Mach Learn (2018) 107:79–108
https://doi.org/10.1007/s10994-017-5687-8

Speeding up algorithm selection using average ranking and active testing by introducing runtime

Salisu Mamman Abdulrahman[1,2] · **Pavel Brazdil**[2,3] · **Jan N. van Rijn**[4,5] · **Joaquin Vanschoren**[6]

Received: 10 May 2016 / Accepted: 4 October 2017 / Published online: 14 November 2017

Abstract Algorithm selection methods can be speeded-up substantially by incorporating multi-objective measures that give preference to algorithms that are both promising and fast to evaluate. In this paper, we introduce such a measure, A3R, and incorporate it into two algorithm selection techniques: *average ranking* and *active testing*. Average ranking combines algorithm rankings observed on prior datasets to identify the best algorithms for a new dataset. The aim of the second method is to iteratively select algorithms to be tested on the new dataset, learning from each new evaluation to intelligently select the next best candidate. We show how both methods can be upgraded to incorporate a multi-objective measure A3R that combines accuracy and runtime. It is necessary to establish the correct balance between accuracy and runtime, as otherwise time will be wasted by conducting less informative tests. The correct balance can be set by an appropriate parameter setting within function A3R that trades off accuracy and runtime. Our results demonstrate that the upgraded

Editor: Christophe Giraud-Carrier.

✉ Salisu Mamman Abdulrahman
salisu.abdul@gmail.com

Pavel Brazdil
pbrazdil@inesctec.pt

Jan N. van Rijn
vanrijn@informatik.uni-freiburg.de

Joaquin Vanschoren
j.vanschoren@tue.nl

1 Kano University of Science and Technology, Wudil, Kano State, Nigeria

2 LIAAD, INESC TEC, Rua Dr. Roberto Frias, 4200-465 Porto, Portugal

3 Faculdade de Economia, Universidade do Porto, Rua Dr. Roberto Frias, 4200-465 Porto, Portugal

4 University of Freiburg, Freiburg, Germany

5 Leiden Institute of Advanced Computer Science, Leiden University, Leiden, The Netherlands

6 Eindhoven University of Technology, Eindhoven, Netherlands

versions of Average Ranking and Active Testing lead to much better mean interval loss values than their accuracy-based counterparts.

Keywords Algorithm selection · Meta-learning · Ranking of algorithms · Average ranking · Active testing · Loss curves · Mean interval loss

1 Introduction

A large number of data mining algorithms exist, rooted in the fields of machine learning, statistics, pattern recognition, artificial intelligence, and database systems, which are used to perform different data analysis tasks on large volumes of data. The task to recommend the most suitable algorithms has thus become rather challenging. Moreover, the problem is exacerbated by the fact that it is necessary to consider different combinations of parameter settings, or the constituents of composite methods such as ensembles.

The algorithm selection problem, originally described by Rice (1976), has attracted a great deal of attention, as it endeavours to select and apply the best algorithm(s) for a given task (Brazdil et al. 2008; Smith-Miles 2008). The algorithm selection problem can be cast as a *learning* problem: the aim is to learn a model that captures the relationship between the properties of the datasets, or meta-data, and the algorithms, in particular their performance. This model can then be used to predict the most suitable algorithm for a given new dataset.

This paper presents two new methods, which build on ranking approaches for algorithm selection (Brazdil and Soares 2000; Brazdil et al. 2003) in that it exploits meta-level information acquired in past experiments.

The first method is known as *average ranking (AR)*, which calculates an average ranking for all algorithms over all prior datasets. The upgrade here consists of using A3R, a multi-objective measure that combines accuracy and runtime (the time needed to evaluate a model). Many earlier approaches used only accuracy.

The second method uses an algorithm selection strategy known as *active testing (AT)* (Leite and Brazdil 2010; Leite et al. 2012). The aim of active testing is to iteratively select and evaluate a candidate algorithm whose performance will most likely exceed the performance of previously tested algorithms. Here again, function A3R is used in the estimates of the performance gain, instead of accuracy, as used in previous versions.

It is necessary to establish the correct balance between accuracy and runtime, as otherwise time will be wasted by conducting less informative and slow tests. In this work, the correct balance can be set by a parameter setting within the A3R function. We have identified a suitable value using empirical evaluation.

The experimental results are presented in the form of loss-time curves, where time is represented on a log scale. This representation is very useful for the evaluation of rankings representing *schedules*, as was shown earlier (Brazdil et al. 2003; van Rijn et al. 2015). The results presented in this paper show that the upgraded versions of AR and AT lead to much better mean *interval loss values (MIL)* than their solely accuracy-based counterparts.

Our contributions are as follows. We introduce A3R, a measure that can be incorporated in multiple meta-learning methods to boost the performance in loss-time space. We show how this can be done with the AR and AT methods and establish experimentally that performance indeed increases drastically. As A3R requires one parameter to be set, we also experimentally explore the optimal value of this parameter.

The remainder of this paper is organized as follows. In Sect. 2 we present an overview of existing work in related areas.

Section 3 describes the average ranking method with a focus on how it was upgraded to incorporate both accuracy and runtime. As the method includes a parameter, this section describes also how we searched for the best setting. Finally, this section presents an empirical evaluation of the new method.

Section 4 provides details about the active testing method. We explain how this method relates to the earlier proposals and how it was upgraded to incorporate both accuracy and runtime. This section includes also experimental results and a comparison of both upgraded methods and their accuracy-based counterparts.

Section 5 is concerned with the issue of how robust the average ranking method is to omissions in the meta-dataset. This issue is relevant because meta-datasets gathered by researchers are very often incomplete. The final section presents conclusions and future work.

2 Related work

In this paper we are addressing a particular case of the algorithm selection problem (Rice 1976), oriented towards the selection of classification algorithms. Various researchers addressed this problem in the course of the last 25 years.

2.1 Meta-learning approaches to algorithm selection

One very common approach, that could now be considered as *the classical approach*, uses a set of measures to characterize datasets and establish their relationship to algorithm performance. This information is often referred to as *meta-data* and the dataset containing this information as *meta-dataset*.

The meta-data typically includes a set of simple measures, statistical measures, information-theoretic measures and/or the performance of simple algorithms referred to as *landmarkers* (Pfahringer et al. 2000; Brazdil et al. 2008; Smith-Miles 2008). The aim is to obtain a model that characterizes the relationship between the given meta-data and the performance of algorithms evaluated on these datasets. This model can then be used to predict the most suitable algorithm for a given new dataset, or alternatively, provide a ranking of algorithms, ordered by their suitability for the task at hand. Many studies conclude that ranking is in fact better, as it enables the user to iterative test the top candidates to identify the algorithms most suitable in practice. This strategy is sometimes referred to as the *Top-N* strategy (Brazdil et al. 2008).

2.2 Active testing

The *Top-N* strategy has the disadvantage that it is unable to exploit the information acquired in previous tests. For instance, if the top algorithm performs worse than expected, this may tell us something about the given dataset which can be exploited to update the ranking. Indeed, very similar algorithms are now also likely to perform worse than expected. This led researchers to investigate an alternative testing strategy, known as *active testing* (Leite et al. 2012). This strategy intelligently selects the most useful tests using the concept of *estimates of performance gain*.[1] These estimates the *relative probability* that a particular algorithm

[1] We prefer to use this term here, instead of the term *relative landmarkers* which was used in previous work (Fürnkranz and Petrak 2001) in a slightly different way.

will outperform the current best candidate. In this paper we attribute particular importance to the tests on the new dataset. Our aim is to propose a way that minimizes the time before the best (or near best) algorithm is identified.

2.3 Active learning

Active Learning is briefly discussed here to eliminate a possible confusion with active testing. The two concepts are quite different. Some authors have also used *active learning* for algorithm selection (Long et al. 2010), and exploited the notion of *Expected Loss Optimization* (ELO). Another notable active learning approach to meta-learning was presented by Prudêncio and Ludermir (2007), where the authors used active learning to support the selection on informative meta-examples (i.e. datasets). Active learning is somewhat related to experiment design (Fedorov 1972).

2.4 Combining accuracy and runtime

Different proposals were made in the past regarding how to combine accuracy and runtime. One early proposal involved function *ARR (adjusted ratio of ratios)* (Brazdil et al. 2003), which has the form:

$$ARR^{d_i}_{a_{ref},a_j} = \frac{\frac{SR^{d_i}_{a_j}}{SR^{d_i}_{a_{ref}}}}{1 + AccD * log(T^{d_i}_{a_j} / T^{d_i}_{a_{ref}})} \quad (1)$$

Here, $SR^{d_i}_{a_j}$ 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.

AccD is a parameter that needs to be set and represents the amount of accuracy he/she is willing to trade for a 10 times speed-up or slowdown. For example, AccD = 10% means that the user is willing to trade 10% of accuracy for 10 times speed-up/slowdown.

The ARR function should ideally be monotonically increasing. Higher success rate ratios should lead to higher values of ARR. Higher time ratios should lead to lower values of ARR. The overall effect of combining the two should again be monotonic. In one earlier work (Abdulrahman and Brazdil 2014) the authors have decided to verify whether this property can be verified on data. This study is briefly reproduced in the following paragraphs.

The value of SRR was fixed to 1 and the authors varied the time ratio from very small values 2^{-20} to very high ones 2^{20} and calculated the ARR for three different values of $AccD$ (0.2, 0.3 and 0.7). The result can be seen in Fig. 1. The horizontal axis shows the log of the time ratio ($logRT$). The vertical axis shows the ARR value.

As can be seen, the resulting ARR function is not monotonic and even approaching infinity at some point. Obviously, this can lead to incorrect rankings provided by the meta-learner.

This problem could be avoided by imposing certain constraints on the values of the ratio, but here we prefer to adopt a different function, A3R, that also combines accuracy and runtime and exhibits a monotonic behaviour. It is described in Sect. 3.3.

