Columbia University in the City of New York
Eindhoven University of Technology **2IMC10 Internship**

Importance of Tuning Hyperparameters

Hilde Weerts
h.j.p.weerts@student.tue.nl

May 1, 2018

Abstract

The performance of many machine learning algorithms depends on their hyperparameter settings. The goal of this study is to determine whether it is important to tune a single hyperparameter or whether it can be safely set to a default value. We present a methodology to determine the importance of tuning a hyperparameter using a tunability measure and non-inferiority test. Additionally, we present a simple procedure that can be used to determine reasonable default parameters. We apply our approach in a benchmark study using 59 datasets from OpenML. Our results show that leaving a single hyperparameter at a default value in some cases is non-inferior to tuning the hyperparameter.

1 Introduction

Most modern machine learning algorithms have parameters that need to be fixed before running them. Such parameters are referred to as *hyperparameters*. In many cases, the performance of an algorithm depends on its hyperparameter settings. Methods for tuning hyperparameters for the problem at hand require the machine learning practitioner to define a search space: the set of hyperparameters and ranges that need to be considered. These decisions are usually based on intuition and experience. Currently, there does not exist empirical evidence on which hyperparameters are important to tune and which hyperparameters result in similar performance when set to a reasonable default value. Hyperparameters that fall into the latter category might be eliminated from the search space entirely when computational resources are limited. The objective of this work is to study the importance of tuning hyperparameters and to provide empirical evidence of hyperparameters that might be eliminated from the search space. That is, we aim to determine if using a default setting of a hyperparameter results in similar performance compared to tuning the hyperparameter.

We present a methodology to determine which hyperparameters are important to tune, based on empirical performance data across multiple datasets. Additionally, we introduce a simple procedure to deduce reasonable default parameters from performance data across many datasets. We apply our approach in a benchmark study of two popular classification algorithms. In particular, we analyze the importance of hyperparameter tuning based on the performance of these algorithms across 59 datasets that were taken from the OpenML-CC18 benchmark suite.

Our results suggest that leaving a single hyperparameter at our computed default value can lead to non-inferior performance. Moreover, given a random search approach with a limited number of iterations, fixing the hyperparameter to our default value sometimes even outperforms tuning all hyperparameters at once. Machine learning practitioners might want to leave these hyperparameters to their default value, rather than invoking an expensive random search procedure. For other hyperparameters, we observed a bigger difference between the fixed and non-fixed conditions. This indicates that these hyperparameters should not be left at our computed default value.

This remainder of this work is structured as follows. In Section 2, we provide a short overview of related work. In Section 3, we present our default parameter determination procedure as well as a methodology for determining the importance of tuning a hyperparameter. Section 4 covers details of our benchmark experiments, including the used datasets, algorithms, hyperparameters, performance measures, and search strategies. In Section 5, the results of the experiments are presented. Finally, conclusions and limitations of are discussed in Section 6.

2 Related Work

In this section, we first discuss related work in the field of hyperparameter importance. Second, we briefly review previous work on the determination of default hyperparameters.

2.1 Hyperparameter Importance

In the work of van Rijn and Hutter (2017), important hyperparameters are defined as parameters that explain most variance in performance across multiple datasets. The explained variance is determined through a functional ANOVA framework. A limitation of this approach is that the results do not directly translate into guidelines on which hyperparameters are important to tune. For example, according to this definition, a hyperparameter for which one specific setting in the hyperparameter range always results in good performance is considered very important, as it explains variance in the performance. However, such a parameter does not require elaborate tuning. Instead, it can simply be set to the setting that always results in good performance.

This issue is alleviated by Probst et al. (2018), who introduce the *tunability* of a hyperparameter: the performance gain that can be achieved by tuning the hyperparameter. The authors compare the performance of leaving all hyperparameters at their default value and the performance of tuning the hyperparameter of interest while leaving all other hyperparameters at their default value. By taking into account a baseline default parameter, their approach can be used directly to determine the importance of tuning a hyperparameter. The approach taken in this work is very similar to the work of Probst et al. (2018). However, in contrast to the aforementioned work, we are interested in the performance loss that is incurred when a hyperparameter is set to a default value, while all other parameters are tuned. Probst et al. (2018) mention that "the two alternatives can be seen as similar to forward and backward selection of variables in stepwise regression" (p. 7).

2.2 Hyperparameter Default Values

Besides determining the importance of hyperparameter tuning, we also investigate the determination of reasonable default hyperparamter values.

In the work of Probst et al. (2018), default values are determined using surrogate models as follows. First, for each of the 38 binary classification tasks in the study, a surrogate model that predicts the performance of the algorithm based on hyperparameter settings is learned from empirical performance data. Second, a large number of hyperparameter configurations are sampled and their performance is estimated using the surrogate models. Finally, the default parameters are determined by minimizing the average risk. In contrast to this method, we will use a simple yet intuitive heuristic to determine default parameters directly from the performance data.

3 Methods

In this section, we first discuss our default value determination approach. Second, we present the experiment design and associated method that is used to determine the importance of tuning a hyperparameter.

3.1 Notations

Let A be an algorithm with n hyperparameters with domains $\Theta_1, ..., \Theta_n$ and configuration space $\boldsymbol{\Theta} = \Theta_1 \times ... \times \Theta_n$. Let $\boldsymbol{\theta} = \langle \theta_1, ..., \theta_n \rangle$ with $\theta_i \in \Theta_i$ denote an instantiation of A. Let θ_i^j denote the default setting of hyperparameter i for dataset j. Let $R_j(\theta_i)$ denote the expected loss (risk) of fixing hyperparameter i to θ_i while tuning all other $n-1$ hyperparameters for dataset j. Let θ_i^{*j} denote the best setting for parameter i for dataset j, i.e.:

$$\theta_i^{*j} = \underset{\theta_i \in \Theta_i}{\arg\min}\, R_j(\theta_i) \tag{1}$$

Given an algorithm A and hyperparameter i, we aim to determine (1) a default setting θ_i^j, for $i = 1, ..., n$ and $j = 1, ..., m$, (2) whether $R_j(\theta_i^j)$ is non-inferior to $R_j(\theta_i^{*j})$ across datasets.

3.2 Hyperparameter Default Values

The default setting problem can be formalized as follows. Given

- an algorithm A with configuration space $\boldsymbol{\Theta}$
- a hyperparameter i with domain Θ_i
- M datasets $\mathcal{D}^{(1)}, ..., \mathcal{D}^{(m)}$
- for each dataset, a set of empirical performance measurements $\langle \boldsymbol{\theta}_k, y_k \rangle_{k=1}^K$ for different hyperparameter settings $\boldsymbol{\theta}_k \in \boldsymbol{\Theta}$

we aim to determine a default setting θ_i^* for hyperparameter i. However, if we were to use a single default value in the hyperparameter importance experiment, we leak information on the best hyperparameter value of a specific dataset. Therefore, we determine a default setting using a leave-one-out approach. That is, we determine the default setting for dataset j, denoted as θ_i^{j*}, using only performance data related to the other $M-1$ datasets.

