17

Learning from Metadata in Repositories

Summary. This chapter describes the various types of experiments that can be done with the vast amount of data, stored in experiment databases. We focus on three types of experiments done with the data stored in OpenML. First, we describe experiments that determine how a certain algorithm works on a given dataset. We discuss experiments that show which algorithm performs best on a given dataset and experiments determining the effect of a given hyperparameter on the performance. Second, we describe experiments that determine how a certain algorithm works across datasets. We discuss experiments that determine the statistical significance of differences of performance, the effects of hyperparameter optimization of chosen algorithms across datasets, and experiments that determine which algorithms generally make similar predictions. Finally, we describe experiments that determine the effect of certain data or workflow characteristics on the performance. By taking into consideration the various metafeatures that have been calculated, we can explore trends when a given type of algorithm (e.g., linear models, models including feature selection) performs better than another.

17.1 Introduction

OpenML described in the previous chapter (Chapter 16) includes information about a vast set of experiments, which have been collected and organized in a structured way. This allows to explore the data to conduct various studies and this way gain useful knowledge about how algorithms behave. As Vanschoren et al. (2012) pointed out, the studies can be divided into the following groups, depending on what the overall aim is:

- Model-level analysis, whose main aim is to analyze *how* a given algorithm performs
- Data-level analysis, whose aim is to investigate *when* an algorithm performs well
- Method-level analysis, whose aim is to determine *why* an algorithm performs the way it does

This chapter follows loosely the groups detailed above. Section 17.2 describes how the performance of different algorithms or their configurations varies on some datasets. Section 17.3 extends this study to determine how the performance varies across datasets. Section 17.4 presents studies whose aim is to investigate the interplay between the performance of algorithms and measurable features (e.g., algorithm properties or metafeatures) across datasets.

P. Brazdil et al., *Metalearning*, Cognitive Technologies,
https://doi.org/10.1007/978-3-030-67024-5_17

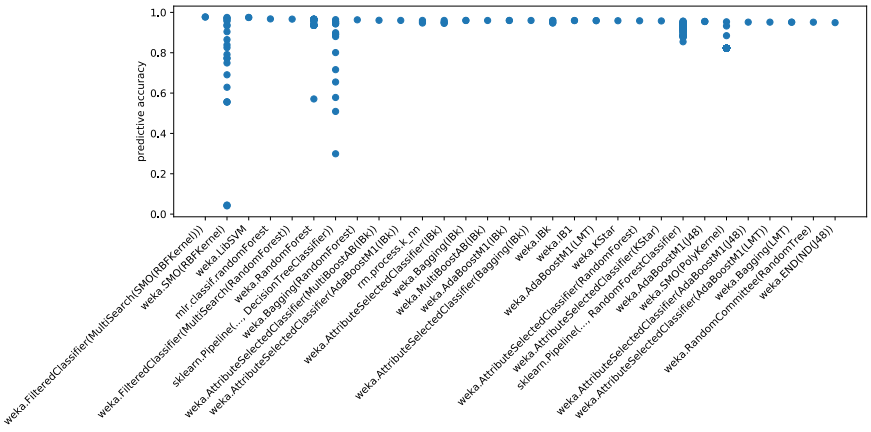


Fig. 17.1: Performance of various algorithms on the "letter" dataset

One aim of these studies is to broaden our understanding of the way algorithms behave on certain types of data. Another aim is to show some simple examples that can be reproduced in other settings (e.g., when new algorithms become available).

17.2 Performance Analysis of Algorithms per Dataset

In this section we present two studies. In the first one, we show how the performance of different algorithms varies on some datasets. In the second one, the focus is on the effects of different hyperparameter settings on performance.

17.2.1 Comparing different algorithms

In this study, the aim was to examine how different algorithms perform on the task of classifying letters in different fonts. The data is available in the UCI dataset "letter" (Frey and Slate, 1991). This task has 26 classes (each letter from the alphabet is a class), and each case is described using a set of predefined attributes.

The results of this study are shown in Figure 17.1. The x-axis shows a particular algorithm in OpenML nomenclature (*flow*),[1] and the y-axis represents the predictive accuracy. Each dot represents a particular variant with a certain parameter setting. The figure contains algorithms (flows) from various classification workbenches, i.e., Weka, mlR, and Scikit-learn. In order not to overload the image, only the 30 best-performing algorithms (flows) are shown. Ensemble methods are grouped by the base-learner that is used (e.g., bagging k-NN is considered a different flow than bagging J48).

The figure shows that kernel-based methods perform fairly well on this particular dataset. Among the top-performing algorithms are many variants of support vector machines (SVMs), random forests, and instance-based methods. The figure also shows that

[1] A *flow* is an implementation of an algorithm or workflow. See Chapter 16 for details.

