Combining Feature and Algorithm Hyperparameter Selection using some Metalearning Methods

Miguel Viana Cachada[1], Salisu Mamman Abdulrahman[2], and Pavel Brazdil[1]

[1] LIAAD - INESC TEC/Faculdade de Economia, Universidade do Porto
mcachada@gmail.com,pbrazdil@inesctec.pt
[2] LIAAD - INESC TEC/Faculdade de Ciências da Universidade do Porto and Kano University of Science and Technology Wudil, Kano State, Nigeria
salisu.abdul@gmail.com

Abstract. Machine learning users need methods that can help them identify algorithms or even *workflows* (combination of algorithms with preprocessing tasks, using or not hyperparameter configurations that are different from the defaults), that achieve the potentially best performance. Our study was oriented towards *average ranking (AR)*, an algorithm selection method that exploits meta-data obtained on prior datasets. We focused on extending the use of a variant of AR* that takes A3R as the relevant metric (combining accuracy and run time). The extension is made at the level of diversity of the portfolio of workflows that is made available to AR. Our aim was to establish whether feature selection and different hyperparameter configurations improve the process of identifying a good solution. To evaluate our proposal we have carried out extensive experiments in a leave-one-out mode. The results show that AR* was able to select workflows that are likely to lead to good results, especially when the portfolio is diverse. We additionally performed a comparison of AR* with Auto-WEKA, running with different time budgets. Our proposed method shows some advantage over Auto-WEKA, particularly when the time budgets are small.

Keywords: Average Ranking, Selection of Classification Algorithms, Combining Feature and Algorithm Selection, Hyperparameters Configuration.

1 Introduction

Users of machine learning systems are facing the problem of how to choose a combination of data processing tools and algorithms. The goal is usually defined as maximizing or minimizing some quantitative measure. In classification problems, the goal could be optimizing the classification accuracy, the lift score or the ROC area (AUC). A typical practical data mining task consists of many sub-tasks which result in an extremely large search space that could be very time consuming for humans to explore manually. These sub-tasks correspond, mostly, to the workflow phases highlighted in Fig. 1: preprocessing (e.g., feature selection), algorithm selection and parametrization. Therefore strategies and methods are needed that can help the users to suggest or select an optimized data mining solution.

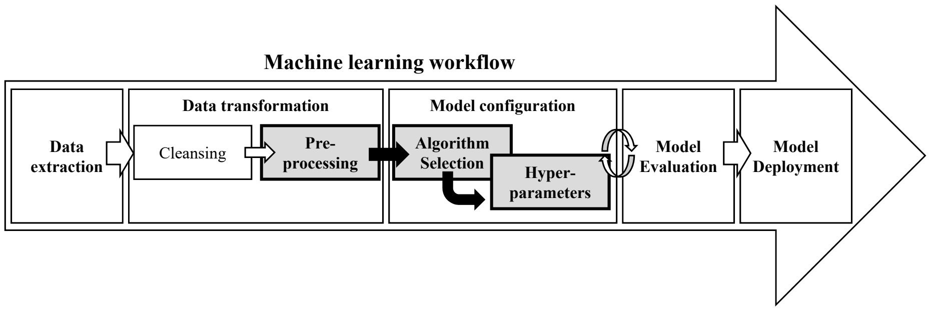

Fig. 1. A typical machine learning workflow.

Our work aimed to use *average ranking*, a very simple algorithm selection method [2], and study how the performance of this method is affected by a more diversified portfolio of workflows. These, in addition to using algorithm default configurations (as in [2]), include workflows where the application of classification algorithms is preceded by a feature selection method and/or alternative hyperparameter configurations are used. It is expected that average ranking would perform better with a larger portfolio. On the other hand, it can be argued that this does not come without a cost, in terms of resources and time.

However, with the emergence of on-line sources such as *OpenML* [35], the results from applying different workflows to a wide range of datasets are becoming available. Our work can be used as a proof of concept that average ranking serves as a good baseline recommendation method with which alternative methods could be compared. With that objective in mind we performed a small-scale comparison between average ranking and Auto-WEKA [34] and present those results here.

The remainder of this paper is organized as follows: in Section 2 we present an overview of existing work in related areas. Section 3 describes the average ranking method with a focus on the variants of ranking methods that incorporate both accuracy and run time in the evaluation strategy. Section 4 provides the experimental results and an empirical comparison with Auto-WEKA [34]. Section 5 presents conclusions and discusses future work.