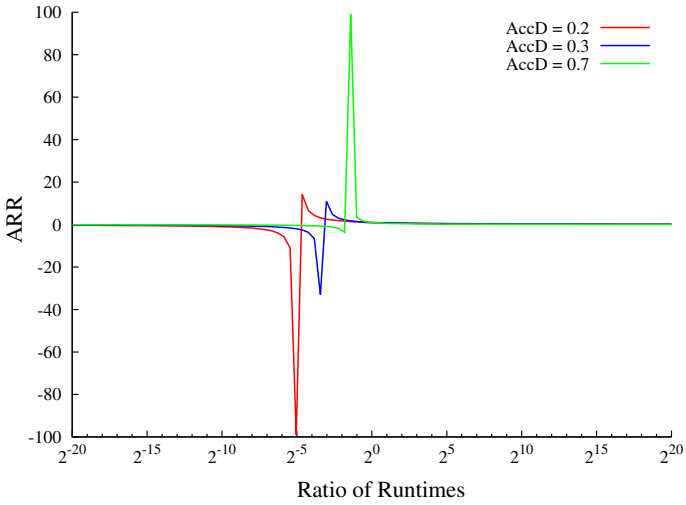

Fig. 1 ARR with three different values for $AccD$ (0.2,0.3 and 0.7)

2.5 Hyperparameter optimization

This area is clearly relevant to algorithm selection, since most learning algorithms have parameters that can be adjusted and whose values may affect the performance of the learner. The aim is to identify a set of hyperparameters for a learning algorithm, usually with the goal of obtaining good generalization and consequently low loss Xu et al. (2011). The choice of algorithm can also be seen as a hyperparameter, in which case one can optimize the choice of algorithm and hyperparameters at the same time. However, these methods can be computationally very expensive and typically start from scratch for every new dataset (Feurer et al. 2015). In this work, we aim to maximally learn from evaluations on prior datasets to find the (near) best algorithms in a shorter amount of time. Our method can also be extended to recommend both algorithms and parameter settings (Leite et al. 2012), which we aim to explore in more depth in future work.

2.6 Aggregation of rankings

The method of aggregation depends on whether we are dealing with complete or incomplete rankings. *Complete rankings* are those in which N items are ranked M times and no value in this set is missing. Aggregation of such rankings is briefly reviewed in Sect. 3.1.

Incomplete rankings arise when only some ranks are known in some of the M rankings. Many diverse methods exist. According to Lin (2010), these can be divided into three categories: Heuristic algorithms, Markov chain methods and stochastic optimization methods. The last category includes, for instance, *Cross Entropy Monte Carlo, (CEMC)* methods. Merging incomplete rankings typically involve rankings of different ranks and some approaches require that these rankings are completed before aggregation. Let us consider a simple example. Suppose ranking R_1 represents 4 elements, namely (a_1, a_3, a_4, a_2), while R_2 represents a of just two elements (a_2, a_1). Some approaches would require that the missing elements in R_2 (i.e. a_3, a_4) be attributed a concrete rank (e.g. rank 3). This does not seem to be cor-

rect: we should not be forced to assume that some information exists when in fact we have none.[2]

In Sect. 5.1 we address the problem of robustness against incomplete rankings. This arises when we have incomplete test results in the meta-dataset. We have investigated how much the performance of our methods degrades under such circumstances. Here we have developed a simple heuristic method based on Borda's method reviewed in Lin (2010). In the studies conducted by this author, simple methods often compete quite well with other more complex approaches.

2.7 Multi-armed bandits

The multi-armed bandit problem involves a gambler whose aim is to decide which arm of a K-slot machine to pull to maximize his total reward in a series of trials. Many real-world learning and optimization problems can be modeled in this way and algorithm selection is one of them. Different algorithms can be compared to different arms. Gathering knowledge about different arms can be compared to the process of gathering meta-data which involves conducting tests with the given set of algorithms on given datasets. This phase is often referred to as *exploration*.

Many meta-learning approaches assume that tests have been done off-line without any cost, prior to determining which is the best algorithm for the new dataset. This phase exploits the meta-knowledge acquired and hence can be regarded as *exploration*. However, the distinction between the two phases is sometimes not so easy to define. For instance, in the active testing approach discussed in Sect. 4, tests are conducted both off-line and online, while the new dataset is being used. Previous tests condition which tests are done next.

Several strategies or algorithms have been proposed as a solution to the multi-armed bandit problem in the last two decades. Some researchers have introduced so called *contextual-bandit problem*, where different arms are characterized by features. For example, some authors (Li et al. 2010) have applied this approach to personalized recommendation of news articles. In this approach a learning algorithm sequentially selects articles to serve users based on contextual information about the users and articles, while simultaneously adapting its article-selection strategy based on user-click feedback. Contextual approaches can be compared to meta-learning approaches that exploit dataset features.

Many articles on multi-armed bandits are based on the notion of *reward* which is received after an arm has been pulled. The difference to the optimal is often referred to as *regret* or *loss*. Typically, the aim is to maximize the accumulated reward, which is equivalent to minimizing the accumulated loss, as different arms are pulled. Although initial studies were done on this issue (e.g. Jankowski (2013)), this area has, so far, been rather under-explored. To the best of our knowledge there is no work that would provide an algorithmic solution to the problem of which arm to pull when pulling different arms can take different amounts of time. So one novelty of this paper is that it takes the time of tests into account with an adequate solution.

3 Upgrading the average ranking method by incorporating runtime

The aim of this paper is to determine whether the following hypotheses can be accepted:

[2] We have considered using a package of R *RankAggreg* (Pihur et al. 2009), but unfortunately we would have to attribute a concrete rank (e.g. k + 1) to all missing elements.

Hyp1: The incorporation of a function that combines accuracy and runtime is useful for the construction of the average ranking, as it leads to better results than just accuracy when carrying out evaluation on loss-time curves.

Hyp2: The incorporation of a function that combines accuracy and runtime for the active testing method leads to better results than only using accuracy when carrying out evaluation on loss-time curves.

The rest of this section is dedicated to the average ranking method. First, we present a brief overview of the method and show how the average ranking can be constructed on the basis of prior test results. This is followed by the description of the function A3R that combines accuracy and runtime, and how the average ranking method can be upgraded with this function. Furthermore, we empirically evaluate this method by comparing the ranking obtained with the ranking representing the golden standard. Here we also introduce loss-time curves, a novel representation that is useful in comparisons of rankings.

As our A3R function includes a parameter that determines the weight attributed to either accuracy or time, we have studied the effects of varying this parameter on the overall performance. As a result of this study, we identify the range of values that led to the best results.

3.1 Overview of the average ranking 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 (Lin 2010).

For each dataset, the algorithms are ordered according to the performance measure chosen (e.g., predictive accuracy) and ranks are assigned accordingly. Among many popular ranking criteria we find, for instance, *success rates*, *AUC*, and *significant wins* (Brazdil et al. 2003; Demšar 2006; Leite and Brazdil 2010). The best algorithm is assigned rank 1, the runner-up is assigned rank 2, and so on. Should two or more algorithms achieve the same performance, the attribution of ranks is done in two steps. In the first step, the algorithms that are tied are attributed successive ranks (e.g. ranks 3 and 4). Then all tied algorithms are assigned the mean rank of the occupied positions (i.e. 3.5).

Let r_i^j be the rank of algorithm i on dataset j. In this work we use *average ranks*, inspired by Friedman's *M statistic* (Neave and Worthington 1988). The average rank for each algorithm is obtained using

$$r_i = \left(\sum_{j=1}^{D} r_i^j \right) \div D \tag{2}$$

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

The average ranking represents a quite useful method for deciding which algorithm should be used. Also, it can be used as a baseline against which other methods can be compared.

The average ranking would normally be followed on the new dataset: first the algorithm with rank 1 is evaluated, then the one with rank 2 and so on. In this context, the average ranking can be referred to as the *recommended ranking*.

3.1.1 Evaluation of rankings

The quality of a ranking is typically established through comparison with the *golden standard*, that is, the ideal ranking on the new (test) dataset(s). This is often done using a leave-one-out

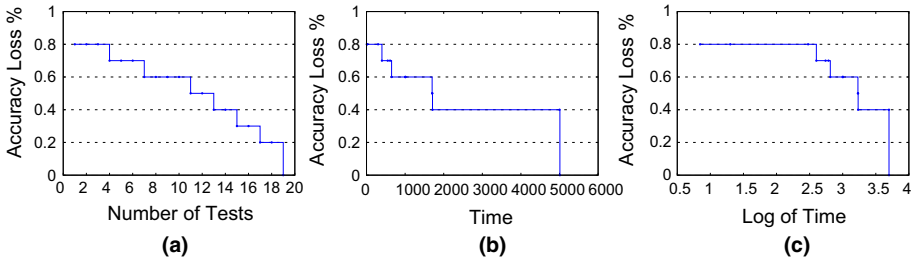

Fig. 2 Loss curves for accuracy-based average ranking. **a** Loss-curve. **b** Loss-time Curve. **c** Loss-time curve (log)

cross-validation (CV) strategy (or in general k-fold CV) on all datasets: in each leave-one-out cycle the recommended ranking is compared against the ideal ranking on the left-out dataset, and then the results are averaged for all cycles.

Different evaluation measures can be used to evaluate how close the recommended ranking is to the ideal one. Often, this is a type of correlation coefficient. Here we have opted for *Spearman's rank correlation* (Neave and Worthington 1988), but Kendall's Tau correlation could have been used as well. Obviously, we want to obtain rankings that are highly correlated with the ideal ranking.

A disadvantage of this approach is that it does not show directly what the user is gaining or losing when following the ranking. As such, many researchers have adopted a second approach which simulates the sequential evaluation of algorithms on the new dataset (using cross-validation) as we go down the ranking. The measure that is used is the *performance loss*, defined as the difference in accuracy between a_{best} and $a*$, where a_{best} represents the best algorithm identified by the system at a particular time and $a*$ the truly best algorithm that is known to us (Leite et al. 2012).

As tests proceed following the ranking, the loss either maintains its value, or decreases when the newly selected algorithm improved upon the previously selected algorithms, yielding a loss curve. Many typical loss curves used in the literature show how the loss depends on the number of tests carried out. An example of such curve is shown in Fig. 2a. Evaluation is again carried out in a leave-one-out fashion. In each cycle of the leave-one-out cross-validation (LOO-CV) one loss curve is generated. In order to obtain an overall picture, the individual loss-time curves are aggregated into a *mean loss curve*. An alternative to using LOO-CV would be to use k-fold CV (with e.g. k = 10). This issue is briefly discussed in Sect. 6.1.