3.2.1 Method

We use a simple procedure to determine default parameters. The intuition behind our method is to find a hyperparameter setting that, across many datasets, most often results in good performance.

First, we take a subset of all empirical performance measurements that represent good performance. That is, for each dataset, we pick the n hyperparameter settings that resulted in the best performance. Let this be $\langle \boldsymbol{\theta}_{ij}, y_{ij} \rangle$, for $i = 1, ..., n$ and $j = 1, ..., M$. Note that the top n performance data consists of $M \cdot n$ empirical performance measurements. This is similar to the first step of the prior distribution estimation method proposed by van Rijn and Hutter (2017). The choice of n is potentially important. On the one hand, picking n too small might give a large weight to outliers. On the other hand, picking n too large could result in the inclusion of bad performance measurements in the subset.

Second, we determine which parameter setting occurs most frequently within this subset. The intuition is that a setting that worked well on most datasets is a good candidate for a default parameter. For hyperparameters with a nominal or boolean domain, the setting that worked well most often is simply the mode. Since hyperparameter settings are picked randomly across the domain, parameters with a continuous domain or large integer domain (e.g. more than 50 possible values) must first be discretized. In this work, we will determine the bin size h using the Freedman-Diaconis rule (Freedman and Diaconis, 1981), presented in Equation 2.

$$h = 2\frac{IQR}{\sqrt[3]{n}} \tag{2}$$

where IQR is the interquartile range of the data and n is the number of observations. The Freedman-Diaconis rule is known to be resilient to outliers. An alternative of this rule is Sturges formula (Sturges, 1926). The automatic histogram binning, as implemented in the python library `numpy`[1], uses the maximum of the Freedman-Diaconis rule and Sturges formula. However, Sturges formula assumes that the data follows a Gaussian distribution. Since we have no reason to assume that this holds for our data, we only use the Freedman-Diaconis rule.

The advantage of our method is that it is very simple and intuitive. In future work, it would be interesting to compare our simple heuristic with more sophisticated methods, such as default parameter estimation through surrogate models (Probst et al., 2018).

3.2.2 Meta-feature dependent default values

In the open source machine learning library `scikit-learn` (Pedregosa et al., 2011), the default values of some hyperparameters depend on meta-features of the dataset. In particular, *max_features* of the random forest and *gamma* of the support vector machine depend on the number of features in the dataset. Let p denote the number of features of a dataset. Then the default parameters are $\sqrt{p}$ and $1/p$ for `max_features` and `gamma` respectively. For these hyperparameters, we investigate an alternative default parameter estimation approach. Rather than choosing the hyperparameter that occurs most often, we use non-linear least squares to fit several functions to the top 10 performance data, as implemented in the `curve_fit` function of the python library `scipy`[2].

3.3 Tunability of Hyperparameters

The main goal of our study is to determine whether leaving a hyperparameter at the default value (as determined in the previous section) leads to much worse results than tuning the hyperparameter. In this section, we explain the experiment design and methods that are used to answer this question.

3.3.1 Experiment Design

We adhere a repeated measures experiment design, where the unit of analysis is a dataset. The dependent variable in our study is the cross validated *performance*

[1] See `http://www.numpy.org`
[2] See `https://www.scipy.org`

of an instantiation of an algorithm. The two independent variables in our study are *condition* and *random search seed*. The *condition* variable indicates whether a hyperparameter is *fixed* or *non-fixed*. That is, for some algorithm A, hyperparameter i, and dataset j, we have:

- *Fixed:* fix hyperparameter i to default setting θ_i^{j*} and tune all other $n-1$ hyperparameters.
- *Non-fixed:* tune all n hyperparameters;

The hyperparameters will be tuned using a random search strategy (Bergstra and Bengio, 2012) within a nested cross validation procedure. To ensure that we measure the effect of tuning hyperparameter i, and not the effect of better tuning of other hyperparameters, all hyperparameters other than hyperparameter i will be the same for both conditions in each iteration of the random search. Which hyperparameters are sampled during the random search is determined by the *random search seed*. Note that the measured performance might depend on the sampled hyperparameter settings. In particular, the performance might depend on interactions between hyperparameter i and other hyperparameters. In order to alleviate this potential bias, we repeat the experiment R times, using a different random search seed each time. As stated before, the random seed is the same between conditions *fixed* and *non-fixed*. The random seeds that are used for each of the datasets are selected at random. Note that this implies that the first random seed of task 1 is not necessarily identical to the first random seed of task 2.

3.3.2 Method

We will evaluate the results of the experiment through both an absolute measure (tunability) and a statistical measure (non-inferiority).

Tunability - To understand to what extent the *condition* affects the performance of the algorithm, we compute the difference in risk between the fixed and non-fixed condition. Analogous to Probst et al. (2018), we define *tunability* of hyperparameter i for dataset j and seed r as follows:

$$d_{i,j,r} = R(\theta_i^{j,r}) - R(\theta_i^{*j,r}) \tag{3}$$

Larger values of $d_{i,j,r}$ indicate that the risk of the fixed condition is higher than the risk of the non-fixed condition. To summarize the differences over all M datasets and seeds, we simply compute the average tunability of hyperparameter i:

$$d_i = \frac{1}{M * R} \sum_{r=1}^{R} \sum_{j=1}^{M} d_{i,j,r} \tag{4}$$

.

Another interesting statistic is the standard deviation of $d_{i,j,r}$, which is denoted by s_i.

An assumption of this definition of tunability is that every unit of risk is equally relevant. In practice, however, this might not be the case. For example, when considering the missclassification rate, an increase from 0.01 to 0.02, might be considered worse than an increase from 0.49 to 0.50, as in the former case the number of misclassified instances is doubled. To investigate this, we define the *relative tunability* as the relative difference in risk between the fixed and non-fixed condition as follows:

$$d_{i,j,r}^R = \frac{R(\theta_i^{j,r}) - R(\theta_i^{*j,r})}{R(\theta_i^{*j,r})} \tag{5}$$

Again, relative tunability can be aggregated by computing the average:

$$d_i^R = \frac{1}{M * R} \sum_{r=1}^{R} \sum_{j=1}^{M} d_{i,j,r}^R \tag{6}$$

Additionally, we can compute the standard deviation of $d_{i,j,r}^R$ which we will denote with s_i^R.

Non-inferiority - We use the statistical measure of non-inferiority testing to determine whether the performance of the fixed condition is comparable to the performance of the non-fixed condition. This measure differs from traditional hypothesis testing, where the goal is to show that one group is different from another group with respect to some variable. Instead, we wish to determine whether an effect of one group is non-inferior to the effect in another group. Two one-sided non-inferiority tests can be combined into a method referred to as equivalence testing. This method is often applied in e.g. clinical and psychological research (Walker and Nowacki, 2010; Lakens, 2017). For example, one might like to determine whether some new treatment has similar benefits compared to an existing treatment.