2 Related Work

In this paper we address a particular case of the algorithm selection problem, oriented towards the selection of classification algorithms, that has been thoroughly investigated over the last 25 years. One approach to algorithm selection/recommendation relies on metalearning. The simplest method uses just performance results on different datasets in the form of rankings. Some commonly used measures of performance are accuracy, AUC or A3R that combines accuracy and run time [1]. The rankings are then aggregated to obtain a single aggregated ranking. The aggregated ranking can be used as a simple model to identify the top algorithms to be used. This strategy is sometimes referred to as the *Top-N* strategy [5].

A more advanced approach, often considered as the *classical metalearning approach*, uses, in addition to performance results, a set of measures that characterize datasets [28, 5, 32]. Other approaches exploit estimates of performance based on past tests in so-called active testing method for algorithm selection [22].

The main idea of feature subset selection (FS) is to remove redundant or irrelevant features from the dataset as they can lead to a reduction of the classification accuracy or clustering quality and to an unnecessary increase of computational cost. Feature selection and dimensionality reduction approaches are discussed extensively in literature [27, 14]. Many practical studies were performed to evaluate these techniques on different fields, for example: e-mail filtering and drug discovery [18], e-mail spam detection [12], bioinformatics [31] and healthcare [8, 7]. In some of these studies it was observed that dimensionality reduction techniques improved the classifiers accuracy while others conclude the opposite. It can be observed that the usefulness of such techniques is likely to be dependent on the type of data and problem. A question arises whether the inclusion of such workflows in the given portfolio improves the overall quality of search for the best solution.

The choice of algorithm hyperparameters has been typically formalized as an optimization problem, being the objective function the same metric used to evaluate the performance of the corresponding algorithm. A simple method is *grid search*, which consists in exhaustively searching within a predefined set of hyperparameter values. Another method, *random search*, was introduced by Bergstra and Bengio [3] to overcome the potential large computational cost of grid search. Some classes of algorithms, like neural networks, allow for the use of gradient descent for hyperparameter optimization [25]. Classical heuristics for optimized search have also been suggested, for example, genetic algorithms [29], tabu search [13] and particle swarm optimisation [26].

An approach that has recently been attracting attention for its good results is *Bayesian optimization*. It consists in iteratively fitting a probabilistic model as each hyperparameter combination is tested. The aim is that this model gives good suggestions on what combinations should be tried next. Some methods to construct this function, *optimizers*, have been suggested, namely, SMAC - Sequential Model-based Algorithm Configuration [17], Spearmint [33] and TPE - Tree Parzen Estimator [3]. An optimizer benchmarking framework and comparison of the three methods is presented in [10].

Auto-WEKA [20, 34] is a tool designed to help novice users of ML by automatically searching through the joint space of WEKA learning algorithms and their respective hyperparameter settings to maximize a given performance measure (for instance accuracy, AUC, etc.) by using a SMAC [17] optimizer.

Other algorithm selection tools include *auto-sklearn* [11] that, apart from using a SMAC optimizer, uses classical metalearning techniques, namely, dataset similarity through metafeatures and automatic ensemble construction, *ASlib* [4], a benchmark library for algorithm selection containing 17 algorithm selection scenarios from six different areas with a focus on (but not limited to) constraint satisfaction problems, *AutoFolio* [24] that uses SMAC to automatically determine a well-performing algorithm selection approach and its hyperparameters for a given algorithm selection data and *Leveraging Learning to Automatically Manage Algorithms* (LLAMA) [19], an R package for algorithm portfolios and selection.

3 Overview of the AR Method

This section presents a brief review of the average ranking method that is often used in comparative studies in the machine learning literature. This method can be regarded as a variant of Borda's method [23]. For each dataset the algorithms are ordered according to the performance measure chosen (e.g., predictive accuracy) and assigned ranks. Among popular ranking criteria we find, for instance, *success rates*, *AUC*, and *significant wins* [6, 9, 21]. The best algorithm is assigned rank 1, the runner-up is assigned rank 2, and so on. Let $r^j_{A_k}$ be the rank of algorithm k on dataset j. In this work we use *average ranks*, which is obtained using

$$r\hat{_{A_k}} = \left(\sum_{j=1}^{D} r^j_{A_k} \right) \div n \tag{1}$$

where n is the number of datasets. The final ranking is obtained by ordering the average ranks and assigning ranks to the individual algorithms accordingly.

Average ranking represents a useful method for deciding which algorithm should be used. It would normally be followed on a new, target dataset: first the algorithm with rank 1 is evaluated, then the one with rank 2 and so on. In each step the better one is maintained as the potentially best option. In this context, average ranking can be referred to as the *recommended ranking*.