3.1.2 Loss-time curves

A disadvantage of loss curves it that they only show how loss depends on the number of tests. However, some algorithms are much slower learners than others—sometimes by several orders of magnitude, and these simple loss curves do not capture this.

This is why, in this article, we follow Brazdil et al. (2003) and van Rijn et al. (2015) and take into account the actual time required to evaluate each algorithm and use this information when generating the loss curve. We refer to this type of curve as a *loss versus time curve*, or *loss-time curve* for short. Figure 2b shows an example of a loss-time curve, corresponding to the loss curve in Fig. 2a.

As train/test times include both very small and very large numbers, it is natural to use the logarithm of the time (log_{10}), instead of the actual time. This has the effect that the same time

intervals appear to be shorter as we shift further on along the time axis. As normally the user would not carry out exhaustive testing, but rather focus on the first few items in the ranking, this representation makes the losses at the beginning of the curve more apparent. Figure 2c shows the arrangement of the previous loss-time curve on a log scale.

Each loss time curve can be characterized by a number representing the mean loss in a given interval, an area under the loss-time curve. The individual loss-time curves can be aggregated into a mean loss-time curve. We want this *mean interval loss* (MIL) to be as low as possible. This characteristic is similar to AUC, but there is one important difference. When talking about AUCs, the x-axis values spans between 0 and 1, while our loss-time curves span between some T_{min} and T_{max} defined by the user. Typically the user searching for a suitable algorithm would not worry about very short times where the loss could still be rather high. In the experiments here we have set T_{min} to 10 s. In an on-line setting, however, we might need a much smaller value. The value of T_{max} needs also to be set. In the experiments here it has been set to 10^4 s corresponding to about 2.78 h. We assume that most users would be willing to wait a few hours, but not days, for the answer. Also, many of our loss curves reach 0, or values very near 0 at this time. Note that this is an arbitrary setting that can be changed, but here it enables us to compare loss-time curves.

3.2 Data used in the experiments

This section describes the dataset used in the experiments described in this article. The meta-dataset was constructed from evaluation results retrieved from OpenML (Vanschoren et al. 2014), a collaborative science platform for machine learning. This dataset contains the results of 53 parameterized classification algorithms from the Weka workbench (Hall et al. 2009) on 39 classification datasets.[3] More details about the 53 classification algorithms can be found in the Appendix.

3.3 Combining accuracy and runtime

In many situations, we have a preference for algorithms that are fast and also achieve high accuracy. However, the question is whether such a preference would lead to better loss-time curves. To investigate this, we have adopted a multi-objective evaluation measure, A3R, described in Abdulrahman and Brazdil (2014), that combines both accuracy and runtime. Here we use a slightly different formulation to describe this measure:

$$A3R^{d_i}_{a_{ref},a_j} = \frac{\frac{SR^{d_i}_{a_j}}{SR^{d_i}_{a_{ref}}}}{(T^{d_i}_{a_j}/T^{d_i}_{a_{ref}})^P} \tag{3}$$

Here $SR^{d_i}_{a_j}$ 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.

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 three orders of magnitude slower (or faster) than

[3] Full details: http://www.openml.org/s/37.

Table 1 Effect of varying P on time ratio

C	$P = 1/2^C$	1000^P	C	$P = 1/2^C$	1000^P
0	1	1000.000	6	1/64	1.114
1	1/2	31.623	7	1/128	1.055
2	1/4	5.623	8	1/256	1.027
3	1/8	2.371	9	1/512	1.013
4	1/16	1.539	10	1/1024	1.006
5	1/32	1.241	∞	0	1.000

another. Obviously, we do not want the time ratios to completely dominate the equation. If we take the N^{th} root of the ratios, we will get a number that goes to 1 in the limit, when N is approaching infinity (i.e. if P is approaching 0).

For instance, if we used $P = 1/256$, an algorithm that is 1000 times slower would yield a denominator of 1.027. It would thus be equivalent to the faster reference algorithm only if its accuracy was 2.7% higher than the reference algorithm. Table 1 shows how a ratio of 1000 (one algorithm is 1000 times slower than the reference algorithm) is reduced for decreasing values of P. As P gets lower, the time is given less and less importance.

A simplified version of A3R introduced in van Rijn et al. (2015) assumes that both the success rate of the reference algorithm $SR^{d_i}_{a_{ref}}$ and the corresponding time $T^{d_i}_{a_{ref}}$ have a fixed value. Here the values are set to 1. The simplified version, $A3R'$, which can be shown to yield the same ranking, is defined as follows:

$$A3R'^{d_i}_{a_j} = \frac{SR^{d_i}_{a_j}}{(T^{d_i}_{a_j})^P} \tag{4}$$

We note that if P is set to 0, the value of the denominator will be 1. So in this case, only accuracy will be taken into account. In the experiments described further on we used $A3R$ (not $A3R'$).

3.3.1 Upgrading the average ranking method using A3R

The performance measure A3R can be used to rank a given set of algorithms on a particular dataset in a similar way as accuracy. Hence, the average rank method described earlier was upgraded to generate a time-aware average ranking, referred to as the *A3R-based average ranking*.

Obviously, we can expect somewhat different results for each particular choice of parameter P that determines the relative importance of accuracy and runtime, thus it is important to determine which value of P will lead to the best results in loss-time space. Moreover, we wish to know whether the use of A3R (with the best setting for P) achieves better results when compared to the approach that only uses accuracy. The answers to these issues are addressed in the next sections.

3.3.2 Searching for the best parameter setting

Our first aim was to generate different variants of the A3R-based average ranking resulting from different settings of P within A3R and identify the best setting. We have used a grid search and considered settings of P ranging from P = 1/4 until P = 1/256, shown in Table 2.

Table 2 Mean interval loss of AR-A3R associated with the loss-time curves for different values of P

P=	1/4	1/16	1/64	1/128	1/256	0
MIL	0.752	0.626	**0.531**	0.535	0.945	22.11

The bold value indicates the best value corresponding to the lowest MIL loss

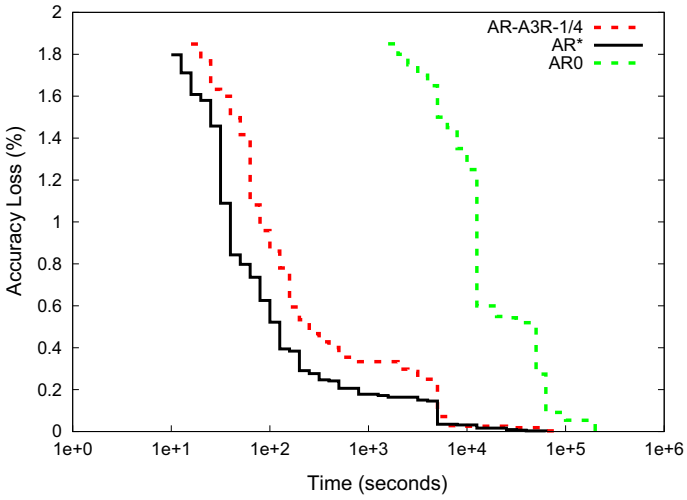

Fig. 3 Loss-time curves for A3R-based and accuracy-based average ranking

The last value shown is P = 0. If this value is used in $(T_{a_j}^{d_i} / T_{a_{ref}}^{d_i})^P$ the result would be 1. The last option corresponds to a variant when only accuracy matters.

All comparisons were made in terms of *mean interval loss (MIL)* associated with the mean loss-time curves. As we have explained earlier, different loss-time curves obtained in different cycles of leave-one-out method are aggregated into a single mean loss-time curve, shown also in Fig. 3. For each one we calculated MIL, resulting in Table 2.

The MIL values in this table represent *mean values* for different cycles of the leave-one-out mode. In each cycle the method is applied to one particular dataset.

The results show that the setting of P = 1/64 leads to better results than other values, while the setting P = 1/128 is not too far off. Both settings are better than, for instance, P = 1/4, which attributes a much higher importance to time. They are also better than, for instance, P = 1/256 which attributes much less importance to time, or to P = 0 when only accuracy matters.

The boxplots in Fig. 4 show how the MIL values vary for different datasets. The boxplots are in agreement with the values shown in Table 2. The variations are lowest for the settings P = 1/16, P = 1/64 and P = 1/128, although for each one we note various outliers. The variations are much higher for all the other settings. The worst case is P = 0 when only accuracy matters.

For simplicity, the best version identified, that is AR-A3R-1/64, is identified by the short name AR* in the rest of this article. Similarly, the version AR-A3R-0 corresponding to the case when only accuracy matters is referred to as AR0.

As AR* produces better results than AR0 we have provided evidence in favor of hypothesis Hyp1 presented earlier.

An interesting question arises why AR0 has such a bad performance. Using AR with accuracy-based ranking leads to disastrous results (MIL = 22.11) and should be avoided at all costs! This issue is addressed further on in Sect. 5.2.2.

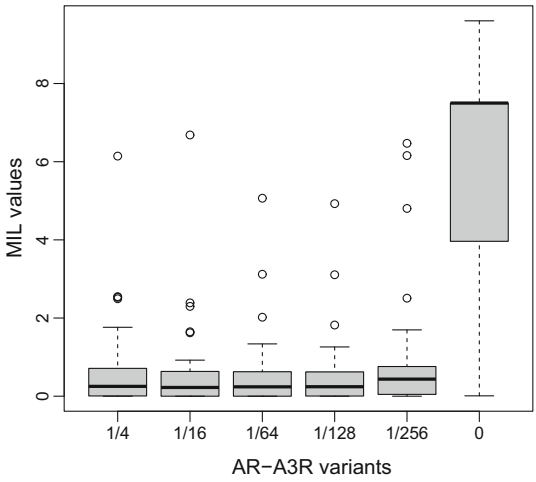

Fig. 4 Boxplots showing the distribution of MIL values for the settings of P

3.3.3 Discussion

Parameter P could be used as a user-defined parameter to determine his/her relative interest on accuracy or time. In other words, this parameter could be used to establish the trade-off between accuracy and runtime, depending on the operation condition required by the user (e.g. a particular value of T_{max}, determining the time budget).