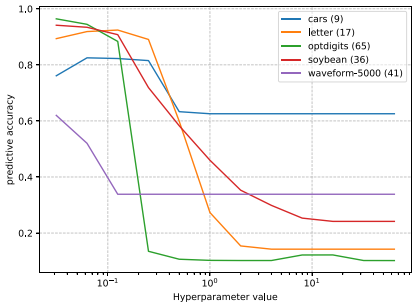


Fig. 17.2: The effect of varying SVM gamma hyperparameter

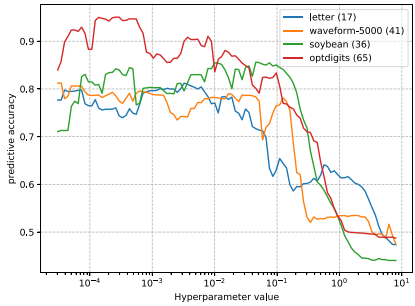


Fig. 17.3: Marginal of the SVM gamma hyperparameter

the performance of some algorithms (flows) (e.g., weka.SMO(RBFKernel)) has a great variation, as the hyperparameter settings for this flow span across a wide range.

17.2.2 Effect of varying some hyperparameter settings

In this section we present a study of the effects of varying certain hyperparameter settings on performance. We focus on the effects of the *gamma* parameter of SVM with the RBF kernel, as implemented in Weka. All the other hyperparameters were left with their respective default settings.

Figure 17.2 shows these effects for some datasets only. As we can see, the performance curves follow a somewhat similar trend for some datasets. In the case of "waveform", "soybean", and "optdigits", the performance degrades to the default accuracy. In the case of "letter" and "car", the performance first increases until it has reached an optimum and then degrades to the default accuracy. The optimal value of the parameter is different for all datasets used in this study, as would be expected. It seems that setting this value too low is less harmful for performance than setting it too high.

This study shows a local effect of how a particular hyperparameter affects performance under the assumption that all other hyperparameters use a fixed setting (default values).

Global effect of a hyperparameter, based on a marginal

A *marginal* defines, for each hyperparameter, the expected performance when all other hyperparameters have been averaged across all possible values. Although this seems infeasible to compute, Hutter et al. (2014) showed that it can be computed efficiently by using tree-based *surrogate models*.

Figure 17.3 plots the marginal of the gamma parameter of support vector machines with the RBF kernel, as implemented in Scikit-learn, on several datasets. Note that the values of the marginal are lower than the values in the plot that shows the local effect. This is because these are averaged across all possible values for the other hyperparameters, which likely involve also many suboptimal values. This shows a global effect of how the hyperparameter gamma affects performance. Using the marginal solves the question regarding which value the other hyperparameters should be set to.

In Chapter 8, we discussed the work of van Rijn and Hutter (2018), who showed how one can use the marginal to determine which hyperparameters are important to optimize.

17.3 Performance Analysis of Algorithms across Datasets

In this section we discuss experiments whose aim is to analyze how the performance of different algorithms (and/or configurations) varies across different datasets. We discuss three types of experiments. The first one concerns the benchmarking of algorithms, the second one studies the effect of hyperparameter optimization, and the third type analyzes how several algorithms differ in predictions.

17.3.1 Effect of using different classifiers with default hyperparameters

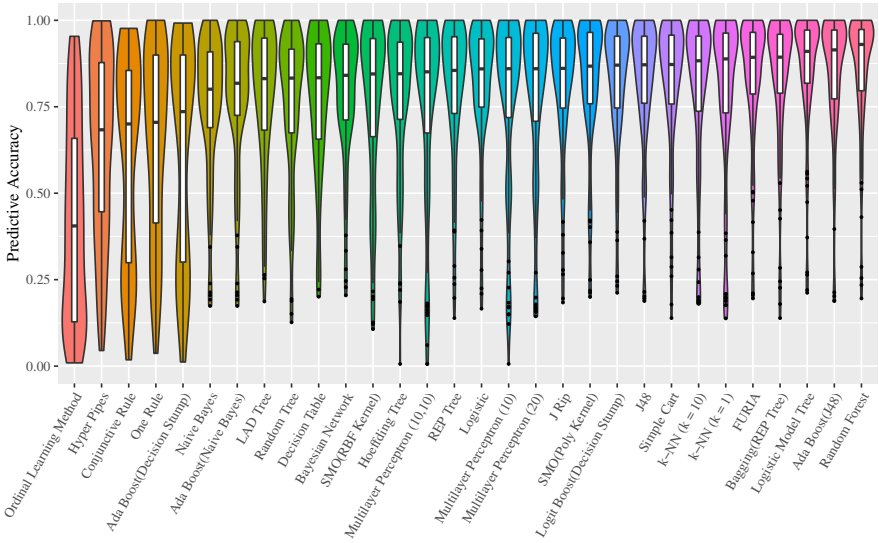
In order to assess the performance of an algorithm in a robust way, the benchmarking and evaluation should involve a large number of datasets. In this section we discuss a study to illustrate this. We have selected a set of classifiers and datasets from OpenML and the corresponding metadata, which includes the predictive accuracy and area under the ROC curve (AUC).

As the performance of a particular algorithm varies on different datasets, it is possible to elaborate violin plots, which are similar to box plots, except that they also show the probability density of the data at different values, smoothed by a kernel density estimator. Figure 17.4 shows violin plots (with integrated box plots) of the results of various Weka classifiers with default hyperparameter settings across 105 datasets. The classifiers are sorted by the median. Classifiers to the right perform generally better than classifiers to the left. Random forest (Breiman, 2001) performs best on average on this set of datasets. Some other ensemble methods perform well, e.g., adaptive boosting (Freund and Schapire, 1996) and logistic boosting (Friedman et al., 1998). Logistic model tree (LMT) (Landwehr et al., 2005) (which is a combination of trees and logistic regression) also performs reasonably well.