3.1 Average Ranking AR* that gives preference to fast tests

Ranking methods use a particular performance measure to construct an ordering of algorithms. Some commonly used measures of performance are accuracy, AUC or A3R that combines accuracy and run time [1]. Although other measures could have been used instead, here we focus on the *ranking* that exploits a combined measure of accuracy and run time, *A3R*. As the authors of [1] have shown, this leads to good results when loss time curves [30] are used in the evaluation. Here we use the following formulation for this measure:

$$A3R^{d_i}_{a_{ref},a_j} = \frac{\frac{SR^{d_i}_{a_j}}{SR^{d_i}_{a_{ref}}}}{(T^{d_i}_{a_j}/T^{d_i}_{a_{ref}})^P} \tag{2}$$

where $SR^{d_i}_{a_j}$ and $SR^{d_i}_{a_{ref}}$ represent the *success rates* (accuracies) of algorithms a_j and a_{ref} on dataset d_i, where a_{ref} represents a given *reference algorithm.* Instead of accuracy, AUC or another measure can be used as well. Similarly, $T^{d_i}_{a_j}$ and $T^{d_i}_{a_{ref}}$ represent the run times of the algorithms, in seconds. As the average ranking method does not require pairwise comparisons, the values of $SR^{d_i}_{a_{ref}}$ and $T^{d_i}_{a_{ref}}$ can be set to 1. Hence, we can use the simplified formula $A3R^{'}_{a_j} = SR^{d_i}_{a_j}/(T^{d_i}_{a_j})^P$, as in [30].

To trade off the importance of time, the denominator is raised to the power of P, while P is usually some small number, such as 1/64, representing in effect, the 64^{th} root. This is motivated by the observation that run times vary much more than accuracies. It is not uncommon that one particular algorithm is several orders of magnitude slower (or faster) than another. Obviously, we do not want the time ratios to completely dominate the equation. If we take $P = 1/N$ (i.e. N^{th} root), we get a number that goes to 1 when P is approaching 0. In our experiments described in Section 4, we follow [30, 2] and choose $P = 1/64$, as it was shown to lead to better results than some other settings. This version of average ranking is referred to as *AR**.

3.2 Evaluation using loss time curves

Our aim was to investigate the impact of using feature selection and different hyperparameter configurations on average ranking method. The results are presented in the form of loss time curves [30] which show how the *performance loss* depends on time. The *loss* is calculated as the difference between the performance of the algorithm identified using the proposed ranking method and the ideal choice (which is the best performance known, identified within the largest set of performance results available in our study). The individual loss time curves are aggregated into a mean loss time curve[1]. The mean loss time curve can be characterized by a number representing the *mean loss* in a given interval (*MIL*). This characteristic is similar to AUC, but there is an important difference. When talking about AUCs, the values fall in the 0-1 interval. Our loss time curves span the interval T_{min} - T_{max} which is specified by the user. Typically the user only worries about run times when they exceed a minimum value. In the experiments here we have set T_{min} to 10 seconds. The value of T_{max} was set to 10^4 seconds, i.e. about 2.78 hours.

[1] Another possibility would be to use median loss time curves. These could be accompanied by 25% and 75% percentile bands.

4 The Methodology Adopted and Experimental Results

4.1 Overview of the methodology adopted

Our first aim is to study the effect of using feature selection with given classification algorithms when using the A3R-based average raking method. We accomplished this by constructing portfolios of algorithms that are preceded or not by feature selection. We then performed a similar analysis with portfolios of workflows that also include different hyperparameter configurations for some of the algorithms.

The workflows are evaluated over each of the 37 datasets we use (refer to Table 4 in the Appendix). In each study, loss time curves and MIL values are generated using leave-one-out (LOO) cross-validation, where 36 datasets are used to generate the model (i.e., average ranking) and the corresponding loss time curve on the dataset left out.

The average ranking is followed sequentially. The AR* method uses the concept of *current best* (a_{best}). When this method comes to testing algorithm a_i in the ranking, the test is carried out using 10-fold cross validation (CV). If the performance of algorithm a_i is better than that of a_{best}, a_i is used as the new a_{best}. This way we obtain one loss time curve per dataset in every LOO cycle. The individual loss time curves are used to generate the mean loss time curve. Using LOO cross-validation helps to gain confidence that the average ranking can be effectively transferred to new datasets and produce satisfactory outcomes.

Note that the loss is computed in reference to the best workflow overall for the dataset left out, which is identified within the largest portfolio of algorithms available. In our study, this portfolio includes all variants that take different hyperparameter configurations into consideration, with and without feature selection.

All experiments were performed using Weka software [15] and its algorithms implementations.

4.2 Effect of feature selection

To study the effect of feature selection, we use a total of 124 workflows. The setting for the experiment uses two different portfolios. The first one contains 62 classification algorithms with their default hyperparameter settings, while the second includes workflows consisting of Correlation Feature Selection (CFS) [16] followed by one of the 62 classification algorithms.