However, one very important result of our work is that there is an optimum for which the user will obtain the best result in terms of MIL.

The values of T_{min} and T_{max} define an interval of interest in which we wish to minimize MIL. It is assumed that all times in this interval are equally important. We assume that the user could interrupt the process at any time T lying in this interval and request the name of a_{best}, the best algorithm identified.

4 Active testing using accuracy and runtime

The method of A3R-based average ranking described in the previous section has an important shortcoming: if the given set of algorithms includes many similar variants, these will be close to each other in the ranking, and hence their performance will be similar on the new dataset. In these circumstances it would be beneficial to try to use other algorithms that could hopefully yield better results. The AR method, however, passively follows the ranking, and hence is unable to skip very similar algorithms.

This problem is quite common, as similar variants can arise for many reasons. One reason is that many machine learning algorithms include various parameters which may be set to different values, yet have limited impact. Even if we used a *grid* of values and selected only some of possible alternative settings, we would end up with a large number of variants. Many of them will exhibit rather similar performance.

Clearly, it is desirable to have a more intelligent way to choose algorithms from the ranking. One very successful way to do this is *active testing* (Leite et al. 2012). This method starts with a *current best algorithm*, a_{best}, which is initialized to the topmost algorithm in the average ranking. It then selects new algorithms in an iterative fashion, searching in each step

for the *best competitor*, a_c. This best competitor is identified by calculating the *estimated performance gain* of each untested algorithm with respect to a_{best} and selecting the algorithm that maximizes this value. In Leite et al. (2012) the performance gain was estimated by finding the most similar datasets, looking up the performance of every algorithm, and comparing that to the performance of a_{best} on those datasets. If another algorithm outperforms a_{best} on many similar datasets, it is a good competitor.

In this paper we have decided to use a simpler approach without focusing on the most similar datasets, but correct one major shortcoming, which is that it does not take runtime into account. As a result, this method can spend a lot of time evaluating slow algorithms even if they are expected to be even marginally better. Hence, our aim is to upgrade the active testing method by incorporating A3R as the performance measure and analyzing the benefits of this change. Moreover, as A3R includes a parameter P, it is necessary to determine the best value for this setting.

4.1 Upgrading active testing with A3R

In this section we describe the upgraded active testing method in more detail. The main algorithm is presented in *Algorithm 1 (AT-A3R)*, which shows how the datasets are used in a leave-one-out evaluation. In step 5 the method constructs the $AR*$ average ranking, $\bar{A}$. This ranking is used to identify the topmost algorithm, which is used to initialize the value of a_{best}. Then *Algorithm 2 (AT-A3R′)* containing the actual active testing procedure is invoked. Its main aim is to construct the loss curve L_i for one particular dataset d_i and add it to the other loss curves Ls. The final step involves aggregating all loss curves and returning the *mean loss curve* L_m.

Algorithm 1 AT-A3R - Active testing with A3R

Require: algorithms A, datasets D_s, parameter P
1: $Ls \leftarrow ()$ (*Initialize the list of loss-time curves to an empty list*)
2: *Leave-one-out cycle (d_i represents d_{new}):*
3: **for all** d_i in D_s **do**
4: $D_x \leftarrow D_s - d_i$
5: Construct $AR*$ average ranking, $\bar{A}$, of algorithms A on D_x
6: $a_{best} \leftarrow \bar{A}[1]$ (the topmost element)
7: $\bar{A} \leftarrow \bar{A} - a_{best}$
8: $(L_i, a_{best}) \leftarrow$ **ATA3R'**$(d_i, D_x, a_{best}, \bar{A}, P)$ ***(Algorithm 2)***
9: *Add the new loss curve L_i to the list:*
$Ls \leftarrow Ls + L_i$
10: **end for**
11: Construct the mean loss curve L_m by aggregating all loss cuves in Ls
Return: Mean loss curve L_m

4.1.1 Active testing with A3R on one dataset

The main active testing method in *Algorithm 2* includes several steps:

Step 1: It is used to initialize certain variables.

Step 2: The performance of the current best algorithm a_{best} on d_{new} is obtained using a cross-validation (CV) test.

Steps 3–12 (overview): These steps include a *while loop*, which at each iteration identifies the best competitor (step 4), removes it from the ranking $\bar{A}$ (step 5) and obtains its performance (step 6). If its performance exceeds the performance of a_{best}, it replaces it. This process is repeated until all algorithms have been processed. More details about the individual steps are given in the following paragraphs.

Algorithm 2 ATA3R' - Active testing with A3R on one dataset

Require: $d_i, D_x, a_{best}, \bar{A}, P$
1: Initialize ranking d_{new} and loss curve L_i:
$d_{new} \leftarrow d_i,\ L_i \leftarrow ()$

2: Obtain the performance of a_{best} on dataset d_{new} using a CV test:
$(T_{a_{best}}^{d_{new}},\ SR_{a_{best}}^{d_{new}}) \leftarrow CV(a_{best}, d_{new})$

3: **while** $|\bar{A}| > 0$ **do**
4: Find the most promising competitor a_c of a_{best} using estimates of performance gain:
$a_c = \underset{a_k}{\text{argmax}} \sum_{d_i \in D_s} \Delta Pf(a_k, a_{best}, d_i)$
5: $\bar{A} \leftarrow \bar{A} - a_c$ *(Remove a_c from $\bar{A}$)*
6: Obtain the performance of a_c on dataset d_{new} using a CV test:
$(T_{a_c}^{d_{new}},\ SR_{a_c}^{d_{new}}) \leftarrow CV(a_c, d_{new})$
$L_i \leftarrow L_i + (T_{a_c}^{d_{new}},\ SR_{a_c}^{d_{new}})$

7: Compare the accuracy performance of a_c with a_{best} and carry out updates:
8: **if** $SR_{a_c}^{d_{new}} > SR_{a_{best}}^{d_{new}}$ **then**
9: $a_{best} \leftarrow a_c,\ T_{a_{best}}^{d_{new}} \leftarrow T_{a_c}^{d_{new}},\ SR_{a_{best}}^{d_{new}} \leftarrow SR_{a_c}^{d_{new}}$
10: **end if**
11: **end while**
12: **return** Loss-time curve L_i and a_{best}

Step 4: This step is used to identify the best competitor. This is done by considering all past tests and calculating the sum of estimated performance gains ΔPf for different datasets. This is repeated for all different algorithms and the one with the maximum sum of ΔPf is used as the *best competitor* a_c, as shown in Eq. 5:

$$a_c = \underset{a_k}{\text{argmax}} \sum_{d_i \in D} \Delta Pf(a_k, a_{best}, d_i) \tag{5}$$

The estimate of performance gain, ΔPf, is reformulated in terms of A3R:

$$\Delta Pf(a_j, a_{best}, d_i) = r\left(\frac{\frac{SR_{a_j}^{d_i}}{SR_{a_{best}}^{d_i}}}{(T_{a_j}^{d_i}/T_{a_{best}}^{d_i})^P} - 1 > 0\right) * \left(\frac{\frac{SR_{a_j}^{d_i}}{SR_{a_{best}}^{d_i}}}{(T_{a_j}^{d_i}/T_{a_{best}}^{d_i})^P} - 1\right) \tag{6}$$

where a_j is an algorithm and d_i a dataset. The function $r(test)$ returns 1 if the *test* is true and 0 otherwise.

An illustrative example is presented in Fig. 5, showing different values of ΔPf of one potential competitor with respect to a_{best} on all datasets.

Table 3 shows the estimates of potential performance gains for 5 potential competitors. The competitor with the highest value (a_2) is chosen, expecting that it will improve the accuracy on the new dataset.

Step 6: After the best competitor has been identified, the method proceeds with a cross-validation (CV) test on the new dataset to obtain the actual performance of the best competitor. After this the loss curve L_i is updated with the new information.

Step 7-10: A test is carried to determine whether the best competitor is indeed better than the current best algorithm. If it is, the new competitor is used as the new best algorithm.

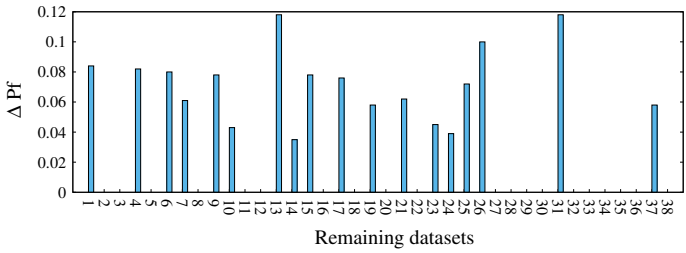

Fig. 5 Values of ΔPf of a potential competitor with respect to a_{best} on all datasets

Table 3 Determining the best competitor among different alternatives

Algorithm	$\sum \Delta Pf$
a_1	0.587
a_2	**3.017**
a_3	0.143
a_4	0.247
a_5	1.280

The bold value indicates the highest estimated performance gain

Table 4 MIL values of AT-A3R for different parameter setting of P

P	1	1/2	1/4	1/8	1/16	1/32	1/64	1/128	0
MIL	0.846	0.809	0.799	0.809	**0.736**	0.905	1.03	1.864	3.108

The bold value indicates the best value

Step 12: In this step the loss-time curve L_i is returned together with the final best algorithm a_{best} identified.

4.2 Optimizing the parameter settings for AT-A3R

To use AT-A3R in practice we need to determine a good setting of parameter P in A3R used within AT-A3R. We have considered different values shown in Table 4. The last value shown, P = 0, represents a situation when only accuracy matters.

The empirical results presented in this table indicate that the optimal parameter setting for P is 1/16 for the datasets used in our experiments. We will refer to this variant of active testing as AT-A3R-1/16, or AT* for simplicity. We note, however, that the MIL values do not vary much when the values of P are larger than 1/16 (i.e. 1/4 etc.). For all these setting time is given a high importance.

When time is ignored, which corresponds to the setting of P = 0 and the version is AT-A3R-0. For simplicity, this version will be referred to as AT0 in the rest of this article.