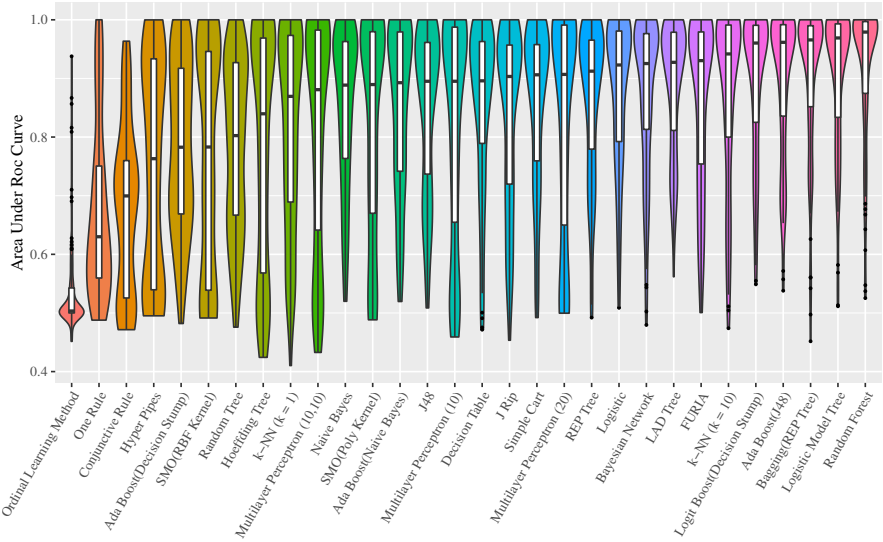
Note that, when a classifier is ranked low, it does not necessarily mean that is it a badly performing classifier overall. It might very well be that the classifier is specialized on a certain type of dataset, or just has hyperparameters that need to be tuned. For example, we note that the support vector machine with the RBF kernel is not well positioned in this ranking. However, as we saw in Figure 17.1, some variants of this classifier (variants with particular hyperparameter settings) are among the best performing ones on the "letter" dataset. As such, it is rare in modern applications to benchmark algorithms without applying proper hyperparameter optimization.

Assessing statistical significance

To assess the statistical significance of the above results, we can use the Friedman test accompanied by the posthoc Nemenyi test, which were discussed in Chapter 3. Figure 17.5 shows the result of the Nemenyi test on the predictive accuracy of the classifiers from Figure 17.4. The classifiers are sorted by their average rank (lower is better). We see a similar ordering of classifiers as in Figure 17.4. Here, we also see that the logistic model tree and random forest perform best.



(a) Accuracy



(b) Area under the ROC curve

Fig. 17.4: Ranking of algorithms across a set of 105 datasets. This image was taken from van Rijn (2016)

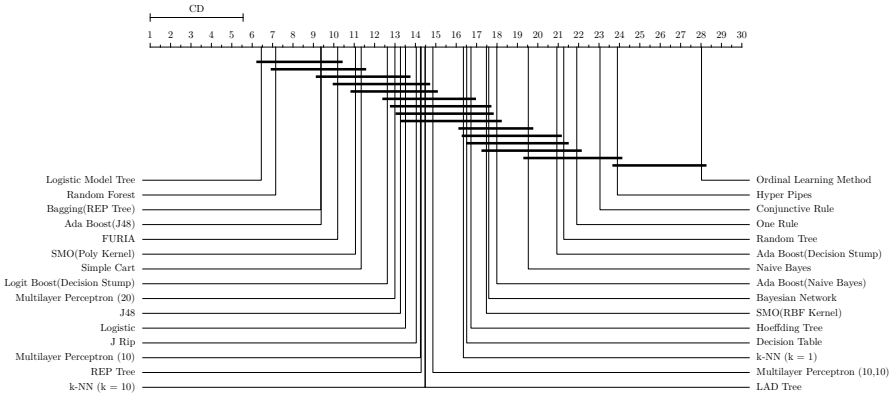


Fig. 17.5: Results of Nemenyi test ($\alpha = 0.05$) on the predictive accuracy of classifiers in OpenML. This image was taken from van Rijn (2016)

Classifiers that are connected by a horizontal line are statistically equivalent. For instance, the figure shows that there is no statistical evidence that logistic model tree is better than FURIA. On the other hand, there is statistical evidence that the logistic model tree is better than the "Simple CART" algorithm.

When applied correctly, statistical tests give a more reliable assessment of the performance of algorithms than simply comparing performance values (see Chapter 3 for details).