Fig. 2 shows the results of 3 different variants of the A3R-based average ranking method. The variant *AR** uses just our set of classification algorithms with no feature selection method and default hyperparameter configurations. *AR*+FS+A* is the variant where our algorithms are always preceded by CFS. The variant *AR*±FS+A* uses all 124 algorithm configurations.

Overall, the variant *AR*±FS+A* yields the smallest MIL value (Table 1). However, the MIL of the variant *AR** is very close and, in addition, both loss time curves are quite similar (Fig. 2).

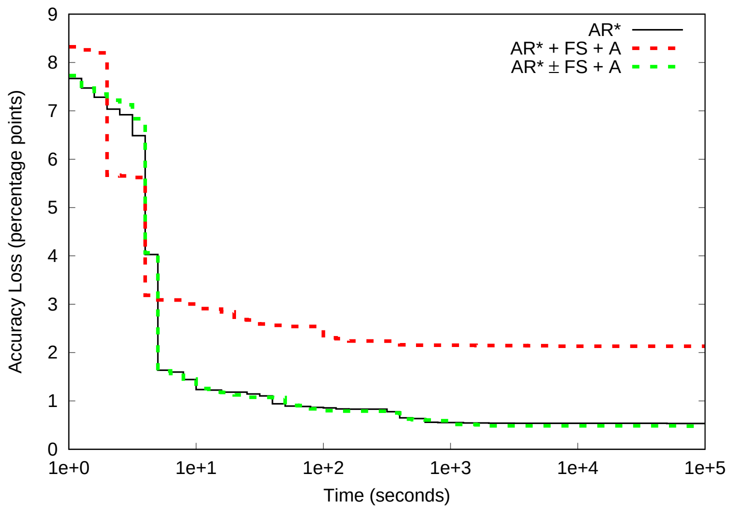

Fig. 2. Loss time curves of the variants of A3R-based average ranking method used to study the effect of feature selection

Table 1. Mean interval loss associated with the variants of A3R-based average ranking method used to study the effect of feature selection

AR-Variant	AR*	AR*+FS+A	AR*±FS+A
MIL	0.7735	2.3396	**0.7476**

Regarding *AR*+FS+A*, although the algorithms preceded by feature selection are, in general, faster than their counterparts, as they deal with datasets with fewer features, their accuracy tends to be negatively affected. In Fig. 2 it can be observed that the *AR*+FS+A* loss time curve is able to compete with the other two variants up to approximately the 6 second mark, but eventually reaches its limit in accuracy improvement. This happens because the set of algorithms used does not include the ones with performance closer to the best available. In our study, *AR*+FS+A* provides the best accuracy for just 4 datasets, while *AR*±FS+A* achieves the best accuracy in 13 datasets.

4.3 Effect of diversity of hyperparameter configurations

We have additionally experimented with several versions of some of the algorithms in our algorithm portfolio. Each version was associated with a particular configuration of the corresponding hyperparameters. The extended portfolio included 3 versions of Multilayer Perceptron, 7 of Support Vector Machines (with polynomial and radial basis function kernels), 7 of Random Forests, 8 of J48 and

5 of K-Nearest Neighbors, totalling 30 additions (refer to Tables 5 and 6 in the Appendix for more information regarding the algorithms used).

To study the effect of different algorithm parametrizations we again define *AR** as the baseline portfolio and compare it with *AR*+Hyp+A* that includes the 30 extra algorithm versions mentioned above and the variant *AR*±FS+Hyp+A*, which contemplates also feature selection.

AR±FS+Hyp+A* is the portfolio with the largest number of workflows, 184, corresponding to 62 algorithms with default configurations, 30 versions with alternative hyperparameter configurations and the same 92 previous configurations preceded by feature selection.

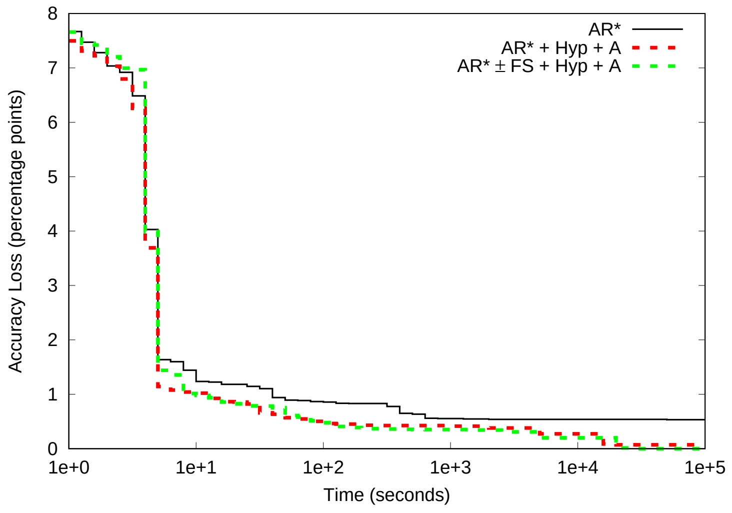

Fig. 3. Loss time curves of the variants used to study the effect of hyperparameter configurations diversity

