Fast Algorithm Selection using Learning Curves

Jan N. van Rijn[1], Salisu Mamman Abdulrahman[2],
Pavel Brazdil[2], and Joaquin Vanschoren[3]

[1] Leiden University, Leiden, Netherlands,
j.n.van.rijn@liacs.leidenuniv.nl
[2] University of Porto, Porto, Portugal,
{sma,pbrazdil}@inescporto.pt
[3] Eindhoven University of Technology, Eindhoven, Netherlands,
j.vanschoren@tue.nl

Abstract. One of the challenges in Machine Learning is given a dataset, find a classifier that works well on it. Performing a cross-validation evaluation procedure on all possible combinations typically takes too much time; many solutions have been proposed that attempt to predict which classifiers are most promising to try. As the first recommended classifier is not always the correct choice, multiple recommendations should be made, making this a ranking problem rather than a classification problem. Even though this is a well studied problem, there is currently no good way of evaluating such rankings. We advocate the use of Loss Time Curves, as used in optimization literature. These visualize the amount of budget (time) needed to converge to a acceptable solution.
We investigate a method that utilizes the measured performances of classifiers on small samples of data to make such recommendation, and adapt it so that it works well in Loss Time space. Experimental results show that this method converges extremely fast to an acceptable solution.

Keywords: Algorithm Selection, Meta-Learning, Subsampling

1 Introduction

When presented with a new classification problem, a key challenge is to identify a classifier that obtains good predictive performance and its optimal parameter settings. This problem is known as the *Algorithm Selection Problem* [13]. Since many classifiers exist, all containing a number of parameters that potentially influence predictive performance, this is a challenging problem. Performing a cross-validation evaluation procedure on all possible combinations of classifiers and parameters (e.g., using a grid search) is typically infeasible, as this would take too much time. The field of *meta-learning* attempts to solve this by learning from prior examples. Typically, a set of classifiers is recommended based on performance on similar datasets.

The meta-learning method SAM [8] identifies similar datasets based on the *learning curves* of classifiers trained on them, and recommends the classifier that performs best on these similar datasets. Although the results are convincing,

it does not take into account some important aspects of algorithm selection. First, it only recommends the single best classifier, rather than a ranking of candidates. Second, it does not take the training time of the models into account. Indeed, in practical applications there is typically a budget (e.g., limited time or a maximum number of cross-validation runs) within which a number of classifiers can be evaluated. As such, the meta-learning method should be evaluated on how well it performs within a given budget.

Our contributions are the following. We extend the aforementioned technique so that it produces a ranking of classifiers and takes into account the run times of classifiers. Furthermore, we study the performance of this method in Loss space [9], taking into account both predictive accuracy and spent time. We will argue that Loss Curves as presented in [9] are biased, and propose the use of Loss Time Curves, as presented in [6]. Finally, we compare the method against a range of alternative methods, including a rather strong baseline that recommends the classifier that performed best on a small sample of the data [4]. Although our proposed technique dominates the baseline methods, the results suggest that the Best on Sample approach has been mistakenly neglected in the literature.

2 Related Work

Meta-learning aims to learn which learning techniques work well on what data [14]. A common task, known as the Algorithm Selection Problem [13], is to determine which classifier performs best on a given dataset. We can predict this by training a meta-model on meta-data comprised of dataset characterizations, i.e. *meta-features* [2], and the performances of different classifiers on these datasets. The same meta-features can be computed on each new dataset and fed to the meta-model to predict which classifiers will perform well.

Hence, the Algorithm Selection Problem is reduced to a Machine Learning problem. Meta-features are often categorized as either simple (number of examples, number of attributes), statistical (mean standard deviation of attributes, mean skewness of attributes), information theoretic (class entropy, mean mutual information) or landmarkers [11] (performance evaluations of simple classifiers). Many meta-learning studies in literature follow this approach [2, 11, 15, 18, 19].

However, meta-feature based approaches have some intrinsic limitations. First, it is hard to construct a meta-feature set that adequately characterize the problem space [7]. Second, the most successful meta-features, landmarkers, can be computationally expensive, limiting the options [11]. Finally, because not all classifiers run on all datasets, or take prohibitively long to do so, the meta-dataset usually contains many missing values, complicating the classification task.