17.3.2 Effect of hyperparameter optimization

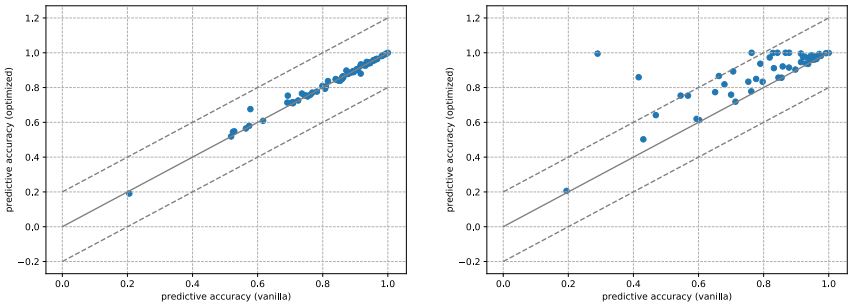
It has been widely recognized that the performance of algorithms is affected heavily by hyperparameter optimization (Lavesson and Davidsson, 2006; Hutter et al., 2011; Bergstra and Bengio, 2012; Snoek et al., 2012; Domhan et al., 2015; Klein et al., 2017; Li et al., 2017; Thomas et al., 2018; Falkner et al., 2018). This section describes an experiment showing how OpenML can be exploited to investigate this relationship.

Different search strategies exist for identifying the best set of hyperparameter settings for a given algorithm. Several of those are discussed in Chapter 6.

In order to obtain an unbiased recommendation, it is common to follow the nested cross-validation procedure (see Chapter 3), which splits the data into a training set, validation set, and test set. The training set is used to train various variants with different hyperparameter settings. These variants are then evaluated on the validation set. The model that performs best on the validation set is recommended, and its performance is determined on the test set, representing an *unseen data sample*.

OpenML uses the notion of *tasks* to define the training set and test set; it is up to the user to split off a part from the training set into a validation set.

In the following illustrative example, we compare two Weka classifiers with optimized hyperparameters against the corresponding versions with default hyperparameters across different datasets. Figure 17.6 shows the results of a decision tree and a



(a) Decision tree (b) Support vector machine (SVM)

Fig. 17.6: Comparing performance of optimized hyperparameter of two classifiers against the default configuration

support vector machine (SVM) with the RBF kernel. For each dataset, both classifiers were applied twice; once with default hyperparameters and the second time with hyperparameter settings obtained by a hyperparameter optimization method (Weka's MultiSearch module). The results are presented in a scatter plot. Each dot represents the performance of both the default hyperparameters and optimized hyperparameters on a specific dataset. The performance of the default hyperparameters is displayed on the x-axis, and the performance of the optimized hyperparameters on the y-axis.

The diagonal solid line shows the break-even points; all points on this line represent an experiment for which hyperparameter optimization neither benefited, nor harmed the predictive performance. Each dot that is above (below) the solid line represents a dataset on which the version of the classifier with optimized hyperparameters performs better (worse) than the non-optimized version. Note that it is possible that hyperparameter optimization can deteriorate performance. The selection of a configuration is based on the performance of this configuration on the validation set; the selected configuration may not perform as well on the test set. However, this should not happen too often.

The plot reveals that support vector machines often benefit from hyperparameter optimization. For almost all datasets, the version with optimized hyperparameters outperforms the version with standard hyperparameters. For the decision tree algorithm this benefit is less apparent. The effect of this hyperparameter optimization method seems negligible on most datasets. We base this on the fact that most data points are scattered around the aforementioned diagonal line.

Most state-of-the-art algorithms, such as deep neural networks and gradient boosting, benefit from hyperparameter optimization. As such, in all realistic settings the question is usually not whether to use hyperparameter optimization, but rather which hyperparameter optimization technique to use.

17.3.3 Identifying algorithms (workflows) with similar predictions

This section describes a study whose aim is to identify classifiers with similar (or dissimilar) performance on individual examples. Identifying classifiers with similar performance is important, as it permits to substitute one classifier (which may be slow to train) by

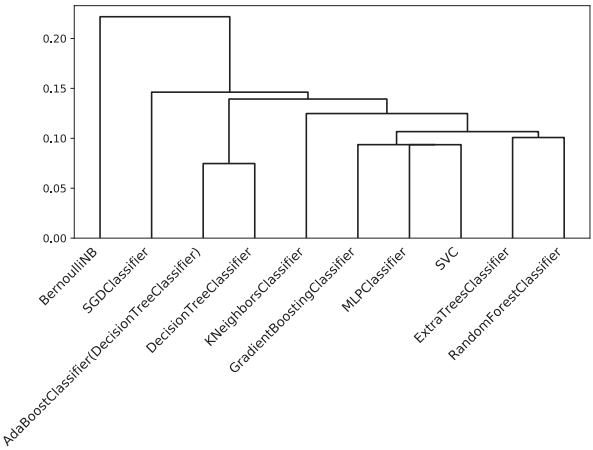


Fig. 17.7: Hierarchical clustering of Auto-sklearn classifiers

another. Identifying classifiers with different predictions is also important for another reason. As shown in Chapters 9 and 10, many ensembles (e.g., bagging ensembles) require a set of diverse classifiers. Simply assessing the performance of classifiers is not enough. Two classifiers can have similar performance but differ in some instances they are tested on.

The methodology adopted here is based on the proposal of Lee and Giraud-Carrier (2011) which used the *classifier output difference (COD)* metric, based on the difference in predictions between a pair of classifiers. The reader can consult Chapter 8 (Section 8.5) for details.

Hierarchical agglomerative clustering (HAC) converts this information into a hierarchical clustering. It starts by assigning each observation to its own cluster, and greedily joins the two clusters with the smallest distance (Rokach and Maimon, 2005). The *complete linkage* strategy is used to determine the distance between two clusters. Formally, the distance between two clusters A and B is defined as $max\{COD(a,b) : a \in A, b \in B\}$.

