	0.016
cost	0.006	0.006	0.003	0.003	0.002	0.002
gamma	0.022	0.028	0.020	0.031	0.012	0.016
degree	0.014	0.020	0.014	0.027	0.009	0.014
ranger	0.006	0.007	0.007	0.009	0.005	0.006
num.trees	0.001	0.002	0.001	0.003	0.001	0.001
replace	0.001	0.002	0.001	0.002	0.001	0.001
sample.fraction	0.002	0.002	0.003	0.003	0.003	0.003
mtry	0.003	0.004	0.003	0.005	0.002	0.003
respect.unordered.factors	0.000	0.000	0.000	0.001	0.000	0.000
min.node.size	0.001	0.001	0.002	0.002	0.001	0.001
xgboost	0.014	0.017	0.011	0.012	0.009	0.011
nrounds	0.002	0.002	0.002	0.003	0.002	0.002
eta	0.005	0.006	0.005	0.006	0.005	0.006
subsample	0.002	0.002	0.002	0.002	0.002	0.002
booster	0.008	0.008	0.005	0.005	0.004	0.004
max_depth	0.001	0.001	0.001	0.001	0.001	0.001
min_child_weight	0.002	0.003	0.002	0.002	0.002	0.003
colsample_bytree	0.001	0.002	0.001	0.001	0.002	0.002
colsample_bylevel	0.001	0.001	0.001	0.001	0.001	0.002
lambda	0.002	0.003	0.002	0.003	0.003	0.004
alpha	0.002	0.004	0.002	0.003	0.004	0.004

Table 10: Tunability with optimal defaults as reference without (Tun.O) and with (Tun.O-CV) cross-validation for AUC, accuracy and Brier score

References

Charles Audet, John E. Dennis, Douglas Moore, Andrew Booker, and Paul Frank. A surrogate-model-based method for constrained optimization. In *8th Symposium on Multidisciplinary Analysis and Optimization*, page 4891, 2000.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.

André Biedenkapp, Marius Thomas Lindauer, Katharina Eggensperger, Frank Hutter, Chris Fawcett, and Holger H Hoos. Efficient parameter importance analysis via ablation with surrogates. In *AAAI*, pages 773–779, 2017.

Mauro Birattari, Zhi Yuan, Prasanna Balaprakash, and Thomas Stützle. F-Race and iterated F-Race: An overview. In *Experimental Methods for the Analysis of Optimization Algorithms*, pages 311–336. Springer, 2010.

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

Bernd Bischl, Michel Lang, Lars Kotthoff, Julia Schiffner, Jakob Richter, Erich Studerus, Giuseppe Casalicchio, and Zachary M. Jones. mlr: Machine learning in R. *Journal of Machine Learning Research*, 17(170):1–5, 2016.

Bernd Bischl, Giuseppe Casalicchio, Matthias Feurer, Frank Hutter, Michel Lang, Rafael G. Mantovani, Jan N. van Rijn, and Joaquin Vanschoren. OpenML benchmarking suites and the OpenML100. *ArXiv preprint arXiv:1708.03731*, 2017a. URL `https://arxiv.org/abs/1708.03731`.

Bernd Bischl, Jakob Richter, Jakob Bossek, Daniel Horn, Janek Thomas, and Michel Lang. mlrMBO: A modular framework for model-based optimization of expensive black-box functions. *ArXiv preprint arXiv:1703.03373*, 2017b. URL `https://arxiv.org/abs/1703.03373`.

Jakob Bossek, Bernd Bischl, Tobias Wagner, and Günter Rudolph. Learning feature-parameter mappings for parameter tuning via the profile expected improvement. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 1319–1326. ACM, 2015.

Giuseppe Casalicchio, Jakob Bossek, Michel Lang, Dominik Kirchhoff, Pascal Kerschke, Benjamin Hofner, Heidi Seibold, Joaquin Vanschoren, and Bernd Bischl. OpenML: An R package to connect to the machine learning platform OpenML. *Computational Statistics*, 32(3):1–15, 2017.