The MIL values in this table represent *mean interval values* obtained in different cycles of the leave-one-out mode. Individual values vary quite a lot for different datasets, as can be seen in the boxplots in Fig. 6. The loss-time curves for some of the variants are shown in Fig. 7.

This study has provided an evidence that the AT method too works quite well when time is taken into consideration. When time is ignored (version AT0), the results are quite poor

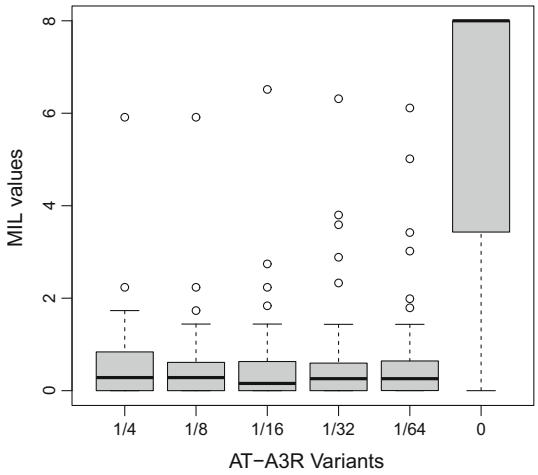

Fig. 6 Boxplot showing the distribution of MIL values for the methods in Table 4

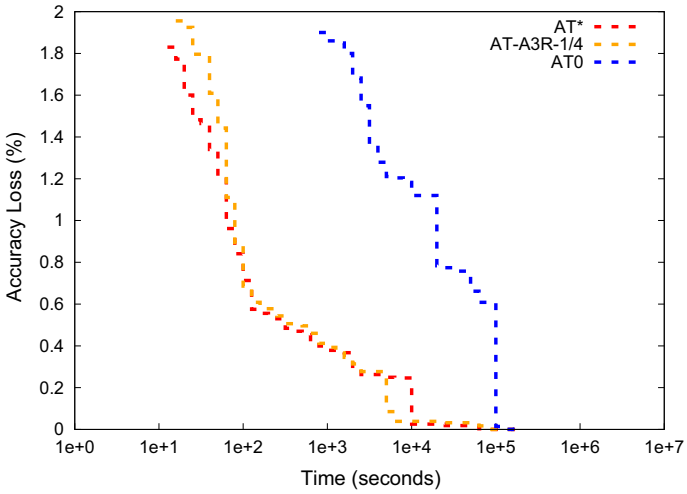

Fig. 7 Mean loss-time curves for AT-A3R with different settings for P

(MIL = 3.108). But if we compare AT0 and AR0 approaches, the AT0 result is not so bad in comparison.

The fact that AT0 achieved much better value that AR0 can be explained by the initialization step used in Algorithm 1. We note that the AR* has been used to initialize the value of a_{best}. This version takes runtime into account. If AR0 were used instead, the MIL of AT0 would increase to 21.89%, that is a value comparable to AR0.

The values shown in Table 4 and the accompanying boxplot indicate that the MIL scores for the AT method have relatively high *variance*. One plausible explanation for this is the following. The method selects the best competitor on the basis of the estimate of the highest performance gain. Here the topmost element is used in an ordered list. However, there may be other choices with rather similar value, albeit a bit smaller, which are ignored. If the conditions are changed slightly, the order in the list changes and this affects the choice of the best competitor and all subsequent steps.

Table 5 MIL values of the AR and AT variants described above

Method	AR*	AT*	AR0	AT0
MIL	**0.531**	0.736	22.11	3.108

The bold value indicates that the AR* method achieves the lowest loss

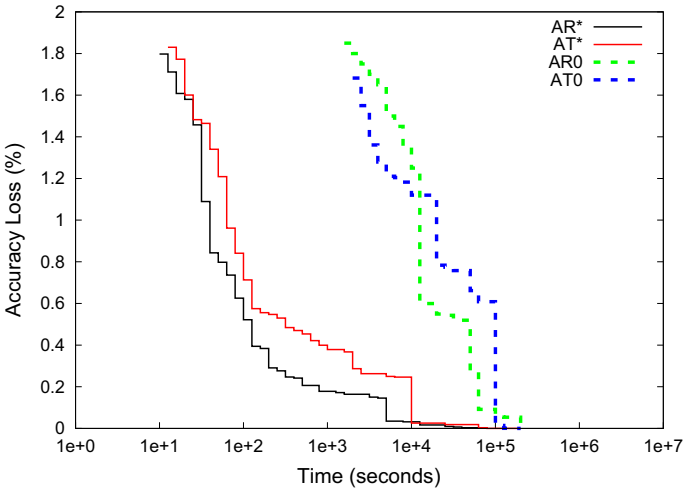

Fig. 8 Mean loss-time curves of the AR and AT variants described above

4.3 Comparison of average rank and active testing method

In this section we present a comparison of the two upgraded methods discussed in this article, the average ranking method and the active testing method (the hybrid variant) with optimized parameter settings. Both are also compared to the original versions based on accuracy. To be more precise, the comparison involves:

- AR*: Upgraded average ranking method, described in Sect. 3;
- AT*: Upgraded active testing method, described in the preceding section;
- AR0: Average ranking method based on accuracy alone;
- AT0: Active testing method based on accuracy alone;

The MIL values for the four variants above are presented in Table 5. The corresponding loss curves are shown in Fig. 8. Note that the curve for AR* is the same curve shown earlier in Fig. 3.

The results show that the upgraded versions of AR and AT that incorporate both accuracy and runtime lead to much better loss values (MIL) than their accuracy-based counterparts. The corresponding loss curves are shown in Fig. 8.

Statistical tests were used to compare the variants of algorithm selection methods presented above. Following (Demšar 2006) the Friedman test was used first to determine whether the methods were significantly different. As the result of this test was positive, we have carried out Nemenyi test to determine which of the methods are (or are not) statistically different. The data used for this test is shown in Table 6. For each of the four selection methods the table shows the individual MIL values for the 39 datasets used in the experiment.

Table 6 MIL values for the four meta-learners mentioned in Fig. 8 on different datasets

Dataset	AR*		AR0		AT*		AT0	
	MIL	Rank	MIL	Rank	MIL	Rank	MIL	Rank
Anneal.ORIG	0.00	1.0	4.50	4.0	0.05	2.0	0.95	3.0
Kr-vs-kp	0.01	1.0	10.77	4.0	0.03	2.0	4.67	3.0
Letter	1.07	1.0	83.94	4.0	2.60	2.0	5.26	3.0
Balance-scale	0.34	1.0	0.49	3.0	0.47	2.0	0.70	4.0
Mfeat-factors	0.70	3.0	45.58	4.0	0.70	2.0	0.67	1.0
Mfeat-fourier	0.76	2.0	31.39	4.0	0.43	1.0	1.79	3.0
Breast-w	0.00	1.5	1.10	4.0	0.00	1.5	0.08	3.0
Mfeat-karhunen	0.21	2.0	30.81	4.0	0.08	1.0	0.43	3.0
Mfeat-morphol.	0.37	1.0	13.01	4.0	0.65	2.0	3.49	3.0
Mfeat-pixel	0.00	2.0	73.88	4.0	0.00	2.0	0.00	2.0
Car	0.53	1.0	3.82	4.0	0.98	2.0	3.04	3.0
Mfeat-zernike	2.02	2.0	28.82	4.0	1.980	1.0	5.75	3.0
Cmc	0.24	1.0	4.11	4.0	0.36	2.0	1.98	3.0
Mushroom	0.00	1.5	30.97	4.0	0.00	1.5	0.02	3.0
Nursery	0.27	1.0	28.45	4.0	0.28	2.0	5.34	3.0
Optdigits	0.56	1.0	57.70	4.0	0.71	2.0	1.43	3.0
Credit-a	0.01	1.0	0.56	4.0	0.01	2.0	0.41	3.0
Page-blocks	0.03	1.0	3.11	4.0	0.09	2.0	0.54	3.0
Credit-g	0.00	1.0	0.53	4.0	0.00	2.0	0.05	3.0
Pendigits	0.29	1.0	55.04	4.0	0.41	2.0	1.21	3.0
Cylinder-bands	0.00	1.5	8.17	4.0	0.00	1.5	1.18	3.0
Segment	0.00	1.0	25.05	4.0	0.03	2.0	1.21	3.0
Diabetes	0.00	2.0	0.83	4.0	0.00	2.0	0.00	2.0
Soybean	0.00	1.0	47.29	4.0	0.00	2.0	0.01	3.0
Spambase	0.06	1.0	17.25	4.0	0.19	2.0	1.79	3.0
Splice	0.41	2.0	34.46	4.0	0.31	1.0	0.44	3.0
Tic-tac-toe	0.07	2.0	2.45	3.0	0.00	1.0	9.84	4.0
Vehicle	1.34	1.0	5.76	4.0	1.85	2.0	5.12	3.0
Vowel	0.68	1.0	17.89	4.0	1.05	2.0	10.50	3.0
Waveform-5000	0.96	2.0	32.62	4.0	0.71	1.0	1.81	3.0
Electricity	3.12	2.0	23.18	4.0	2.71	1.0	14.09	3.0
Solar-flare	0.00	1.5	0.01	3.0	0.00	1.5	0.14	4.0
Adult	0.76	2.0	9.61	4.0	0.72	1.0	1.84	3.0
Yeast	0.00	1.0	4.25	4.0	0.23	2.0	2.22	3.0
Satimage	0.27	1.0	36.29	4.0	0.90	2.0	2.51	3.0
Abalone	0.42	1.0	9.01	4.0	0.59	2.0	1.01	3.0
Kropt	5.07	1.0	58.64	4.0	9.22	2.0	21.35	3.0
Baseball	0.12	1.0	1.57	3.0	0.21	2.0	4.90	4.0
Eucalyptus	0.03	1.0	19.30	4.0	0.16	2.0	3.47	3.0
Mean	0.53	1.4	22.11	3.9	0.74	1.7	3.11	3.0

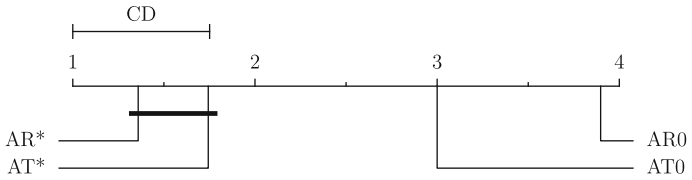

Fig. 9 Results of Nemenyi test. Variants that are connected by a horizontal line are statistically equivalent

Statistical tests require that the MIL values be transformed into ranks. We have done that and the resulting ranks are also shown in this table. The mean values are shown at the bottom of the table. If we compare the mean values of AR* and AT*, we note that AR* is slightly better than AT* when considering MILs, but the ordering is the other way round when considering ranks.