An important consideration in non-inferiority studies is the non-inferiority margin: the range of values for which the difference in effect size is considered to be irrelevant in practice. The non-inferiority margin is also known as smallest effect size of interest. Recall that we would like to determine whether the performance

of the fixed condition is "close enough" to the performance in the non-fixed condition. Therefore, we are mostly interested in the case where the observed risk in the fixed condition is higher than the risk observed in the non-fixed condition. We formulate our research question as follows:

> *Is the observed risk in the fixed condition non-inferior to the risk observed in the non-fixed condition?*

To answer our research question, we will perform a non-parametric one sided test of non-inferiority for paired samples. We use a one-sided version of the two one-sided test (TOST) procedure described in the work of Mara and Cribbie (2012). The authors show that the non-parametric TOST procedure is more powerful for non-normal distributions than the TOST procedure using a paired t-test, because the former is less sensitive to outliers. Since we have no reason to assume that our data follows a normal distribution, we use the non-parametric alternative. The procedure is very similar to the Wilcoxon signed ranks tests, the biggest difference being the inversion of the null and alternative hypothesis. In the remainder of this section, we will describe the procedure in more detail.

Let δ be the non-inferiority margin for the relative risk. Additionally, let M_f and M_{nf} be the population median of the risk in the fixed and non-fixed condition, respectively. Then the null and alternative hypothesis are as follows:

$$H_0 : \frac{M_f - M_{nf}}{M_{nf}} \geq \delta \tag{7}$$

$$H_1 : \frac{M_f - M_{nf}}{M_{nf}} < \delta \tag{8}$$

We compute signed ranks for observations $\frac{x_f - x_{nf}}{x_{nf}} - \delta$. Let sr denote the absolute value of the sum of the negative ranks. Then the test statistic z is defined as follows:

$$z = \frac{sr - \left(\frac{N(N+1)}{4}\right)}{\sqrt{\frac{N(N+1)(2N+1)}{24}}} \tag{9}$$

For a single test, H_0 is rejected if $z \geq z_{1-\alpha}$, where $z_{1-\alpha}$ is the value of the standard normal distribution at significance level α. However, the experiment will be performed multiple times in this study, each time for a different hyperparameter. In order to control the family-wise error, we use the Holm-Bonferroni method (Holm, 1979).

4 Experiment details

In this section we will give an overview of the datasets, algorithms, performance measures, and experiment setups. All experiments are executed using dockerized Python applications[3] on the Azure Cloud Computing platform.

4.1 Datasets

We will use data from the open machine learning environment OpenML (Vanschoren et al., 2014). In particular, we use datasets from the OpenML-CC18, a curated machine learning benchmark suite of 73 classification datasets. A description of the properties of these datasets can be found in Appendix A.1. In this study, we used 59 of these 73 datasets. One dataset was not included in this study because of technical issues arising from the large number of nominal features in this dataset. Additionally, due to time constraints, we limited the training time of a single algorithm run on a dataset to 3 hours. Datasets for which more than 10% of the default value experiment runs lasted more than 3 hours are left out of the analysis. The 14 datasets that are excluded from the study are listed in Appendix A.2. It is important to note that all excluded datasets had a relatively high number of features, instances, or both.

4.2 Algorithms and Hyperparameters

In this work we consider two algorithms as implemented in `scikit-learn`, aliased `sklearn`. We consider a Support Vector Machine (SVM) with a Radial Basis Function (RBF) kernel and the Random Forest. The parameter ranges that are considered are taken from the automatic machine learning package `auto-sklearn` (Feurer et al., 2015). The ranges for SVM and random forest can be found in Table 1 and Table 2 respectively.

The total pipeline consists of simple pre-processing steps and the algorithm. The pre-processing pipeline includes missing value imputation by the mean (continuous features) or mode (categorical features), one hot encoding for categorical features, and removal of features with zero variance. Because SVM's are known to be sensitive to different feature scales, a scaling step is added to the SVM pre-processing pipeline.

[3] See `https://github.com/hildeweerts/hyperimp`.

Table 1: Hyperparameter ranges for `SVC`. Parameters annotated with * are fixed to a value different than the default of `scikit-klearn`.

hyperparameter	type	range
gamma	continuous	$[2^{-15}, 2^{3}]$ (log-scale)
C	continuous	$[2^{-}5, 2^{1}5]$ (log-scale)
tol	continuous	$10^{-5}, 10^{-1}]$ (log-scale)
shrinking	boolean	{True, False}
kernel*	discrete	rbf

Table 2: Hyperparameter ranges for `RandomForestClassifier`. Parameters annotated with * are fixed to a value different than the default of `sklearn`.

hyperparameter	type	range
bootstrap	boolean	{True, False}
criterion	nominal	{gini, entropy}
max_features	continuous	[0,1]
min_samples_leaf	integer	[1, 20]
min_samples_split	integer	[2, 20]
n_estimators*	integer	500

4.3 Performance Measures

We will compare two performance measures, as implemented in OpenML, at the default parameter determination stage: accuracy and macro-averaged Area under the ROC Curve (AUC). In macro-averaging, the measure is computed locally over each category first, then the average over all categories is taken, weighted by the number of instances of that class. The macro-averaged AUC is computed using the approach of Provost and Domingos (2000). In this approach, the ROC curve of each class is constructed by comparing the class of interest to the union of all other classes. Note that these curves can be sensitive to changes in prevalence within these classes. As multi-class AUC is not yet implemented in `sklearn`, accuracy will be used as a performance measure in the random search in the importance of tuning experiment.

4.4 Experiment 1: Default Values

In order to determine good default values, we need empirical performance measurements $\langle \boldsymbol{\theta}_i, y_i \rangle_{k=1}^{K}$ for each of the M datasets. In this experiment $M = 59$ and we choose $K = 1000$. In other words, for each dataset, we evaluate 1,000 random configurations $\boldsymbol{\theta}_i$ of each algorithm, i.e. 59,000 evaluations per algorithm. Each hyperparameter will be picked uniformly at random from the corresponding hyperparameter range and scale. The performance of a configuration is calculated using 10-fold cross validation. Recall that we use the best n settings for each task to determine the default parameters. In accordance to the work of van Rijn and Hutter (2017), we pick $n = 10$.

4.5 Experiment 2: Importance of Tuning

Recall that our goal is to compare two conditions: tuning all hyperparameters versus tuning all parameters but leave one fixed to the default value. We use nested cross validation to determine the performance of each of these situations. In the outer loop, we use 10-fold cross validation to split the data in training and test sets. In the inner loop, we use 5-fold cross validation to split the data further into training and validation sets. For each fold in the inner loop, a random search strategy with 100 iterations is applied to tune hyperparameters. Note that the hyperparameter settings that are used to determine test set performance can be different for each of the 10 folds in the outer cross validation. We repeat the experiment $R = 10$ times, using a different seed for the random search each time.

5 Results

In this section we will discuss the results of our experiments. First, we will discuss the results of the default value experiment. Second, we discuss the results from the hyperparameter tuning importance experiments. All performance data that was used in this study is publicly available on OpenML[4].