Figure 17.7 shows the resulting dendrogram built for all classifiers from Auto-sklearn. This figure uses test results obtained on 45 datasets from the OpenML-CC18 benchmark suite (Bischl et al., 2021). It shows which classifiers make relatively similar predictions. Apparently, the Adaboost classifier and decision tree classifier make very similar predictions on this set of datasets, whereas the predictions of the Bernoulli naive Bayes classifier are rather different from the others. This could of course also happen when a classifier performs rather badly, while all the others have quite good performance.

17.4 Effect of Specific Data/Workflow Characteristics on Performance

In this section we present three experiments. In the first one, we investigate on what type of datasets non-linear models have an advantage over linear models. This study involves both algorithm properties (whether it produces a linear model or a non-linear model) and data characteristics. In the second study we investigate the effect of feature selection on algorithm performance. As in the first case, it involves algorithm properties and data characteristics. In the final experiment, we investigate the tunability of algorithms and the effect of optimizing a given hyperparameter.

17.4.1 Effect of selecting linear vs. non-linear models

In this section we show that previous experiments can be used to investigate the relationship between certain groups (types) of algorithms and performance.

Strang et al. (2018) conducted experiments with the objective of comparing performance results of linear and non-linear models. The results were optimized using 200 iterations of random search. The linear models included linear SVM, linear neural networks (NNs) (no hidden layer), and decision stumps; the non-linear ones included SVM with RBF kernel, NNs with hidden layers, and decision trees. Strang et al. (2018) aimed to investigate for which type of datasets the former are better than the latter.

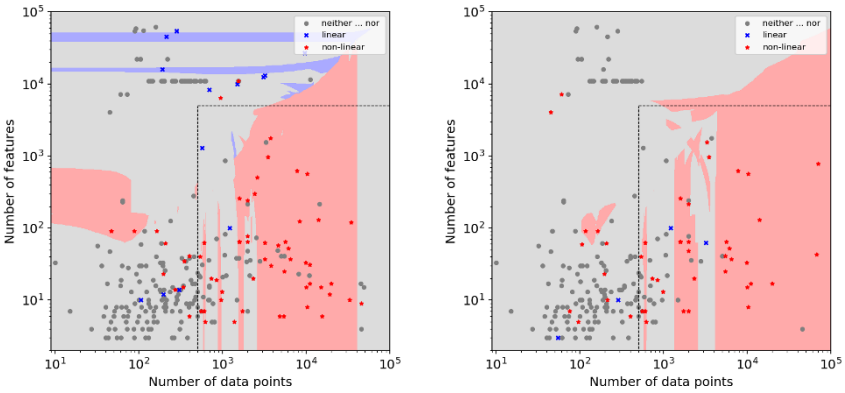
Intuitively, non-linear models have the capacity to outperform linear models. However, linear models are still used in practice, since they are simpler, computationally more efficient, and easier to interpret than their non-linear counterparts.

Figure 17.8 shows the results of the experiment involving SVM and NNs. Each dot in this figure represents the experiment of a linear version and a non-linear version of a model on different datasets. Its position is determined by two dataset characteristics, namely *number of observations* (x-axis) and *number of features* (y-axis). The color of each dot shows which type of model was better on the respective dataset. Red (blue) color indicates that the non-linear model was significantly better (worse) than the linear one. Grey color is used for cases when the difference was not statistically significant according to the Nemenyi test.

The background color shows which type of classifier is dominant in the respective region, based on a k-nearest-neighbor model with $k = 5$.[2] Figure 17.8 suggests that the non-linear models are dominant in the red regions characterized by a large number of data points (instances). Linear models are seldom significantly better than their non-linear counterparts. There are many datasets on which the performance of the two types of models is comparable, according to a statistical test. These appear in the grey region that occupies a portion of the area.

Despite the low number of datasets on which a linear model is superior, the results show that linear models may still be useful in many situations. When the performance is comparable, one argument in their favor is that they are typically faster to train and simpler to analyze. An open question is whether the results would still hold if non-linear models with advanced regularization schemes were used.

[2]The shape of the background coloring is affected by the fact that it is determined in Euclidean space and represented in log space.



(a) SVM (b) Neural network

Fig. 17.8: Linear vs. non-linear models, plotted against two data characteristics. The image was taken from Strang et al. (2018)