Table 2. Mean interval loss associated with the variants of A3R-based average ranking method used to study the effect of hyperparameter configurations diversity

AR-Variant	AR*	AR*+Hyp+A	AR*±FS+Hyp+A
MIL	0.7735	0.5035	**0.4691**

From Table 2 it can be observed that, similarly as in the previous study (Section 4.2), the variant with more diversity, *AR*±FS+Hyp+A* in this case, is the one with the lowest MIL. However, when compared with *AR*+Hyp+A*, we note that the difference in terms of MIL is not very high and also, when observing both loss time curves (Fig. 3), we cannot say that one is better than the other. In fact, up to the 100 seconds mark *AR*±FS+Hyp+A* appears to be worse than *AR*+Hyp+A*. In our view, this is because, in many cases, average ranking includes different variants of the same algorithm in a close succession. This may lead to a loss of time spent on testing variants that do not result in any gain. Nevertheless, the variant *AR*±FS+Hyp+A* is the most complete set of algorithm alternatives and includes all best possible accuracies achievable per dataset, hence its loss time curve eventually reaches 0.

When compared with *AR**, the other two portfolios allow to clearly achieve better loss time curves and lower MIL, confirming that a portfolio that is richer in algorithm versions has increased the chances of providing better recommendations. Our results also suggest, however, that it is desirable to provide methods that allow to construct portfolios that include all important variants, but exclude others, in order to avoid losing time testing many "similar" alternatives.

4.4 Comparison between A3R-based AR and Auto-WEKA

For the purpose of evaluating the effectiveness of our proposed method in practice, we have decided to carry out similar experiments using Auto-WEKA [20, 34]. An empirical study was done by comparing the accuracy obtained by each system for the same run time. The Auto-WEKA runs were made using cross-validation with 10 folds: *reference accuracy* results from the average of all folds; *reference run time* is the sum of training and testing time, measured using all data to train and test the model.

Regarding the Auto-WEKA experiment, a required parameter is *time limit*, which sets a time budget for it to run. The actual run time is, however, somewhat different from the one defined in the parameter, so it needs to be measured. After running, Auto-WEKA outputs one or more recommendations for the model configurations (that always include an algorithm and its hyperparameters and may or may not include a feature selection procedure). For the purpose of this empirical study, we used only the first Auto-WEKA recommendation for each dataset. The train and test time spent needs to be added to the given budget. In this study we have used four time budgets: 5, 15, 30 and 60 minutes. The steps to compare both methods are detailed below. They refer to one dataset but the same steps were repeated for all datasets.

1. Auto-WEKA is run for a predefined time budget. Its actual run time (*search time*) is measured and the recommended configuration is recorded.
2. The recommended configuration is run within a *WEKA Experiment class*, using cross-validation with 10 folds. The predictive accuracy for the recommended algorithm returned by Auto-WEKA is obtained.
3. The Auto-WEKA *model run time* is measured from the tasks of training and testing the recommended configuration over all data.
4. Auto-WEKA total run time is computed by adding the search time to the recommended model run time. The sum of the two times is used to retrieve the actual performance of our system.

One way to compare the overall performances of the algorithm recommendation methods is to count the number of data sets on which a particular method is the overall winner. Table 3 shows the aggregated results for the four Auto-WEKA time budgets, where *Win* refers to the number of cases where *AR*±FS+Hyp+A* has higher accuracy than Auto-WEKA. This is a simple comparison that does not consider the statistical significance test of the differences. Still, it can be observed that *AR*±FS+Hyp+A* has more wins for lower Auto-WEKA time budgets.

Table 3. Comparing *AR*±FS+Hyp+A* with Auto-WEKA

Budget Time(min)	Win	Loss	Ties
5	35	1	1
15	31	5	1
30	29	7	1
60	25	11	1

5 Conclusion and Future Work

In this paper we have presented a study that exploits the average ranking method AR* that gives preference to well-performing workflows which are also fast to test. Our portfolios combine algorithm selection with feature selection and a set of hyperparameter configurations. The portfolio that uses the most diverse number of workflow configurations (*AR*±FS+Hyp+A*) achieved the best results. The second runner-up was *AR*+Hyp+A* with quite similar performance. These results confirm that it is indeed important to incorporate hyperparameter configurations into the portfolio, as it can lead to improved performance. Although the addition of extra variants to the given portfolio could, in principle, slow down the process of identifying the best or the near best solution, the usage of AR* mitigates this adverse effect. It opts for fast and good performing workflows before the others.