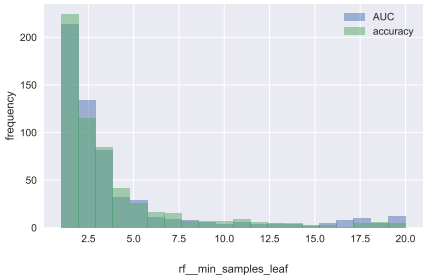
5.1 Experiment 1: Default Values

In this section, we first discuss the collected performance data. Subsequently, we discuss the computed default values. In particular, we compare our computations with the default values computed by Probst et al. (2018) and the default values used in `sklearn`. Finally, we discuss the determination of meta-feature dependent default parameters.

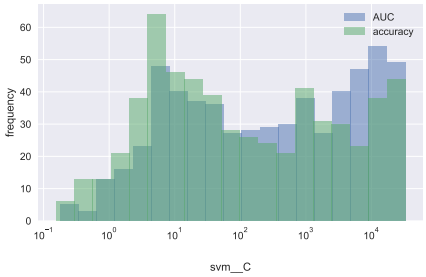
[4] See `https://www.openml.org/s/98/`.

5.1.1 Performance data

As stated before, the first step of our default setting determination procedure is collecting top 10 performance data. The distribution of the hyperparameter settings of the top 10 performance data can be visualized in a histogram, as illustrated in Figure 1. For example, from the distribution depicted in Figure 1a, we can conclude that fixing *min_samples_leaf* to 1 resulted in good performance very often. As stated in section 4.3, we determined the top 10 performance data based on both accuracy and AUC. For most hyperparameters, the distribution of accuracy and AUC based data is very similar. For others, one may arrive at a different default parameter depending on the performance measure that was used. For instance, consider the distribution of C shown in Figure 1b. An accuracy based default value for C would be in the order of magnitude 1, whereas an AUC based default value would be in the order of magnitude 4. In future research, it would be interesting to investigate this difference further. The histograms of all other hyperparameters can be found in Appendix B.



(a) Random Forest - *min_samples_leaf*



(b) SVM - C

Figure 1: The distribution of *min_samples_leaf* and C within the top 10 highest performing hyperparameter settings per dataset, derived from either accuracy or AUC based performance data.

5.1.2 Default values

Recall that the default values are computed for each task separately, using a leave-one-out approach. The average and standard deviation of the computed default values are displayed in Table 3. In the remainder of this section, we discuss our computed default values and compare them to the default values implemented in `sklearn` and those computed by Probst et al. (2018).

For most hyperparameters, the standard deviation of the computed default values across datasets is zero or relatively small, considering the scaling. For example, the standard deviation of C using an accuracy based approach is 3.224, which is negligible on a logarithmic scale. The distribution of default values with non-zero variance are displayed in Appendix C.

The differences between defaults resulting from the accuracy and AUC approach are in line with the histograms shown in the previous section. For *bootstrap*, *criterion*, *min_samples_split*, *C*, *tol* we find different default parameters than the ones currently used in `sklearn`, although it should be noted that the default parameter of *min_samples_split* in `sklearn` was returned often by the AUC based approach. For *bootstrap* and *gamma*, we find values in a similar order of magnitude as Probst et al. (2018). On the other hand, the default values for C differ a lot between the accuracy based defaults, AUC based defaults, the default used in `sklearn`, and the defaults computed by Probst et al. (2018).

5.1.3 Meta-feature dependent hyperparameters

As stated in Section 3, we will determine the hyperparameters for *max_features* and *gamma* using an approach that considers the number of features of a dataset. Figure 2 shows a scatter plot of the feature versus the number of features, including the functions that that were fitted. The Root Mean Squared Error (RMSE), R^2, Root Mean Squared Logarithmic Error (RMSLE), and logarithmic R^2 (LR^2) for each of the functions can be found in TableZ4.

Recall that we denote the number of features as

Table 3: Average/Mode (numerical/categorical) and standard deviation of our calculated defaults, defaults of `sklearn`, and defaults as calculated by Probst et al. (2018). Hyperparameters that are not computed in the work of Probst et al. (2018) (e.g. because they do not exist in the R implementation) are indicated with N/A. The standard deviation that is shown for *max_features*, annotated with *, is the standard deviation of the exponent. The standard deviation that is shown for boolean variables, annotated with **, is the standard deviation when considering *True* as 1 and *False* as 0. The standard deviation shown for *criterion*, annotated with ***, is the standard deviation when considerint *entropy* as 1 and *gini* as 0.

Hyperparameter	Average/Mode (accuracy)	Standard deviation (accuracy)	Average/Mode (AUC)	Standard deviation (AUC)	sklearn	Probst et al. (2018)
Random Forest						
bootstrap	False	**0	False	**0	True	False
criterion	entropy	***0	entropy	***0	gini	N/A
max_features	nˆ0.737	*0.009	nˆ0.714	*0.006	sqrt(n)	0.284
min_samples_leaf	1	0	1	0	1	0
min_samples_split	5	0	2.3	0.8	2	N/A
SVM						
C	8.888	3.224	16646	63	1.000	475.504
gamma	1.2 e-02	1.5 e-04	8.1 e-03	1.5 e-03	1/n	5 e-03
shrinking	True	**0	True	**0.36	True	N/A
tol	4.6 e-05	8.8 e-08	5.4 e-02	1.3 e-02	1.0 e-03	N/A

Table 4: RMSE, R^2, RMSLE, and LR^2 of several functions fitted on the accuracy based top 10 performance data for *max_features* and *gamma*.

function	RMSE	R^2	RSMLE	LR^2
max_features				
m = 0.16*p	21.64	0.77	0.90	0.13
m = p ˆ0.74	23.13	0.73	0.73	0.42
m = 1.15ˆsqrt(p)	24.33	0.70	1.24	-0.66
m = pˆ0.25	47.16	-0.11	1.39	-1.10
gamma				
m = 0.00574*p	1.25	-0.07	0.46	-0.12
m = 1/p	1.24	-0.05	0.44	-0.04
m = 0.006	1.27	-0.09	0.46	-0.14

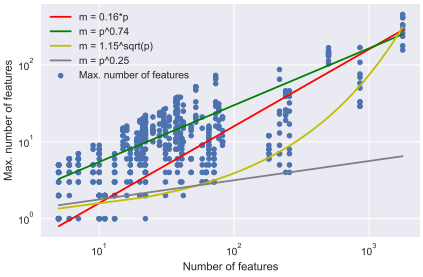
p. From Figure 2a, it becomes clear that for our datasets, the default value in `sklearn`, $\sqrt{p}$, often underestimates *max_features*. In fact, it performs worse than all other functions that we considered. On a linear scale, $m = 0.16^p$ performs best. However, RMSE and R^2 are slighty biased, because there are relatively few large datasets. When we consider the RSMLE and LR^2 instead, $m = p^{0.74}$ performs best. In Figure 2b, we do not observe a clear pattern for *gamma*. Additionally, we observe negative values for both R^2 and LR^2 for all functions in Table 4. The default value in `sklearn`, $1/p$, performs slightly better than the other two functions.

In experiment 2, where we investigate the importance of tuning, we will consider $\sqrt{p}$ and $p^{0.74}$ for *max_features*, and $1/p$ and 0.006 for *gamma*.