17.4.2 Effect of employing feature selection

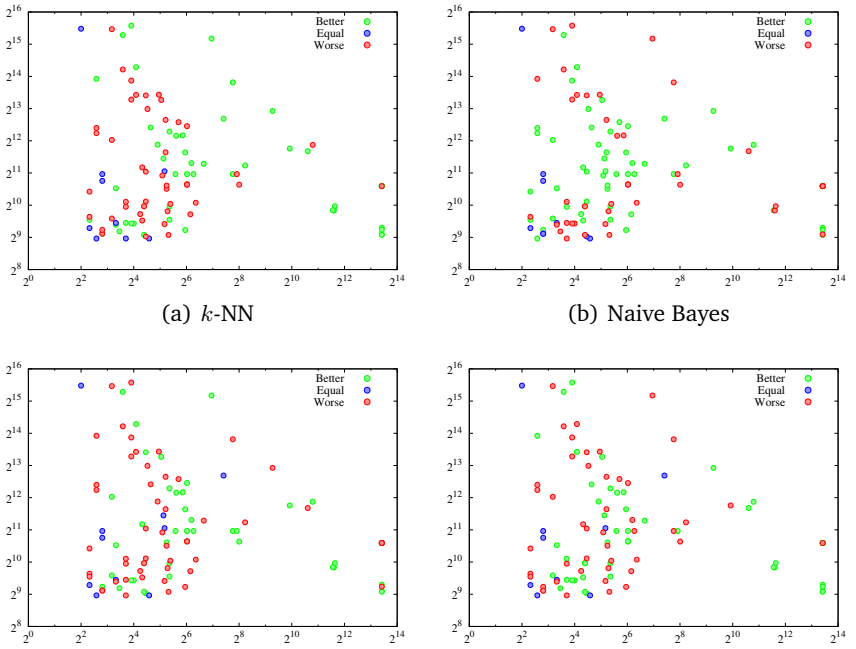
Data preprocessing is often considered an important factor that can affect the performance of classification algorithms. The experiments in OpenML can help us investigate specific questions related to preprocessing.

In this section we consider the following questions: *Does feature selection improve classification performance? If so, how do the classifier type and dataset properties influence this?*.

This section is based upon the experiments presented by Post et al. (2016). For each dataset, the authors ran a classifier without feature selection and then repeated this with feature selection. There are many different feature selection methods. In this experiment, *correlation-based feature subset selection* was used. This method tries to identify features that are highly correlated with the target attribute and uncorrelated with each other (Hall, 1999).

The results are shown in the form of a scatter plot, similar to the one in the previous section. In Figure 17.9 each dot represents a dataset used in the experiment. Its position is determined by the number of features (attributes) of that dataset (x-axis) and the corresponding number of instances (y-axis). The color of the dot shows whether feature selection yielded better or worse result. Green (red) color is used when the performance with feature selection is better (worse) than without feature selection. Blue color is used when the performance is not significantly different.

These plots show some expected behavior, as well as some interesting patterns. First of all, they reveal that feature selection is most beneficial for methods such as k-NN and naive Bayes. This is what one might expect: due to the curse of dimensionality, the nearest-neighbor methods can suffer from too many attributes (Radovanović et al., 2010) and naive Bayes is vulnerable to correlated features (John and Langley, 1995).



(a) k-NN (b) Naive Bayes

(c) Decision tree (d) Multilayer perceptron

Fig. 17.9: The effect of feature selection on classifier performance and its dependence on the number of features (x-axis) and number of instances (y-axis). This image was taken from van Rijn (2016)

Also, this study confirms that, when using k-NN, feature selection yields good results on datasets with many features (Post et al., 2016).

We also note some unexpected behavior. For example, it has been noted that some tree induction algorithms have built-in protection against irrelevant features (Quinlan, 1986). However, Figure 17.9 shows that, in many cases, feature selection is still beneficial. Also, the multilayer perceptron model is considered to be capable of selecting relevant features. As the figure shows, the situation is not so clear-cut.

Altogether, it is hard to draw clear conclusions regarding the correlation between the aforementioned dataset characteristics (metafeatures) and performance. It is conceivable that more complex models capable of predicting when to use feature selection or not can be developed (Post et al., 2016).

17.4.3 Effect of specific hyperparameter settings

As has been shown in various other chapters of this book, metalearning and AutoML systems explore a set of pre-specified alternatives in the process of elaborating a specific solution for the target task. As was shown in Chapter 8, which discussed *configuration*

spaces, the alternatives typically include different algorithms, the associated hyperparameters, and, for each one, the possible values or ranges.

In this section we focus first on some specific algorithms and investigate whether their performance can be improved by tuning their hyperparameters. In our second study, we consider different hyperparameters of some specific algorithms and investigate which of these hyperparameters have the most significant effect on performance. The answers to these questions facilitate the process of providing good recommendations, as it is possible to focus on those options that actually matter and disregard the ones that are irrelevant.

Tunability across algorithms

Probst et al. (2019) define the notion of *tunability* across algorithms, which is defined for each algorithm as the difference between the default hyperparameters and the best found hyperparameters for a given dataset. Figure 17.10(a) shows these results. This study confirms the common belief that tuning the hyperparameters of SVM is indeed of importance.

Tunability across hyperparameters

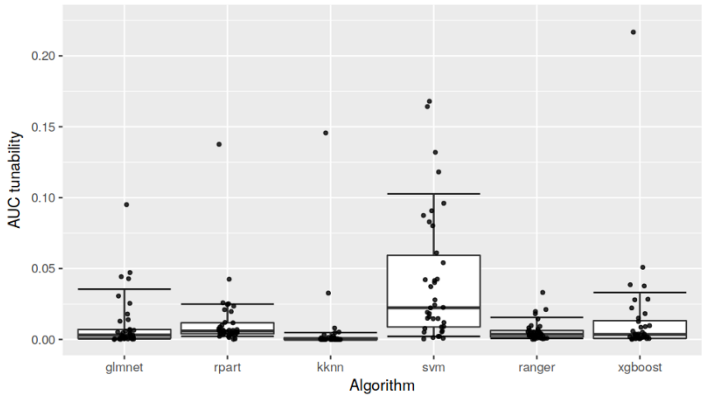
Probst et al. (2019) define also tunability across different hyperparameters of a particular algorithm. This is done by comparing the performance of the default hyperparameters against the performance of the algorithm with a *single hyperparameter* optimized. Figure 17.10(b) shows these results. Based on this figure, one could conjecture that the hyperparameters "mtry" and "samples.fraction" have the greatest influence on performance.