Figure 9 shows the result of the statistical test discussed earlier. The two best variants are A3R-based Average Ranking (AR*) and the active testing method AT*. Although AR* has achieved better performance (MIL), the statistical test indicates that the difference is not statistically significant. In other words, the two variants are statistically equivalent.

Both of these methods outperform their accuracy based counterparts, namely AT0 and AR0. The reasons for this were already explained earlier. This is due to the fact that the accuracy based variants tend to select slow algorithms in the initial stages of the testing. This is clearly a wrong strategy, if the aim is to identify algorithms with reasonable performance relatively fast.

An interesting question is whether the AT* method could ever beat AR and, if so, under which circumstances. We believe this could happen if much larger number of algorithms were used. As we have mentioned earlier, in this study we have used 53 algorithms, which is a relatively modest number by current standards. If we were to consider variants of algorithms with different parameter settings, the number of algorithm configurations would easily increase by 1–2 orders of magnitude. We expect that under such circumstances the active testing method would have an advantage over AR. AR would tend to spend a lot of time evaluating very similar algorithms rather than identifying which candidates represent good competitors.

5 Effect of incomplete meta-data on average ranking

Our aim is to investigate the issue of how the generation of the average ranking is affected by incomplete test results in the meta-dataset available. The work presented here focuses on the AR* ranking discussed earlier in Sect. 3. We wish to see how robust the method is to omissions in the meta-dataset. This issue is relevant because meta-datasets that have been gathered by researchers are very often incomplete. Here we consider two different ways in which the meta-dataset can be incomplete: First, the test results on some datasets may be completely missing. Second, there may be certain proportion of omissions in the test results of some algorithms on each dataset.

The expectation is that the performance of the average ranking method would degrade when less information is available. However, an interesting question is how grave the degradation is. The answer to this issue is not straightforward, as it depends greatly on how diverse the datasets are and how this affects the rankings of algorithms. If the rankings are very similar, then we expect that the omissions would not make much difference. So the issue of the effects of omissions needs to be relativized. To do this we will investigate the following issues:

Table 7 Missing test results on certain percentage of datasets (MTD)

Algorithms	D_1	D_2	D_3	D_4	D_5	D_6
a_1	0.85		0.77	0.98		0.82
a_2	0.95		0.67	0.68		0.72
a_3	0.63		0.55	0.89		0.46
a_4	0.45		0.34	0.58		0.63
a_5	0.78		0.61	0.34		0.97
a_6	0.67		0.70	0.89		0.22

Table 8 Missing test results on certain percentage of algorithms (MTA)

Algorithms	D_1	D_2	D_3	D_4	D_5	D_6
a_1	0.85	0.77		0.98		0.82
a_2		055	0.67	0.68	0.66	
a_3	0.63		0.55	0.89		0.46
a_4	0.45	0.52	0.34		0.44	0.63
a_5	0.78	0.87	0.61	0.34	0.42	
a_6		0.99		0.89		0.22

– Effects of missing test results on X% of datasets (alternative MTD);
– Effects of missing X% of test results of algorithms on each dataset (alternative MTA).

If the performance drop of alternative MTA were not too different from the drop of alternative MTD, then we could conclude that X% of omissions is not unduly degrading the performance and hence the method of average ranking is relatively robust. Each of these alternatives is discussed in more detail below.

Missing all test results on some datasets (alternative MTD): This strategy involves randomly omitting all test results on a given proportion of datasets from our meta-dataset. An example of this scenario is depicted in Table 7. In this example the test results on datasets D_2 and D_5 are completely missing. The aim is to show how much the average ranking degrades due to these missing results.

Missing some algorithm test results on each dataset (alternative MTA): Here the aim is to drop a certain proportion of test results on each dataset. The omissions are simply distributed uniformly across all datasets. That is, the probability that the test result of algorithm a_i is missing is the same irrespective of which algorithm is chosen. An example of this scenario is depicted in Table 8.

The proportion of test results on datasets/algorithms omitted is a parameter of the method. Here we use the values shown in Table 9. We use the same meta-dataset described earlier in this article. This dataset was used to obtain a new one in which the test results of some datasets, chosen at random, would be obliterated. The resulting dataset was used to construct the average ranking. Each ranking was then used to construct a *loss-time curve*. The whole process was repeated 10 times. This way we would obtain 10 loss-time curves, which would be aggregated into a single loss-time curve. Our aim is to upgrade the average ranking method to be able to deal with incomplete rankings. The enhanced method (AR*-MTA-H) is described in the next section. It can be characterized as a heuristic method of aggregation of incomplete rankings that uses weights of ranks. Later it is compared to the classical approach (AR*-MTA-B), that serves as a baseline here. This method is based on the original Borda's method (Lin 2010) and is commonly used by many researchers.

Table 9 Percentages of omissions and the numbers of datasets and algorithms used

Omissions %	0	5	10	20	50	90	95
No of datasets used in MTD	38	36	34	30	19	4	2
No of tests per dataset in MTA	53	50	48	43	26	5	3

Table 10 An example of two rankings R_1 and R_2 and the aggregated ranking R^A

R_1	Rank	R_2	Rank	R^A	Rank	Weight
a_1	1	a_2	1	a_1	1.67	1.2
a_3	2	a_1	2	a_3	2	1
a_4	3			a_4	3	1
a_2	4			a_2	3.5	1.2
a_6	5			a_6	5	1
a_5	6			a_5	6	1

5.1 Aggregation method for incomplete rankings (AR*-MTA-H)

Before describing the method, let us consider a motivating example (see Table 10), illustrating why we cannot simply use the usual average ranking method (Lin 2010), often used in comparative studies in machine learning literature.

Let us compare the rankings R_1 and R_2 (Table 10). We note that algorithm a_2 is ranked 4 in ranking R_1, but has rank 1 in ranking R_2. If we used the usual method, the final ranking of a_2 would be the mean of the two ranks, i.e. (4+1)/2=2.5. This seems intuitively not right, as the information in ranking R_2 is incomplete. If we carry out just one test and obtain ranking R_2 as a result, this information is obviously inferior to having conducted more tests leading to ranking R_1. So these observations suggest that the number of tests should be taken into account to set the weight to of the individual elements of the ranking.

In our method the weight is calculated using the expression $(N-1)/(Nmax-1)$, where N represents the number of filled-in elements in the ranking and $Nmax$ the maximum number of elements that could be filled-in. So, for instance, in the ranking R_1, $N = 6$ and $Nmax = 6$. Therefore, the weight of each element in the ranking is $5/5 = 1$. We note that $N - 1$ (i.e. 5), represents the number of non-transitive relations in the ranking, namely $a_1 > a_3, a_3 > a_4, .., a_6 > a_5$. Here $a_i > a_j$ is used to indicate that a_i is preferred to a_j.

Let us consider the incomplete ranking R_2. Suppose we know a priori that the ranking could include 6 elements and so $Nmax = 6$, as in the previous case. Then the weight of each element will be $(N - 1)/(Nmax - 1) = 1/5 = 0.2$. The notion of *weight* captures the fact that ranking R_2 provides less information than ranking R_1. We need this concept in the process of calculating the average ranking.

Our upgraded version of the aggregation method for incomplete rankings involves the initialization step, which consists of fetching the first ranking and using it to initialize the average ranking R^A. Then in each subsequent step a new ranking is read-in and aggregated with the average ranking, producing a new average ranking. The aggregation is done by going through all elements in the ranking, one by one. If the element appears in both the aggregated ranking and the read-in ranking, its rank is recalculated as a weighted average of the two ranks:

$$r_i^A := r_i^A * w_i^A/(w_i^A + w_i^j) + r_i^j * w_i^j/(w_i^A + w_i^j) \quad (7)$$

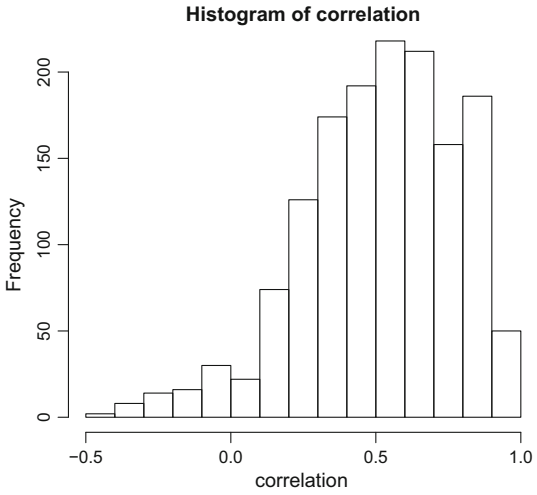

Fig. 10 Spearman's rank correlation coefficient between rankings for pairs of datasets

where r_i^A represents the rank of element i in the aggregated ranking and r_i^j the rank of the element i in the ranking j and w_i^A and w_i^j represent the corresponding weights. The weight is updated as follows:

$$w_i^A := w_i^A + w_i^j \tag{8}$$

If the element appears in the aggregated ranking, but not in the new read-in ranking, both the rank and the weight are kept unchanged.

Suppose the aim is to aggregate the rankings R_1 and R_2 shown before. The new rank of a_2 will be $r_2^A = 4 * 1/1.2 + 1*0.2/1.2 = 3.5$. The weight will be $w_2^A = 1 + 0.2 = 1.2$. The final aggregated ranking of rankings R_1 and R_2 is R^A as shown in Table 10.