Additionally, we have compared A3R-based average ranking against Auto-WEKA and showed that the proposed method competes quite well with the latter, especially when smaller time budgets are used.

Future work

Our future plan is to devise a method of pruning portfolios of workflows with the aim to avoid investing time in testing the variants of algorithms that are of similar nature and have similar performance. Another alternative line that could be followed could explore the approaches based on active testing (AT) and/or surrogate models, whose aim is to select the most promising candidate to test.

It would be desirable to use a much larger portfolio to guarantee that the potentially best solutions are not really left out. We could collect more test results with alternative feature selection methods and preprocessing methods, as well as hyperparameter configurations, or even reuse a larger set of results already available through OpenML [35].

We could also extend the comparison with Auto-WEKA to higher time budgets, as well as use other currently available methods to perform further comparisons (e.g., auto-sklearn [11]).

We are also considering the investigation of approaches that allow to take into account the costs (run time) of off-line tests used to gather the meta-data that our proposed method exploits. Their cost could be set to some fraction of the cost of on-line test (i.e. tests on a new dataset), but not really ignored altogether.

Acknowledgements The authors wish to express our gratitude to the following institutions which have provided funding to support this work:

– Federal Government of Nigeria Tertiary Education Trust Fund under the TETFund 2012 AST$D Intervention for Kano University of Science and Technology, Wudil, Kano State, Nigeria for PhD Overseas Training;
– Project *NanoSTIMA: Macro-to-Nano Human Sensing: Towards Integrated Multimodal Health Monitoring and Analytics/NORTE-01-0145-FEDER-000016*, which is financed by the North Portugal Regional Operational Programme (NORTE 2020), under the PORTUGAL 2020 Partnership Agreement, and through the European Regional Development Fund (ERDF).

References

1. Abdulrahman, S.M., Brazdil, P.: Measures for Combining Accuracy and Time for Meta-learning. In: Meta-Learning and Algorithm Selection Workshop at ECAI 2014. pp. 49–50 (2014)
2. Abdulrahman, S.M., Brazdil, P., van Rijn, J.N., Vanschoren, J.: Speeding up algorithm selection using average ranking and active testing by introducing runtime. Machine learning Special Issue on Metalearning and Algorithm Selection (2017)
3. Bergstra, J.S., Bardenet, R., Bengio, Y., Kégl, B.: Algorithms for hyper-parameter optimization. In: Advances in Neural Information Processing Systems. pp. 2546–2554 (2011)
4. Bischl, B., Kerschke, P., Kotthoff, L., Lindauer, M., Malitsky, Y., Fréchette, A., Hoos, H., Hutter, F., Leyton-Brown, K., Tierney, K., et al.: Aslib: A benchmark library for algorithm selection. Artificial Intelligence 237, 41–58 (2016)
5. Brazdil, P., Giraud-Carrier, C., Soares, C., Vilalta, R.: Metalearning: Applications to data mining. Springer Science & Business Media (2008)
6. Brazdil, P., Soares, C., Da Costa, J.P.: Ranking learning algorithms: Using ibl and meta-learning on accuracy and time results. Machine Learning 50(3), 251–277 (2003)
7. Chu, C., Hsu, A.L., Chou, K.H., Bandettini, P., Lin, C., Initiative, A.D.N.: Does feature selection improve classification accuracy? Impact of sample size and feature selection on classification using anatomical magnetic resonance images. Neuroimage 60(1), 59–70 (2012)