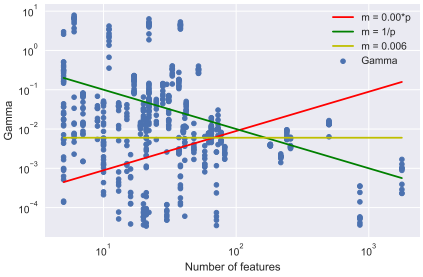
5.2 Experiment 2: Importance of Tuning

In order to investigate the tuning process, we visualize the average rank of the fixed and non-fixed condition. For each observation, the rank is computed based on the maximum average validation set accuracy observed up until a certain iteration of the random search. For each measurement, for each iteration, the condition with the highest validation accuracy receives rank 1 and the other rank 2. As an example, Figure 3 depicts the average rank over number of iterations for *max_features*, using our computed default value. The visualizations of the other hyperparameters can be found in Appendix D.

For all hyperparameters except SVM's *shrinking*, the average rank of the fixed condition is lower than the average rank of the non-fixed condition in the first iteration. For many hyperparameters, the fixed condition, on average, outperforms the non-fixed condition for all 100 iterations. This indicates that 100 iterations was not enough to find the best possible hyperparameter settings of the algorithm. For SVM's *shrinking* and *tol*, we observe that the average rank is consistently close to 1.5, which indicates that neither the fixed nor the non-fixed condition was superior. For random forest's *max_features* (using the computed default value), SVM's *C*, SVM's *gamma* (both computed default value and `sklearn`'s default value), the non-fixed condition,



(a) Random Forest - *max_features*



(b) SVM - *gamma*

Figure 2: Meta-feature dependent parameter values of the top 10 performance data (accuracy based only) against the number of features (p) of the corresponding dataset.

on average, outperforms the fixed condition after 15 to 30 iterations.

5.2.1 Tunability

The average and standard deviation of both tunability and relative tunability are shown in Table 5. Additionally, Figure 4 shows the distribution of tunability for each of the hyperparameters. There exist three datasets for which the non-fixed SVM almost always resulted in perfect accuracy: OpenML task 11, 49, and 10093. Because relative tunability is undefined when the non-fixed risk is equal to 0, these tasks are left out of the tunability analysis.

From Table 5, we observe that for *bootstrap*, *criterion*, *min_samples_leaf*, *min_samples_split*, *shrinking*, and *tol*, both the tunability and relative tunability are close to or lower than zero. This is in line with the

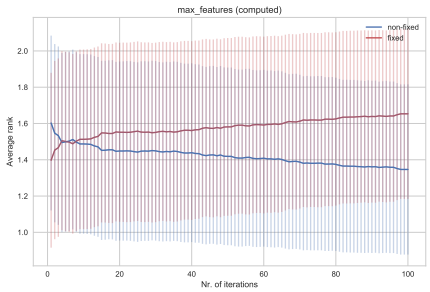


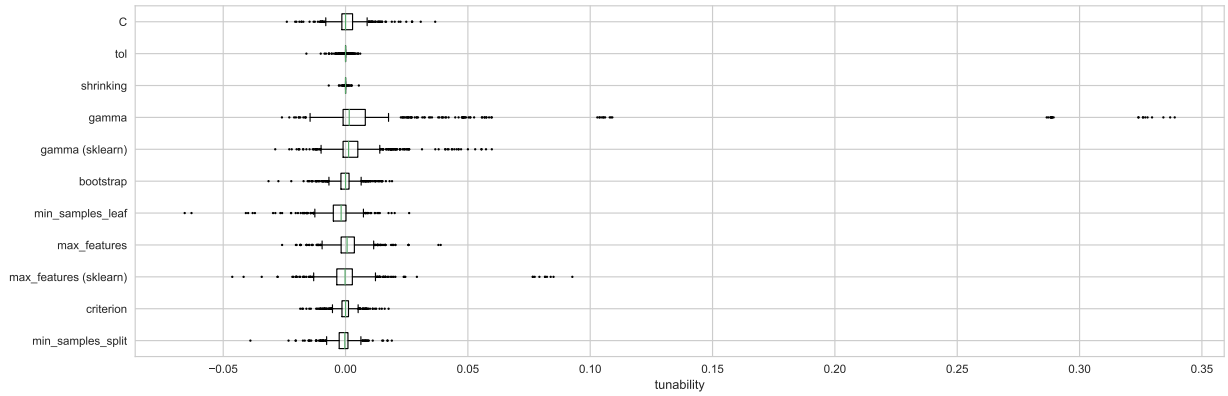
Figure 3: Average rank (+/- standard deviation) of the the fixed and non-fixed condition over number of iterations for hyperparameter *max_features* (computed default setting).

Table 5: Average and standard deviation of tunability (d_i, s_i) and relative tunability (d_i^R, s_i^R).

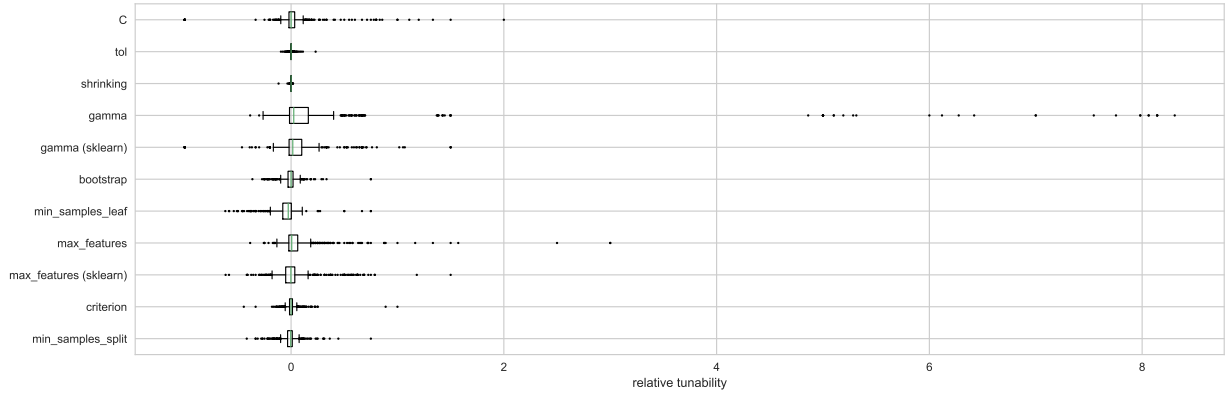
Hyperparameter	d_i	s_i	d_i^R	s_i^R
Random Forest				
bootstrap	-0.0002	0.0053	-0.0053	0.0820
criterion	-0.0002	0.0041	0.0035	0.0793
max_features	0.0011	0.0068	0.0742	0.2822
max_features (sklearn)	0.0010	0.0134	0.0309	0.2037
min_samples_leaf	-0.0030	0.0077	-0.0505	0.1280
min_samples_split	-0.0009	0.0048	-0.0110	0.0829
SVM				
C	0.0008	0.0062	0.0252	0.2415
gamma	0.0180	0.0599	0.4595	1.4687
gamma (sklearn)	0.0039	0.0115	0.0503	0.2498
shrinking	0.0000	0.0005	-0.0003	0.0057
tol	-0.0001	0.0016	0.0006	0.0179

average ranks shown in the previous section.