Hyperparameter importance across datasets based on functional ANOVA

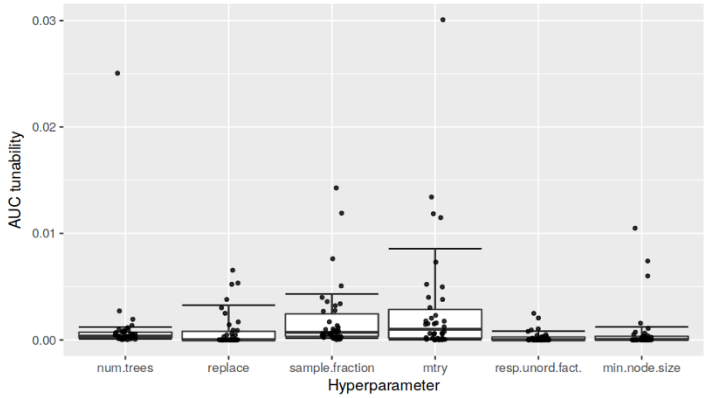
Functional ANOVA was discussed in Chapter 8 (Section 8.4). It decomposes the variance of the general model into additive components relative to a given set of hyperparameters (Hutter et al., 2014; van Rijn and Hutter, 2018). For each algorithm, a *marginal* can be calculated for each hyperparameter (or a combination of hyperparameters).[3] The importance of a given hyperparameter (or combination of hyperparameters) is related to this marginal. The higher the value, the more important the hyperparameter. Figure 17.11 shows the results for the Scikit-learn implementations of random forests, Adaboost, and support vector machine (RBF kernel).[4] The results of Figure 17.11(a) agree, to a certain level, with the results from Figure 17.10(b). In these comparisons we have to take into account that some hyperparameter names do not always agree across toolboxes. For instance, the mlR hyperparameter "mtry" is the same as the Scikit-learn hyperparameter "max features". Both methods discussed in this section identified this hyperparameter as important. There is also a disagreement in relation to some hyperparameters. For example, Scikit-learn identifies "min. samples leaf" as the most important hyperparameter, whereas mlR does not define it. van Rijn and Hutter (2018) speculate that this is because "min. samples leaf" is not as important as it seems, since it has a strong default

[3] The concept of *marginal* was also exploited in Section 17.2.2.

[4] Sharma et al. (2019) apply the same methodology on residual neural networks for image classification.



(a) Tunability across algorithms

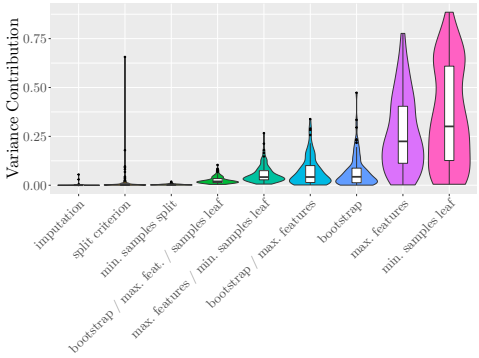


(b) Tunability across random forest hyperparameters

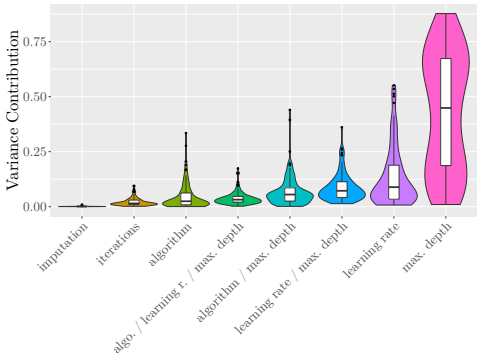
Fig. 17.10: Tunability results. The images were taken from Probst et al. (2019)

value that performs well across datasets. Furthermore, Probst et al. (2019) identify the hyperparameter "samples.fraction" as important, whereas the most closely related hyperparameter in Scikit-learn, "bootstrap", seems rather unimportant. Further investigation is required to completely understand this.

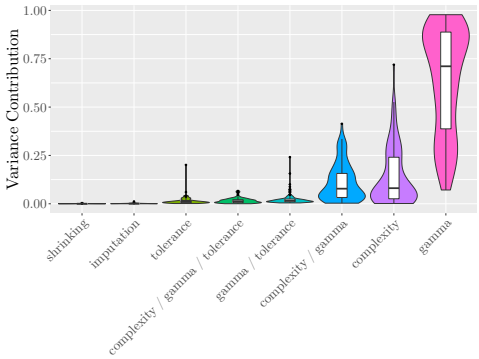
The experiments described in this section increase our understanding of which hyperparameters to focus on in the process of tuning. One important question is how we can use this knowledge to restructure the given configuration space and how it could be incorporated into the new generation of more advanced metalearning/AutoML systems.