8. Cuingnet, R., Chupin, M., Benali, H., Colliot, O.: Spatial and anatomical regularization of SVM for brain image analysis. . In Advances in Neural Information Processing Systems pp. 460–468 (2010)
9. Demšar, J.: Statistical Comparisons of Classifiers over Multiple Data Sets. The Journal of Machine Learning Research 7, 1–30 (2006)
10. Eggensperger, K., Feurer, M., Hutter, F., Bergstra, J., Snoek, J., Hoos, H., Leyton-Brown, K.: Towards an empirical foundation for assessing bayesian optimization of hyperparameters. In: NIPS workshop on Bayesian Optimization in Theory and Practice. pp. 1–5 (2013)
11. Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., Hutter, F.: Efficient and robust automated machine learning. In: Advances in Neural Information Processing Systems. pp. 2962–2970 (2015)
12. Gansterer, W.N., Janecek, A.G., Neumayer, R.: Spam filtering based on latent semantic indexing. In: Survey of Text Mining II, pp. 165–183. Springer (2008)
13. Gomes, T.A., Prudncio, R.B., Soares, C., Rossi, A.L., Carvalho, A.: Combining meta-learning and search techniques to select parameters for support vector machines. Neurocomputing 75(1), 3–13 (2012)
14. Guyon, I., Elisseeff, A.: An introduction to variable and feature selection. In: Journal of machine learning research. pp. 1157–1182. JMLR. (2003)
15. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA Data Mining Software: An Update. ACM SIGKDD explorations newsletter 11(1), 10–18 (2009)
16. Hall, M.A.: Correlation-based feature selection for machine learning. Ph.D. thesis, The University of Waikato (1999)
17. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. International Conference on Learning and Intelligent Optimization pp. 507–523 (2011)
18. Kohavi, R., John, G.H.: Wrappers for feature subset selection. Artificial intelligence. 97(1), 273–324 (1997)
19. Kotthoff, L.: Llama: leveraging learning to automatically manage algorithms. arXiv preprint arXiv:1306.1031 (2013)
20. Kotthoff, L., Thornton, C., Hoos, H.H., Hutter, F., Leyton-Brown, K.: Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka. Journal of Machine Learning Research 17, 1–5 (2016)
21. Leite, R., Brazdil, P.: Active Testing Strategy to Predict the Best Classification Algorithm via Sampling and Metalearning. In: ECAI. pp. 309–314 (2010)
22. Leite, R., Brazdil, P., Vanschoren, J.: Selecting Classification Algorithms with Active Testing. In: Machine Learning and Data Mining in Pattern Recognition, pp. 117–131. Springer (2012)
23. Lin, S.: Rank aggregation methods. WIREs Computational Statistics 2, 555–570 (2010)
24. Lindauer, M., Hoos, H.H., Hutter, F., Schaub, T.: Autofolio: An automatically configured algorithm selector. Journal of Artificial Intelligence Research 53, 745–778 (2015)
25. Maclaurin, D., Duvenaud, D., Adams, R.P.: Gradient-based hyperparameter optimization through reversible learning. In: Proceedings of the 32nd International Conference on Machine Learning (2015)
26. de Miranda, P.B., Prudêncio, R.B., de Carvalho, A.C.P., Soares, C.: Combining a multi-objective optimization approach with meta-learning for svm parameter selection. In: In Systems, Man, and Cybernetics (SMC). pp. 2909–2914. IEEE. (2012)

27. Molina, L.C., Belanche, L., Nebot, A.: Feature selection algorithms: a survey and experimental evaluation. In: Proceedings. 2002 IEEE International Conference on. pp. 306–313. IEEE (2003)
28. Pfahringer, B., Bensusan, H., Giraud-Carrier, C.: Tell me who can learn you and I can tell you who you are: Landmarking various learning algorithms. In: Proceedings of the 17th International Conference on Machine Learning. pp. 743–750 (2000)
29. Reif, M., Shafait, F., Dengel, A.: Meta-learning for evolutionary parameter optimization of classifiers. Machine learning 87(3), 357–380 (2012)
30. van Rijn, J.N., Abdulrahman, S.M., Brazdil, P., Vanschoren, J.: Fast Algorithm Selection using Learning Curves. In: Advances in Intelligent Data Analysis XIV. Springer (2015)
31. Saeys, Y., Inza, I., Larraaga, P.: A review of feature selection techniques in bioinformatics. bioinformatics 23(19), 2507–2517 (2007)
32. Smith-Miles, K.A.: Cross-disciplinary Perspectives on Meta-Learning for Algorithm Selection. ACM Computing Surveys (CSUR) 41(1), 6:1–6:25 (2008)
33. Snoek, J., Larochelle, H., Adams, R.P.: Practical bayesian optimization of machine learning algorithms. In: Advances in neural information processing systems. pp. 2951–2959 (2012)
34. Thornton, C., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In: In Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 847–855. ACM (2013)
35. Vanschoren, J., van Rijn, J.N., Bischl, B., Torgo, L.: Openml: networked science in machine learning. ACM SIGKDD Explorations Newsletter 15(2), 49–60 (2013)

Appendix

Table 4. Datasets used in the study.