Katharina Eggensperger, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Surrogate benchmarks for hyperparameter optimization. In *Proceedings of the 2014 International Conference on Meta-learning and Algorithm Selection-Volume 1201*, pages 24–31. CEUR-WS.org, 2014.

Katharina Eggensperger, Marius Lindauer, Holger H Hoos, Frank Hutter, and Kevin Leyton-Brown. Efficient benchmarking of algorithm configurators via model-based surrogates. *Machine Learning*, pages 1–27, 2018.

Agoston E Eiben and Selmar K Smit. Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm and Evolutionary Computation*, 1(1):19–31, 2011.

Chris Fawcett and Holger H Hoos. Analysing differences between algorithm configurations through ablation. *Journal of Heuristics*, 22(4):431–458, 2016.

Matthias Feurer, Benjamin Letham, and Eytan Bakshy. Scalable meta-learning for bayesian optimization. *arXiv preprint 1802.02219*, 2018. URL `https://arxiv.org/abs/1802.02219`.

Isabelle Guyon, Amir Saffari, Gideon Dror, and Gavin Cawley. Model selection: Beyond the bayesian/frequentist divide. *Journal of Machine Learning Research*, 11(Jan):61–87, 2010.

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

Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Identifying key algorithm parameters and instance features using forward selection. In *International Conference on Learning and Intelligent Optimization*, pages 364–381. Springer, 2013.

Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. An efficient approach for assessing hyperparameter importance. In *ICML*, volume 32 of *JMLR Workshop and Conference Proceedings*, pages 754–762, 2014.

Max Kuhn. Building predictive models in R using the caret package. *Journal of statistical software*, 28(5):1–26, 2008.

Max Kuhn, Steve Weston, Chris Keefer, and Nathan Coulter. *Cubist: Rule- and instance-based regression modeling*, 2016. R package version 0.0.19.

Daniel Kühn, Philipp Probst, Janek Thomas, and Bernd Bischl. OpenML R bot benchmark data (final subset), 2018a. URL `https://figshare.com/articles/OpenML_R_Bot_Benchmark_Data_final_subset_/5882230/2`.

Daniel Kühn, Philipp Probst, Janek Thomas, and Bernd Bischl. Automatic Exploration of Machine Learning Experiments on OpenML. *ArXiv preprint arXiv:1806.10961*, 2018b. URL `https://arxiv.org/abs/1806.10961`.

Upmanu Lall and Ashish Sharma. A nearest neighbor bootstrap for resampling hydrologic time series. *Water Resources Research*, 32(3):679–693, 1996.

Michel Lang, Bernd Bischl, and Dirk Surmann. batchtools: Tools for R to work on batch systems. *The Journal of Open Source Software*, 2(10), 2017.

Gang Luo. A review of automatic selection methods for machine learning algorithms and hyper-parameter values. *Network Modeling Analysis in Health Informatics and Bioinformatics*, 5(1):1–16, 2016.

Rafael G. Mantovani, Tomáš Horváth, Ricardo Cerri, Andre C.P.L.F. Carvalho, and Joaquin Vanschoren. Hyper-parameter tuning of a decision tree induction algorithm. In *Brazilian Conference on Intelligent Systems (BRACIS 2016)*, 2016.

Rafael Gomes Mantovani, Tomáš Horváth, Ricardo Cerri, Sylvio Barbon Junior, Joaquin Vanschoren, André Carlos Ponce de de Carvalho, and Leon Ferreira. An empirical study on hyperparameter tuning of decision trees. *ArXiv preprint arXiv:1812.02207*, 2018. URL `https://arxiv.org/abs/1812.02207`.

Erwan Scornet. Tuning parameters in random forests. *ESAIM: Procs*, 60:144–162, 2018.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

Jan N. van Rijn and Frank Hutter. Hyperparameter importance across datasets. *ArXiv preprint arXiv:1710.04725*, 2017. URL `https://arxiv.org/abs/1710.04725`.

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

Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. OpenML: Networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60, 2013.

Levi Waldron, Melania Pintilie, Ming-Sound Tsao, Frances A Shepherd, Curtis Huttenhower, and Igor Jurisica. Optimized application of penalized regression methods to diverse genomic data. *Bioinformatics*, 27(24):3399–3406, 2011.