In Figure 4a, we observe a few high tunability scores for *gamma* (computed default) and *max_features* (`sklearn` default). This indicates that there exist a few datasets for which the fixed condition performed much worse than the non-fixed condition. This effect is even larger for *gamma* (computed default) when considering relative tunability. The tunability of *gamma* when using `sklearn`'s default parameter is much better than the tunability of *gamma* when using our not meta-feature dependent default setting. For *max_features*, our computed default value resulted in similar tunability but worse relative tunability compared to using `sklearn`'s default value. In particular, there are several outliers with a relative tunability higher than 2. This is not what we expected given the analysis on meta-feature dependent parameter values in Section 5.1.3.



(a) Tunability



(b) Relative tunability

Figure 4: The distribution of tunability and relative tunability measurements ($d_{i,j,r}$) per hyperparameter.

5.2.2 Non-inferiority test

The test statistics and p-values associated with non-inferiority tests with a non-inferiority margin of $\delta = 0.01$ are shown in Table 6. To determine which null hypotheses are rejected, we used the Holm-Bonferroni method with a significance level of $\alpha = 0.05$.

The results show that for all hyperparameters, except for *max_features* (computed default), *C*, and *gamma* (both computed and `sklearn` default), the relative risk is not more than 1% higher in the fixed condition compared to the non-fixed condition. This indicates that for these hyperparameters, setting the hyperparameter to the default value is non-inferior to tuning the parameter through a random search using 100 iterations. For *C*, *gamma*, and *max_features* (computed default) this is not the case.

6 Conclusions

In this study, we have first presented a simple heuristic to find default hyperparameters. In our experiments, we touched upon determining meta-feature dependent default parameters. Additionally, we presented a methodology for determining the importance of tuning a hyperparameter empirically. In contrast to

Table 6: Test statistics, p-values, and conclusions of non-inferiority tests. Conclusions are obtained through the Holm-Bonferroni method (Holm, 1979).

Hyperparameter	z	p	H_0
Random Forest			
bootstrap	7.93	1.11 e-15	rejected
criterion	8.16	1.11 e-16	rejected
max_features	-2.02	9.78 e-01	not rejected
max_features (sklearn)	3.59	1.63 e-04	rejected
min_samples_leaf	15.04	0.00 e+00	rejected
min_samples_split	10.06	0.00 e+00	rejected
SVM			
C	1.83	3.40 e-02	not rejected
gamma	-8.24	1.00 e+00	not rejected
gamma (sklearn)	-4.61	1.00 e+00	not rejected
shrinking	20.49	0.00 e+00	rejected
tol	16.32	0.00 e+00	rejected

previous work, our approach can be used to determine the loss incurred when one of the hyperparameters is not tuned but set to a default value. This is different from the study of Probst et al. (2018), who investigate the performance gain of leaving all hyperparameters to a default value and tuning one hyperparameter. We have applied our methodology in a benchmark study using 59 different datasets. In this way, we provide empirical evidence that can be consulted by machine learning practitioners before they start a computationally and time intensive tuning process.

Our results show that using our computed default value often results in non-inferior performance compared to tuning the hyperparameter. It should be noted that the number of iterations turned out to be too low to tune several configurations of the algorithms to the best possible performance in the non-fixed condition. Although this indicates that our default parameters were reasonable, the number of iterations could be increased in future work. For other hyperparameters, such as random forest's *max_features* and SVM's *gamma* and *C*, we observed a high tunability and relative tunability, which suggests that it is important to tune these hyperparameters.

A limitation of our work is that the default parameters are only determined once and are not validated separately. As a result, it is unclear how our simple default parameter estimation method affects the results of the second experiment. In particular, it is unclear whether our choice of n results in a good representation of hyperparameters with good performance. This could be resolved in future work by comparing different methods for finding default parameters.

Another limitation is that we excluded several datasets from the OpenML-CC18 that turned out to have a high number of instances and/or features. It is unclear how this affects our conclusions, in particular for the determination of meta-feature dependent default values. A related issue is that it is unclear whether the OpenML-CC18 is a good representation of datasets that machine learning practitioners encounter in real life. Our conclusions might not hold for datasets that are very different from the ones analyzed in this work.

Finally, it is important to note that we have only considered leaving a single hyperparameter at a default value. In future work, interactions between hyperparameters could be further investigated.

References

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

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., and Hutter, F. (2015). Efficient and robust automated machine learning. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems 28*, pages 2962–2970. Curran Associates, Inc.

Freedman, D. and Diaconis, P. (1981). On the histogram as a density estimator:l2 theory. *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete*, 57(4):453–476.

Holm, S. (1979). A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6(2):65–70.

Lakens, D. (2017). Equivalence tests. *Social Psychological and Personality Science*, 8(4):355–362.

Mara, C. A. and Cribbie, R. A. (2012). Paired-samples tests of equivalence. *Communications in Statistics - Simulation and Computation*, 41(10):1928–1943.

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. *Journal of Machine Learning Research*, 12:2825–2830.

Probst, P., Bischl, B., and Boulesteix, A.-L. (2018). Tunability: Importance of hyperparameters of machine learning algorithms.

Provost, F. and Domingos, P. (2000). Well-trained pets: Improving probability estimation trees.

Sturges, H. A. (1926). The choice of a class interval. *Journal of the American Statistical Association*, 21(153):65–66.

van Rijn, J. N. and Hutter, F. (2017). Hyperparameter importance across datasets.

Vanschoren, J., van Rijn, J. N., Bischl, B., and Torgo, L. (2014). OpenML. *ACM SIGKDD Explorations Newsletter*, 15(2):49–60.

Walker, E. and Nowacki, A. S. (2010). Understanding equivalence and noninferiority testing. *Journal of General Internal Medicine*, 26(2):192–196.

A Datasets

A.1 Properties OpenML-CC18

The OpenML-CC18 is a curated benchmark suite consisting of 73 classification tasks. For each dataset, the following properties hold:

- The number of observations is larger than 500;
- The number of observations is smaller than 100,000;
- The ratio of the minority class and the majority class is larger than 0.05;
- The number of values for categorical features most not exceed 100;
- The classification problem is not trivial (a model based on 1 feature does not result in perfect performance);
- Each target class contains at least 20 instances;
- The dataset does not belong to one of the following categories:
  - Artificial dataset
  - Simulated dataset
  - Time series dataset
  - Text data
  - Multilabel data
  - Derived versions of another dataset
  - Dataset where the intended classification target is unclear
  - Binarized regression problem
  - Dataset of unknown origin
  - Grouped data

A.2 Excluded tasks

The tasks of OpenML-CC18 that were not included in this study, due to either time constraints or technical issues, are listed in Table 7. Additionally, the number of features and number of instances of the datasets are shown in Figure 5. It is clear that the datasets that were excluded are all datasets with relatively many features or instances.