In order to overcome these problems, Leite and Brazdil [7, 8] identify similar data sets based on partial learning curves. A learning curve is an ordered set of performance scores of a classifier on data samples of increasing size [12]. In this particular method, a *partial* learning curve is computed, using small samples, to measure how similarly algorithms behave on two data sets. As such, running classifiers on these samples is rather cheap.

Alternatively, the `Best on Sample` method uses the performance estimates of classifiers on a small sample and recommends the classifiers which perform best on this sample, in descending order [10]. Prior work is inconclusive about the performance. The authors of [10] suggest that this technique should be used as a baseline method in meta-learning research. The authors of [4] show that this information is not useful as landmarker. Indeed, it has been correctly observed that learning curves sometimes cross, i.e., one classifier can outperform another on a small data sample, but can be surpassed when trained on the whole dataset [7]. However, this happens less often as the sample size increases, making this method quite reliable when using the right sample size, as we will show in Section 4.

The datasets, learning curves and all results of our experiments are made publicly available on OpenML [17], for the purposes of verifiability, reproducibility and generalizability. OpenML is an experiment database [16] that enables the reproduction of earlier results for verification and reuse, and makes much larger studies (covering more classifiers and parameter settings) feasible. Moreover, experiment databases allow a variety of studies to be executed by a database look-up, rather than setting up new experiments.

3 Methods

We consider a set A of classifiers, a_m $(m = 1, 2, 3, \ldots, M)$. We also consider a set D of datasets, d_n $(n = 1, 2, 3, \ldots, N)$, and denote the size of dataset d_n as $|d_n|$. Let $P_{m,n,s}$ denote the performance (e.g., predictive accuracy) of classifier a_m on dataset d_n, using a sample size of s; $P_{m,n,\Omega}$ denotes the performance of classifier a_m on the full dataset d_n.

Let S be the set of data samples, of increasing size $s_t = 2^{5.5+0.5\times T}$ with $s = (1, 2, 3, \ldots, T)$, and T being a parameter set by the user. The samples follow a geometric increase, as suggested in [12]. It can be seen that $s_t \leq \lfloor \log_2 |d_n| \rfloor$. When using a higher value for T, larger samples are calculated, presumably yielding more accurate estimates at the cost of higher run times.

The distance between two datasets d_i and d_j can be determined using the following function [7]:

$$dist(d_i, d_j, a_p, a_q, T) = \sum_{t=1}^{T} (P_{p,i,s_t} - P_{p,j,s_t})^2 + \sum_{t=1}^{T} (P_{q,i,s_t} - P_{q,j,s_t})^2 \quad (1)$$

The distance function is related to Euclidean distance. It gives a measure how similar two datasets are, based on the learning curves of the two classifiers. Other work suggests a distance function that measures the Manhattan distance between learning curves, but experiments show that the difference in performance between these variants is minuscule [8].

Using either of these distance functions, k nearest datasets can be identified, and from the performance of both classifiers on these datasets we can predict which of the two will perform better on the new dataset. Controversially, it has

Algorithm 1 Pairwise Curve Comparison (PCC)

Require: d_{new}, $k \in \mathbb{N}^+$, $T \in \mathbb{N}^+$, *CurveAdaptation* $\in \{0,1\}$, *SmallerSample* $\in \{0,1\}$