Dataset	OpenML Task_id	#Instances	#Attributes	#Classes
anneal	2	898	39	6
kr-vs-kp	3	3 196	37	2
letter	6	20 000	17	26
balance-scale	11	625	5	3
mfeat-factors	12	2 000	217	10
mfeat-fourier	14	2 000	77	10
breast-w	15	699	10	2
mfeat-karhunen	16	2 000	65	10
mfeat-morphological	18	2 000	7	10
mfeat-pixel	20	2 000	241	10
car	21	1 728	7	4
mfeat-zernike	22	2 000	48	10
cmc	23	1 473	10	3
mushroom	24	8 124	23	2
nursery	26	12 960	9	5
optdigits	28	5 620	65	10
credit-a	29	690	16	2
page-blocks	30	5 473	11	5
credit-g	31	1 000	21	2
pendigits	32	10 992	17	10
cylinder-bands	33	540	40	2
segment	36	2 310	20	7
diabetes	37	768	9	2
soybean	41	683	36	19
spambase	43	4 601	58	2
splice	45	3 190	62	3
tic-tac-toe	49	958	10	2
vehicle	53	846	19	4
waveform-5000	58	5 000	41	3
electricity	219	45 312	9	2
solar-flare	2068	1 066	13	3
yeast	2073	1 484	9	10
satimage	2074	6 430	37	6
abalone	2075	4 177	9	29
kropt	2076	28 056	7	18
baseball	2077	1 340	18	3
eucalyptus	2079	736	20	5

Table 5. Algorithms used in the study.

#	Algorithm	#	Algorithm
1	A1DE	32	Kstar
2	AdaBoostM1_DecisionStump	33	LADTree
3	AdaBoostM1_IBk	34	LMT
4	AdaBoostM1_J48	35	LogitBoost_DecisionStump
5	AdaBoostM1_LMT	36	LWL_DecisionStump
6	AdaBoostM1_NaiveBayes	37	LWL_J48
7	AdaBoostM1_OneR	38	LWL_RandomTree
8	AdaBoostM1_RandomTree	39	MultiBoostAB_DecisionStump
9	AdaBoostM1_REPTree	40	MultiBoostAB_IBk
10	Bagging_DecisionStump	41	MultiBoostAB_J48
11	Bagging_IBk	42	MultiBoostAB_JRip
12	Bagging_J48	43	MultiBoostAB_NaiveBayes
13	Bagging_Jrip	44	MultiBoostAB_OneR
14	Bagging_LMT	45	MultiBoostAB_RandomTree
15	Bagging_LWL_DecisionStump	46	MultiBoostAB_REPTree
16	Bagging_NaiveBayes	47	MultilayerPerceptron
17	Bagging_OneR	48	NaiveBayes
18	Bagging_RandomTree	49	NaiveBayesUpdateable
19	Bagging_REPTree	50	NBTree
20	BayesNet	51	OneR
21	ClassificationViaRegression_M5P	52	PART
22	CVParameterSelection_ZeroR	53	RacedIncrementalLogitBoost_D.Stump
23	Dagging_DecisionStump	54	RandomCommittee_RandomTree
24	DecisionStump	55	RandomForest
25	DTNB	56	RandomSubSpace_REPTree
26	END_ND_J48	57	RandomTree
27	HoeffdingTree	58	REPTree
28	IBK	59	SimpleLogistic
29	IterativeClassifierOptimizer_LogitBoost_D.Stump	60	SMO_PolyKernel
30	J48	61	SMO_RBFKernel
31	JRip	62	ZeroR

Table 6. Alternative hyperparameter configurations. The hyperparameters that are not mentioned in the table were used with their default values, with the only exception of the number of training epochs (N) in MultilayerPerceptron, that was set to 100 for all experiments instead of the default 500.

Algorithm	Parameter interval*	Parameter description**
RandomForest	P = [80, **100**]	Size of each bag, as a percentage of the training set size.
	K = [**0**, 250]	Number of attributes to randomly investigate, where 0 = int(log2(#attributes)+1).
	M = [**1**, 10]	Minimum number of instances per leaf .
J48	C = [0.01, **0.25**, 0.5]	Confidence threshold for pruning (smaller values incur more pruning).
	M = [**2**, 10, 20]	Minimum number of instances per leaf .
SMO_PolyKernel	C = [0.1, **1**, 10]	Complexity parameter.
	E = [**1**, 2]	The exponent of the polynomial kernel.
SMO_RBFKernel	C = [0.1, **1**, 10]	Complexity parameter.
IBK	K = [**1**, 5, 20]	The number of neighbors to use.
	I = [**No**, Yes]	Whether to perform distance weighting by 1/distance.
MultilayerPerceptron	H = [**a**, o]	Number of hidden layers, where 'a' = (#attributes + #classes)/2 and 'o' = #classes.
	D = [**No**, Yes]	Whether the learning rate decays as training progresses.

*Parameters in bold are the default.
**Adapted from WEKA documentation.