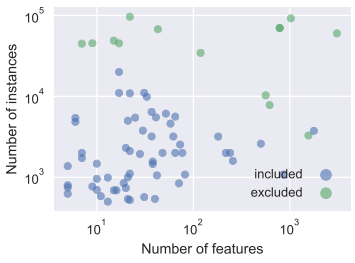


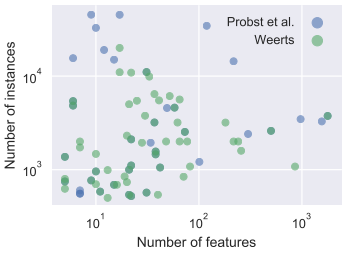
Figure 5: Number of features against number of instances of datasets of the OpenML-CC18 that were included and excluded from this study.

Table 7: OpenML-CC18 tasks excluded from this study. Tasks for which only the random forest or SVM timed out too often are annotated with respectively * and **.

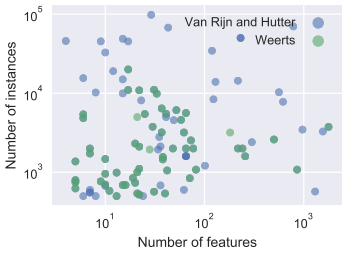
OpenML task id	Reason
167125	technical issue
219**	time constraint
3481*	time constraint
3573	time constraint
7592	time constraint
9977	time constraint
14965**	time constraint
14970*	time constraint
146195	time constraint
146825	time constraint
167119**	time constraint
167120	time constraint
167121	time constraint
167124	time constraint

A.3 Datasets previous work

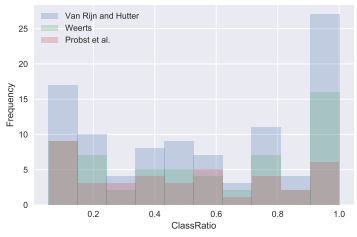
In Figure 6a we observe that both the study of Probst et al. (2018) and van Rijn and Hutter (2017) contain more datasets with a large number of features and instances. In Figure 6c we observe that the data of Probst et al. (2018) and van Rijn and Hutter (2017) contain relatively more imbalanced dataset than ours. This makes sense, since these datasets are taken from the OpenML-100, for which the class imbalance was not yet one of the criteria (as opposed to the OpenML-CC18).



(a) The number of features against number of instances for this work and Probst et al. (2018).



(b) The number of features against number of instances for this work and van Rijn and Hutter (2017).

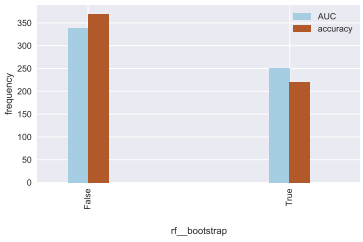


(c) Distribution of the ratio of the number of instances in the minority class compared to the majority class.

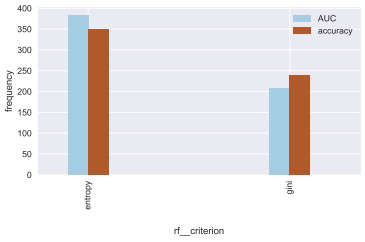
Figure 6: Meta-features for the datasets included in the work of Probst et al. (2018), van Rijn and Hutter (2017) and this study.

B Distribution of top 10 performance data

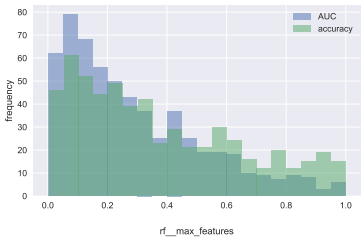
In Figure 7 histograms of the distribution of hyperparameters in the top 10 performance data are displayed for both accuracy and AUC.



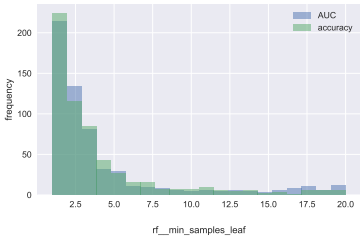
(a) Random Forest - *bootstrap*



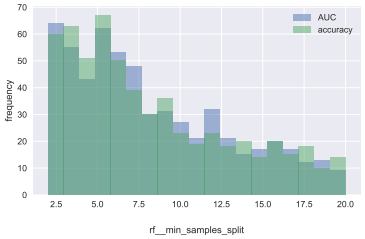
(b) Random Forest - *criterion*



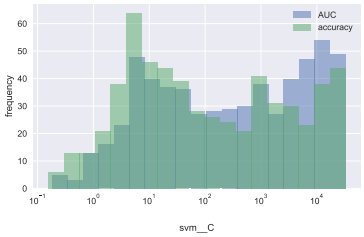
(c) Random Forest - *max_features*



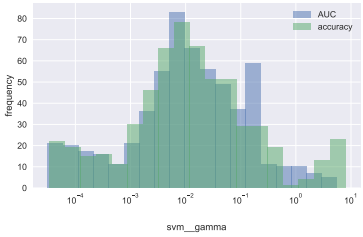
(d) Random Forest - *min_samples_leaf*



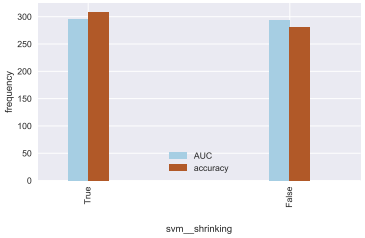
(e) Random Forest - *min_samples_split*



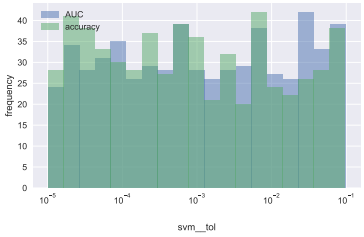
(f) SVM - *C*



(g) SVM - *gamma*



(h) SVM - *shrinking*

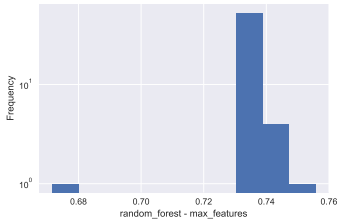


(i) SVM - *tol*

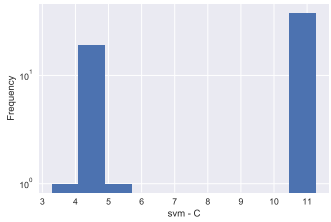
Figure 7: The distribution of a hyperparameters within the top 10 highest performing hyperparameter settings per dataset, measured by either accuracy or AUC.

C Distribution of default values across tasks

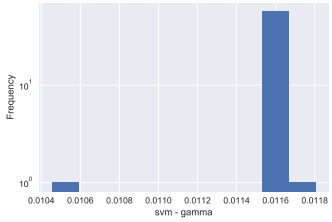
The distributions of the computed default values for all hyperparameters for which the standard deviation of the default values is non-zero is depicted in Figure 8 and Figure 9 derived from respectively accuracy and AUC based performance data.