5.2 Results on the effects of omissions in the meta-dataset

5.2.1 Characterization of the meta-dataset

We were interested in analyzing different rankings of classification algorithms on different datasets used in this work and, in particular, how these differ for different pairs of datasets. If two datasets are very similar, the algorithm rankings will also be similar and, consequently, the correlation coefficient will be near 1. On the other hand, if the two datasets are quite different, the correlation will be low. In the extreme case, the correlation coefficient will be -1 (i.e. one ranking is the inverse of the other). So the distribution of pairwise correlation coefficients provides an estimate of how difficult the meta-learning task is.

Figure 10 shows a histogram of correlation values. The histogram is accompanied by *expected value*, *standard deviation* and *coefficient of variation* calculated as the ratio of standard deviation to the expected value (mean) (Witten and Frank 2005). These measures are shown in Table 11.

5.2.2 Study of accuracy-based rankings and one paradox

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

Table 11 Measures characterizing the histogram of correlations in Fig. 10

Measure %	Expected value	SD	Coefficient of variation
Value	0.5134	0.2663	51.86%

Table 12 Mean interval loss (MIL) values for different percentage of omissions

Method	% Omissions						
	0	5	10	20	50	90	95
AR*-MTD	22.11	21.09	20.66	20.12	19	17.92	15.15

combines accuracy and runtime discussed in Sect. 3.3. In the first study we have focused solely on the average ranking based on accuracy. When studying how omissions of tests on datasets affect rankings and the corresponding loss time curves, we became aware of one paradox that can arise under certain conditions. This paradox can be formulated as follows:

Suppose we use tests of a given set of algorithms on a set of M (N) datasets leading to M (N) accuracy-based rankings. Suppose the M (N) rankings are aggregated to construct the average ranking. Suppose that $M > N$ (for instance, consider M=10 and N=1). Let AR_M represent the average ranking elaborated on the basis M rankings obtained on the respective M datasets. We can then expect that MIL of the variant AR_M would be lower (better) than the MIL of AR_N. However, we have observed that this was exactly the other way round.

Table 12 provides evidence for the observation above. When all datasets are used to construct the average ranking (omissions are 0%) the MIL is 22.11. When only 5% are used (omissions are 95%), the MIL value is lower, i.e. 15.15. In other words, *using more information leads to worse results!*

Our explanation for this paradox is as follows. The accuracy-based average ranking orders the algorithms by accuracy. If we use several such rankings constructed on different datasets, we can expect that we get an ordering where the algorithms with high accuracy on many datasets will appear in the initial positions of the ranking. These algorithms tend to be slower than the ones that require less time to train. So if we use this ranking and use loss time curves in the evaluation, we need to wait a long time before the high-accuracy algorithms get executed. This does not happen so much if use a ranking constructed on fewer datasets.

We have carried out additional experiments that support this explanation. First, if we use just test curves, where each test lasts one unit of time, the paradox disappears. Also, if we use A3R to rank the algorithms, again the problem disappears. In these situations we can see that if we use more information, the result is normally better.

In conclusion, to avoid the paradox, it is necessary to *use a similar criterion* both in the construction of rankings and in the process of loss curve construction.

5.2.3 Study of AR ranking methods and the results*

In the second study we have focused on AR*. Table 13 presents the results for the alternatives *AR*-MTD*, *AR*-MTA-H* and *AR*-MTA-B* in terms of mean interval loss (MIL). All loss-time curves start from the initial loss of the default classification. This loss is calculated as the difference in performance between the best algorithm and the default accuracy for each dataset. The default accuracy is calculated in the usual way, by simply predicting the

Table 13 Mean interval loss (MIL) values for different percentage of omissions

Method	Omission%						
	0	5	10	20	50	90	95
AR*-MTD	0.531	0.535	0.535	0.536	0.550	1.175	1.633
AR*-MTA-H	0.531	0.534	0.537	0.542	0.590	1.665	2.042
AR*-MTA-B	0.531	0.536	0.537	0.544	0.593	2.970	3.402
AR*-MTA-H/AR*-MTD	1.00	1.00	1.00	1.01	1.07	1.42	1.25

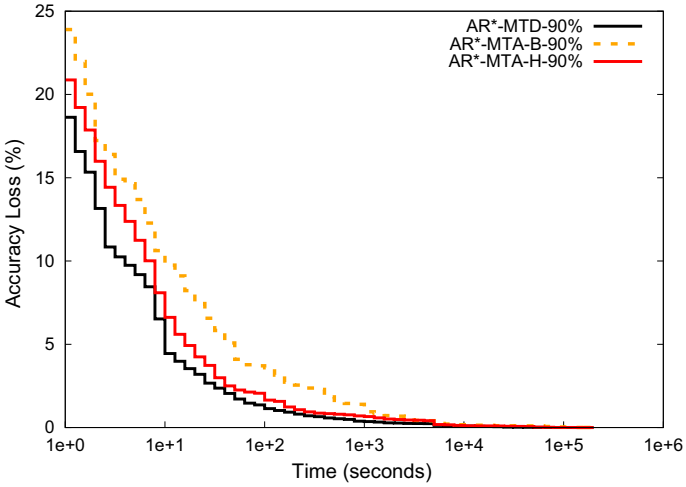

Fig. 11 Comparison of *AR*-MTA-H* with *AR*-MTD* and the baseline method *AR*-MTA-B* for 90% of omissions

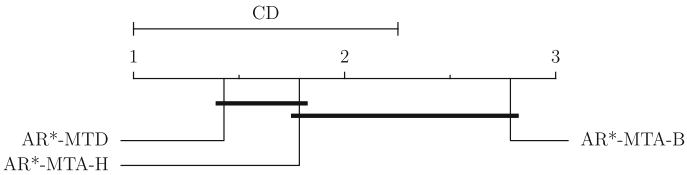

Fig. 12 Results of Nemenyi test, variants of the meta-learners that are connected by a horizontal line are statistically equivalent

majority class for the dataset in question. The values for the ordinary average ranking method, *AR*-MTA-B*, are also shown, as this method serves as a baseline.

Figure 11 shows the loss-time curves for the three alternatives when the number of omissions is 90%. Not all loss-time curves are shown, as the figure would be rather cluttered.

Our results show that the proposed average ranking method *AR*-MTA-H* achieves better results than the baseline method *AR*-MTA-B*. We note also that although the proposed method *AR*-MTA-H* achieves comparable results to *AR*-MTD* for many values of the percentage drop (all values up to about 50%). Only when we get to rather extreme values, such as 90%, the difference is noticeable. Still, the differences between our proposed variant *AR*-MTA-H* and *AR*-MTD* are smaller than the differences between *AR*-MTA-B* and *AR*-MTD*.

These above observations are supported also by the results of a statistical test. The values shown in Table 13 were used to conduct a Nemenyi test and the result is shown in Fig. 12. These results indicate that the proposed average ranking method is relatively robust to omissions.

6 Conclusions

Upgrading AT and AR methods to be more effective

In this paper we addressed an approach for algorithm selection where the recommendation is presented in the form of a ranking. The work described here extends two existing methods described earlier in the literature. The first one was *average ranking (AR)* and the second *active testing (AT)*.

The first method (AR) is a rather simple one. It calculates an average ranking for all algorithms over all datasets. We have shown how to upgrade this method to incorporate the measure A3R that combines accuracy and run time. The novelty here lies in the use of A3R, instead of just accuracy. The second method (AT) employs tests to identify the most promising candidates, as its aim is to intelligently select the next algorithm to test on the new dataset. We have also shown how this method can be to upgraded to incorporate the measure A3R that combines accuracy and runtime.

Establishing the correct balance between accuracy and runtime

We have also investigated the issue of how to establish the correct balance between accuracy and runtime. Giving too much weight to time would promote rather poor algorithms and hence time could be wasted. Giving too little weight to time would work in the opposite direction. It would promote tests on algorithms which exhibit good performance in general, but may occasionally fail. Unfortunately, such algorithms tend to be rather slow and so their incorporation early in the testing process may actually delay the process of identifying good algorithms early.

It is thus necessary to establish the correct balance between the weight given to accuracy and the weight given to time. As we have shown in Sect. 3.3.2, this can be done by determining an appropriate setting for parameter P used within function A3R that combines accuracy and runtime. One rather surprising consequence of this is the following: if the user wished to impose his preference regards the relative importance of accuracy and runtime, he/she would probably end up with a worse result than the one obtained with our setting.

Evaluation methodology using loss-time curves

The experimental results were presented in the form of loss-time curves. This representation, where time is represented on a logarithmic scale, has the following advantages. First, it stresses the losses at the beginning of the curve, corresponding to the initial tests. This is justified by the fact that users would normally prefer to obtain good recommendations as quickly as possible.

We note that testing without a cut-off on time (budget) is not really practicable, as users are not prepared to wait beyond a given limit. On the other hand, many applications may allow that results come only after some small amount of time. This is why here we focused on the selection process within a given time interval. Having a fixed time interval has advantages, as it is possible to compare different variants just by comparing the mean interval loss (MIL).

The results presented earlier, see Sect. 4.3, show that the upgraded versions of AR* and AT* lead to much better results in terms of mean loss values (MIL) than their accuracy-based

counterparts. In other words, the incorporation of both accuracy and runtime into the methods pays off.

Disastrous results of AR0 and a paradox identified

We have shown that if we do not incorporate both accuracy and runtime, the variant AR0 that uses only accuracy leads to disastrous results in terms of MIL. We have provided an explanation for this phenomenon and drawn attention to an apparent paradox, as more information seems to lead to worse results.

Comparison of AT and AR* methods*

When comparing the AT* and AR* methods, we note that both methods lead to comparable results. We must keep in mind, however, that our meta-dataset included test results on 53 algorithms only. If more candidate algorithms were used, we expect that AT* method would lead to better results that AR*.

The effects of incomplete test results on AR method*

We have also investigated the problem of how the process of generating the average ranking is affected by incomplete test results by describing a relatively simple method, *AR*-MTA-H*, that permits to aggregate incomplete rankings. We have also proposed a methodology that is useful in the process of evaluating our aggregation method. This involves using a standard aggregation method *AR*-MTD* on a set of complete rankings, but whose number is decreased following the proportion of omissions in the incomplete rankings. As we have shown, the proposed aggregation method achieves quite comparable results and is relatively robust to omissions in test results in the test data. We have shown that a percentage drop of up to 50% does not make much difference. As the incomplete meta-dataset does not affect much the final ranking and the corresponding loss, this could be explored in future design of experiments, when gathering new test results.