(a) Random forest



(b) Adaboost



(c) Support vector machine (RBF kernel)

Fig. 17.11: Variance contribution across datasets. Images taken from van Rijn and Hutter (2018)

17.5 Summary

In this chapter, we have presented several experiments that were based upon the results in OpenML. These experiments were organized in blocks of increasing complexity. Section 17.2 describes studies conducted with simple experimental setups. In one, we have examined how the performance of algorithms varies on a single dataset. In another, we have examined the effect of a hyperparameter on a small set of datasets, leading to a better understanding of the algorithms.

As we know, experiments conducted on a single dataset typically do not generalize well, so Section 17.3 extends this by considering various datasets. One study was oriented towards comparisons of performance of different algorithms across different datasets. Another study showed the effect of hyperparameter optimization, allowing the reader to assess its potential effects. The final study was concerned with determining which classifiers make similar predictions and using this information to construct a hierarchical clustering, following Lee and Giraud-Carrier (2011).

The aim in Section 17.4 was to relate the performance to specific data and algorithm characteristics. We have described two studies. In the first one the results showed when it is advantageous to use (or not) feature selection, and similarly, when it is advantageous to use (non-)linear models. The last study aimed to increase our understanding about which hyperparameters are important, which can guide us towards building better configuration spaces and metalearning/AutoML systems.

References

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

Bischl, B., Casalicchio, G., Feurer, M., Gijsbers, P., Hutter, F., Lang, M., Mantovani, R. G., van Rijn, J. N., and Vanschoren, J. (2021). OpenML benchmarking suites. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, NIPS'21.

Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

Domhan, T., Springenberg, J. T., and Hutter, F. (2015). Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

Falkner, S., Klein, A., and Hutter, F. (2018). BOHB: Robust and efficient hyperparameter optimization at scale. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *ICML'18*, pages 1437–1446. JMLR.org.

Freund, Y. and Schapire, R. (1996). Experiments with a new boosting algorithm. In *Proceedings of the 13th International Conference on Machine Learning*, ICML'96, pages 148–156.

Frey, P. W. and Slate, D. J. (1991). Letter recognition using Holland-style adaptive classifiers. *Machine Learning*, 6:161–182.

Friedman, J., Hastie, T., and Tibshirani, R. (1998). Additive logistic regression: a statistical view of boosting. *Annals of Statistics*, 28:2000.

Hall, M. (1999). *Correlation-based feature selection for machine learning*. PhD thesis, University of Waikato.

Hutter, F., Hoos, H., and Leyton-Brown, K. (2014). An efficient approach for assessing hyperparameter importance. In *Proceedings of the 31st International Conference on Machine Learning*, ICML'14, pages 754–762.

Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration. *LION*, 5:507–523.

John, G. H. and Langley, P. (1995). Estimating continuous distributions in Bayesian classifiers. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 338–345. Morgan Kaufmann.

Klein, A., Falkner, S., Bartels, S., Hennig, P., and Hutter, F. (2017). Fast Bayesian optimization of machine learning hyperparameters on large datasets. In *Proc. of AISTATS 2017*.

Landwehr, N., Hall, M., and Frank, E. (2005). Logistic model trees. *Machine Learning*, 59(1-2):161–205.

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

Lee, J. W. and Giraud-Carrier, C. (2011). A metric for unsupervised metalearning. *Intelligent Data Analysis*, 15(6):827–841.

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

Post, M. J., van der Putten, P., and van Rijn, J. N. (2016). Does feature selection improve classification? a large scale experiment in OpenML. In *Advances in Intelligent Data Analysis XV*, pages 158–170. Springer.

Probst, P., Boulesteix, A.-L., and Bischl, B. (2019). Tunability: Importance of hyperparameters of machine learning algorithms. *Journal of Machine Learning Research*, 20(53):1–32.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1:81–106.

Radovanović, M., Nanopoulos, A., and Ivanović, M. (2010). Hubs in space: Popular nearest neighbors in high-dimensional data. *JMLR*, 11:2487–2531.

Rokach, L. and Maimon, O. (2005). Clustering methods. In *Data Mining and Knowledge Discovery Handbook*, pages 321–352. Springer.

Sharma, A., van Rijn, J. N., Hutter, F., and Müller, A. (2019). Hyperparameter importance for image classification by residual neural networks. In Kralj Novak, P., Šmuc, T., and Džeroski, S., editors, *Discovery Science*, pages 112–126. Springer International Publishing.

Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems 25*, NIPS'12, page 2951–2959.

Strang, B., van der Putten, P., van Rijn, J. N., and Hutter, F. (2018). Don't rule out simple models prematurely: A large scale benchmark comparing linear and non-linear classifiers in OpenML. In *International Symposium on Intelligent Data Analysis*, pages 303–315. Springer.

Thomas, J., Coors, S., and Bischl, B. (2018). Automatic gradient boosting. *arXiv preprint arXiv:1807.03873*.

van Rijn, J. N. (2016). *Massively collaborative machine learning*. PhD thesis, Leiden University.

van Rijn, J. N. and Hutter, F. (2018). Hyperparameter importance across datasets. In *KDD '18: The 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM.

Vanschoren, J., Blockeel, H., Pfahringer, B., and Holmes, G. (2012). Experiment databases: a new way to share, organize and learn from experiments. *Machine Learning*, 87(2):127–158.