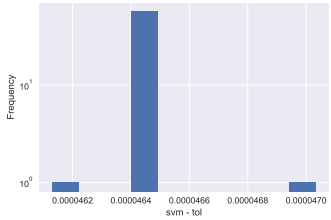
(a) Random Forest - *max_features*



(b) SVM - *criterion*

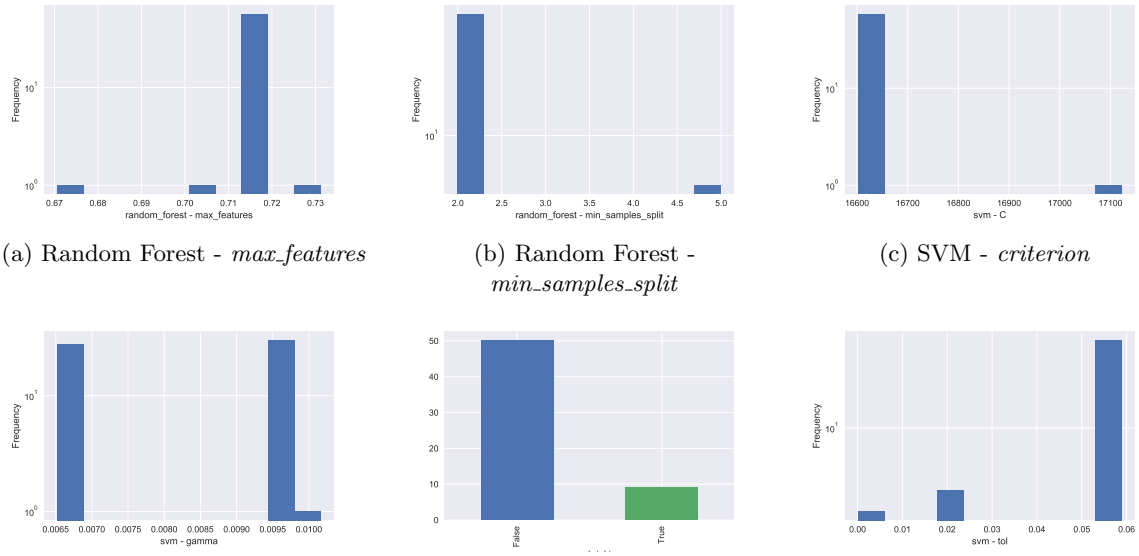


(c) SVM - *gamma*



(d) SVM - *tol*

Figure 8: The distribution of the computed default values, using **accuracy** based performance data, across different tasks for hyperparameters where the default values have a non-zero standard deviation.



(a) Random Forest - *max_features*

(b) Random Forest - *min_samples_split*

(c) SVM - *criterion*

(d) SVM - *gamma*

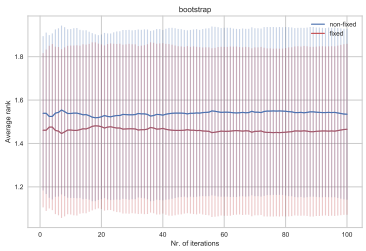
(e) SVM - *shrinking*

(f) SVM - *tol*

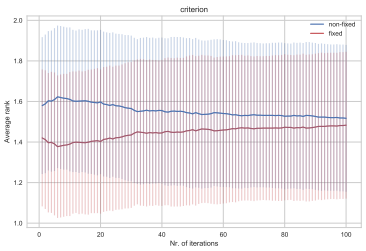
Figure 9: The distribution of the computed default values, using **AUC** based performance data, across different tasks for hyperparameters where the default values have a non-zero standard deviation.

D Average rank

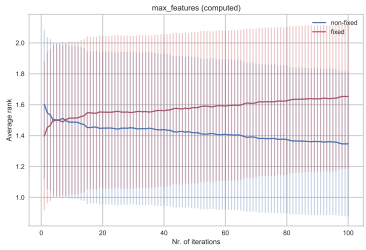
For each measurement, we computed the maximum average validation set accuracy up until a certain iteration. This data was used to rank the fixed and non-fixed condition. The average rank (+/- standard deviation) is shown in Figure 10. Note that ranks are a relative measure of performance and that smaller is better.



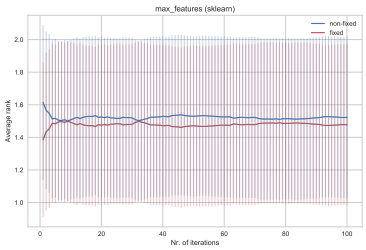
(a) Random Forest - *bootstrap*



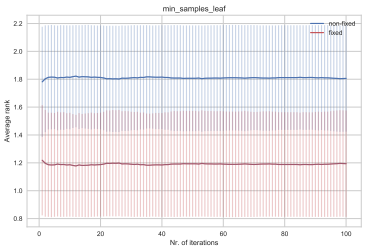
(b) Random Forest - *criterion*



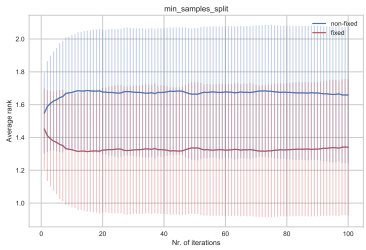
(c) Random Forest - *max_features* (computed default)



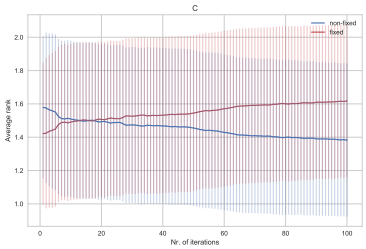
(d) Random Forest - *max_features* (sklearn default)



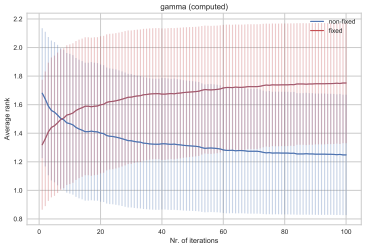
(e) Random Forest - *min_samples_leaf*



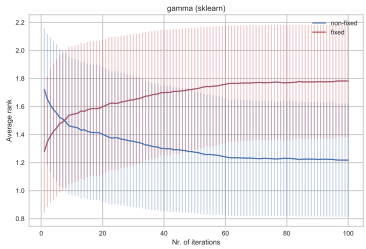
(f) Random Forest - *min_samples_split*



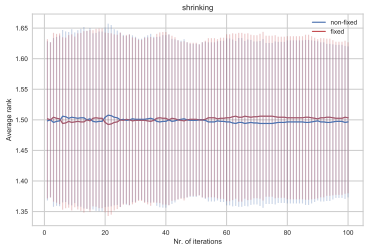
(g) SVM - *C*



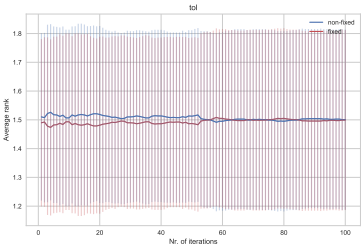
(h) SVM - *gamma* (computed default)



(i) SVM - *gamma* (sklearn default)



(j) SVM - *shrinking*



(k) SVM - *tol*

Figure 10: Average rank (+/- standard deviation) of the the fixed and non-fixed condition across 59 datasets over number of iterations. Ranks are based on the maximum average validation set accuracy. Note that ranks are a relative measure of performance and that smaller is better.