6.1 Discussion and future work

In this section we present some suggestions on what could be done in future work.

*Verifying the conjecture regarding AT**

In Sect. 4.3 we have compared the two best variants - AT* and AR*. We have shown that the results obtained with the AT* variant are slightly worse than those obtained using AR*. However, the two results are statistically equivalent and it is conceivable that if the number of algorithms or their variants were increased, the AT* method could win over AR*. Experiments should be carried out to verify whether this conjecture holds. Also, we plan to re-examine the AT method to see if it could be further improved.

Would the best setting of P transfer to other datasets?

An interesting question is *how stable* the best setting for the parameter P (parameter used within A3R) is. This could be investigated in the following manner. First we can gather a large set of datasets and draw samples of datasets of fixed size at random. We would establish the optimum P_i for sample i and then determine what we would be the loss on sample j if P_i were used, rather than the optimized value P_j.

How does P vary for different budgets?

Another question is whether the optimum value of P varies much if the budget (T_{max}) is altered. A more general question involves the interval T_{min} - T_{max}. Further studies should be carried out, so that these questions could be answered.

Applying AT both to algorithm selection and hyper-parameter tuning

We should investigate whether the AT method can be extended to handle both the selection of learning algorithms and parameter settings. This line of work may require the incorporation of techniques used in the area of parameter settings (e.g. [?], among others).

Combining AT with classical approaches to meta-learning

Another interesting issue is whether the AT approach could be extended to incorporate the classical dataset characteristics and whether this would be beneficial. Earlier work (Leite and Brazdil 2008) found that the AT approach achieves better results than classical meta-models based on dataset characteristics. More recently, it was shown (Sun and Pfahringer 2013) that pairwise meta-rules were more effective than the other ranking approaches. Hence, it is likely that the AT algorithm could be extended by incorporation information of dataset characteristics and pairwise meta-rules.

Leave-one-out CV versus k-fold cross-validation

All methods discussed in this paper were evaluated using the leave-one-out cross-validation (LOO-CV). All datasets except one were used to refine our model (e.g. AR). The model was then applied to the dataset that was left out and evaluated by observing the loss curve. We have opted for this method, as our meta-dataset included test results on a modest number of datasets (39). As LOO-CV may suffer from high variance, 10-fold CV could be used in future studies.

Study of the effects of incomplete meta-datasets

In most work on meta-learning it is assumed that the evaluations to construct the meta-dataset can be carried out off-line and hence the cost (time) of gathering this data is ignored. In future work we could investigate approaches that permit to consider also the costs (time) of off-line tests. Their cost (time) 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. The approaches discussed in this paper could be upgraded so that they would minimize the overall costs.

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; FCT/MEC through PIDDAC and ERDF/ON2 within project NORTE-07-0124-FEDER-000059 and through the COMPETE Programme (operational programme for competitiveness) and by National Funds through the FCT Portuguese Foundation for Science and Technology within project FCOMP-01-0124-FEDER-037281; Grant 612:001:206 from the Netherlands Organisation for Scientific Research (NWO). We wish to express our gratitude to all anonymous referees for their detailed comments which led to various improvements of this paper. Also, our thanks to Miguel Cachada for reading through the paper and his comments and in particular one very useful observation regarding the AT method.

A Algorithm ranks

See Table 14.

Table 14 Algorithms used in the experiment and their ranks ordered using AR*. All classifiers as implemented in Weka 3.7.12. For full details, see `https://www.openml.org/s/37`

Algorithm	Rank-AR*	Rank-AR0
A1DE	1	8
Bagging(RandomTree)	2	14
NaiveBayes	3	35
RandomCommittee(RandomTree)	4	15
NaiveBayesUpdateable	5	36
BayesNet(K2)	6	32
END(ND(J48))	7	13
Bagging(J48)	8	9
LogitBoost(DecisionStump)	9	17
RandomForest	10	3
J48	11	28
RandomSubSpace(REPTree)	12	21
MultiBoostAB(J48)	13	10
SMO(PolyKernel)	14	11
AdaBoostM1(REPTree)	15	18
Bagging(REPTree)	16	22
REPTree	17	37
MultiBoostAB(REPTree)	18	19
SimpleLogistic	19	4
Bagging(NaiveBayes)	20	34
AdaBoostM1(J48)	21	16
PART	22	30
HoeffdingTree	23	33
ClassificationViaRegression(M5P)	24	12
LMT	25	1
JRip	26	31
AdaBoostM1(RandomTree)	27	40
AdaBoostM1(NaiveBayes)	28	26
AdaBoostM1(IBk)	29	24
RandomTree	30	41
Bagging(JRip)	31	6
MultiBoostAB(RandomTree)	32	38
MultiBoostAB(JRip)	33	7
IterativeClassifierOptimizer(LogitBoost(DecisionStump))	34	20
MultiBoostAB(NaiveBayes)	35	29

Table 14 continued

Algorithm	Rank-AR*	Rank-AR0
LADTree	36	25
Bagging(LMT)	37	2
Dagging(DecisionStump)	38	42
OneR	39	45
NBTree	40	23
AdaBoostM1(LMT)	41	5
DTNB	42	27
DecisionStump	43	50
AdaBoostM1(DecisionStump)	44	43
Bagging(OneR)	45	44
SMO(RBFKernel)	46	39
AdaBoostM1(OneR)	47	47
MultiBoostAB(OneR)	48	46
MultiBoostAB(DecisionStump)	49	48
Bagging(DecisionStump)	50	49
ZeroR	51	52
CVParameterSelection(ZeroR)	52	53
RacedIncrementalLogitBoost(DecisionStump)	53	51

References

Abdulrahman, S. M., & Brazdil, P. (2014). Measures for combining accuracy and time for meta-learning. *Meta-Learning and Algorithm Selection Workshop at ECAI*, *2014*, 49–50.

Brazdil, P. & Soares, C. (2000). A Comparison of ranking methods for classification algorithm selection. In *Machine Learning: ECML 2000*, pp. 63–75. Springer.

Brazdil, P., Soares, C., & da Costa, J. P. (2003). Ranking learning algorithms: Using IBL and meta-learning on accuracy and time results. *Machine Learning*, *50*(3), 251–277.

Brazdil, P., Carrier, C . G., Soares, C., & Vilalta, R. (2008). *Metalearning: Applications to data mining*. Berlin: Springer.

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research*, *7*, 1–30.

Fedorov, V . V. (1972). *Theory of optimal experiments*. Cambridge: Academic Press.

Feurer, M., Springenberg, T. & Hutter, F. (2015). Initializing bayesian hyperparameter optimization via meta-learning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, January 2015.

Fürnkranz, J. & Petrak, J. (2001). An evaluation of landmarking variants. In *Working Notes of the ECML/PKDD 2000 Workshop on Integrating Aspects of Data Mining, Decision Support and Meta-Learning*, pp. 57–68.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. (2009). The WEKA data mining software: An update. *ACM SIGKDD Explorations Newsletter*, *11*(1), 10–18.

Jankowski, N. (2013). Complexity measures for meta-learning and their optimality. In *Algorithmic Probability and Friends. Bayesian Prediction and Artificial Intelligence.*, pp. 198–210.

Leite, R. & Brazdil, P. (2008). Selecting classifiers using meta-learning with sampling landmarks and data characterization. In *Proceedings of the 2nd Planning to Learn Workshop (PlanLearn) at ICML/COLT/UAI 2008*, pp. 35–41.

Leite, R. & Brazdil, P. (2010). Active testing strategy to predict the best classification algorithm via sampling and metalearning. In *ECAI*, pp. 309–314.

Leite, R., Brazdil, P. & Vanschoren, J. (2012). Selecting classification algorithms with active testing. In *Machine Learning and Data Mining in Pattern Recognition*, pp. 117–131. Springer.

Li, L., Chu, W., Langford, J. & Schapire, R. E. (2010). A contextual-bandit approach to personalized news article recommendation. In *Proceedings of 19th WWW*, pp. 661–670. ACM.

Lin, S. (2010). Rank aggregation methods. *WIREs Computational Statistics*, *2*, 555–570.

Long, B., Chapelle, O., Zhang, Y., Chang, Y., Zheng, Z. & Tseng, B. (2010). Active learning for ranking through expected loss optimization. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 267–274. ACM.

Neave, H. R., & Worthington, P. L. (1988). *Distribution-free Tests*. London: Unwin Hyman.

Pfahringer, B., Bensusan, H. & Giraud-Carrier, C. (2000). 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.

Pihur, V., Datta, S., & Datta, S. (2009). RankAggreg, an R package for weighted rank aggregation. *BMC Bioinformatics*, *10*(1), 62.

Prudêncio, R. B. & Ludermir, T. B. (2007). Active selection of training examples for meta-learning. In *7th International Conference on Hybrid Intelligent Systems, 2007. HIS 2007.*, pp. 126–131. IEEE.

Rice, J. R. (1976). The algorithm selection problem. *Advances in Computers*, *15*, 65–118.

Smith-Miles, K . A. (2008). Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys (CSUR)*, *41*(1), 6:1–6:25.

Sun, Quan, & Pfahringer, B. (2013). Pairwise meta-rules for better meta-learning-based algorithm ranking. *Machine Learning*, *93*(1), 141–161.

van Rijn, J. N., Abdulrahman, S. M., Brazdil, P. & Vanschoren, J. (2015). Fast algorithm selection using learning curves. In *Advances in Intelligent Data Analysis XIV*, pp. 298–309. Springer.

Vanschoren, J., van Rijn, J. N., Bischl, B., & Torgo, L. (2014). OpenML: Networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, *15*(2), 49–60.

Witten, I . H., & Frank, E. (2005). *Data mining: Practical machine learning tools and techniques*. Burlington: Morgan Kaufmann.

Xu, L., Hutter, F., Hoos, H., & Leyton-Brown, K. (2011). *Hydra-MIP: Automated algorithm configuration and selection for mixed integer programming*. In RCRA workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion at the International Joint Conference on Artificial Intelligence (IJCAI)