Initialize A as a set of all classifiers
Initialize R as empty list
while $|A| > 0$ **do**
  $a_{best} \leftarrow$ Arbitrary element from A
  for all $a_{comp} \in A : a_{comp} \neq a_{best}$ **do**
    Initialize D as the set of all datasets on which a_{best} and a_{comp} were ran
    $votesBest = votesComp = 0$
    while $votesBest + votesComp < k$ **do**
      $d_{sim} \leftarrow \arg\min_{d_i \in D} dist(d_{new}, d_i, a_{best}, a_{comp}, T)$
      $coeff_{best} = coeff_{comp} = 1$
      $samp \leftarrow \Omega$
      if *CurveAdaptation* $= 1$ **then**
        $coeff_{best} \leftarrow f(d_{new}, d_{sim}, a_{best}, T)$
        $coeff_{comp} \leftarrow f(d_{new}, d_{sim}, a_{comp}, T)$
      end if
      if *SmallerSample* $= 1$ **and** $\lfloor \log_2 |d_{new}| \rfloor < \lfloor \log_2 |d_{sim}| \rfloor$ **then**
        $samp = \lfloor \log_2 |d_{new}| \rfloor$
      end if
      if $coeff_{best} \times P_{best,d_{sim},samp} > coeff_{comp} \times P_{comp,d_{sim},samp}$ **then**
        $votesBest \leftarrow votesBest + 1$
      else
        $votesComp \leftarrow votesComp + 1$
      end if
      $D \leftarrow D - d_{sim}$
    end while
    if $votesBest < votesComp$ **then**
      $a_{best} \leftarrow a_{comp}$
    end if
  end for
  $R \leftarrow R + a_{best}$
  $A \leftarrow A - a_{best}$
end while
return R {Ranking of classifiers in decreasing order}

been remarked that as the number of used samples increases, the performance of this technique decreases [7]. The authors of [7] speculate that the learning curves on the nearest datasets are still not similar enough, and propose *curve adaptation*, a technique that can adapt retrieved curves to the learning curves on the new dataset. In order to adapt a learning curve of classifier a_p on dataset d_r to dataset d_i, all points of the prior learning curve are multiplied by a coefficient:

$$f(d_i, d_r, a_p, T) = \frac{\sum_{t=1}^{T}(P_{p,i,s_t} \times P_{p,r,s_t} \times s_t^2)}{\sum_{t=1}^{T}((P_{p,r,s_t})^2 \times s_t^2)} \tag{2}$$

As not all datasets are of the same size, it is possible that a retrieved dataset has a much bigger size than the new dataset, which might give an unfair advantage for slow learners. In that case it might be beneficial to use the performance of the classifier at a sample size close to the full size of the new dataset.

Algorithm 1 shows the method in detail. It requires the new dataset as input, and values for parameters k (number of similar datasets to retrieve) and T (number of samples to use to build the partial learning curve), and boolean parameters whether to use the curve adaptation and smaller sample size option. The while-loop starting on line 3 identifies the most promising classifier left in A (lines 4–29), appends this classifier to the final ranking R (line 30) and removes it from the pool of remaining classifiers to rank.

To find the most promising classifier, we set a_{best} first to a random classifier left in A. We will compare it against all a_{comp} (competing) classifiers left in A (for-loop on line 5). On line 6 we retrieve a set D of datasets on which we have recorded performance results for both classifiers (note that d_{new} is not amongst those). Line 9 uses Equation 1 to retrieve the nearest dataset. Lines 12–15 show how Curve Adaptation shifts the retrieved learning curve to the partial learning curve, using Equation 2. Lines 16–18 show how Smaller Sample Size utilizes learning curves of a size close to the size of the new dataset. The classifier that performed best on the retrieved dataset (line 19) gets a vote, and the dataset is removed from the pool of available datasets. This is repeated k times, for the k nearest datasets. The classifier that has most votes is marked as a_{best}, and will be compared against the next competitor a_{comp} in the following loop iteration.

Because we arbitrarily select the order in which classifiers are considered, the ranking will not always be the same (the meta-algorithm is unstable). However, classifiers that perform consistently better on similar datasets will always be ranked above their inferior competitors. Furthermore, the meta-algorithm has a start up time, as it needs to build the partial learning curves. In Section 4 we will see that this is only a fraction of the run time of large datasets.

`Pairwise Curve Comparison` ranks classifiers based on their predictive accuracy on similar datasets, but instead of predictive accuracy any measure can be used for this selection. If we want to include run times in our experiments, we can use for example *A3R*, which aims to combine predictive accuracy and run time [1]. *A3R* compares the run times and accuracy of two classifiers on a dataset. However, in order to make it useful for methods that do not compare classifiers pairwise, and allow a fair comparison in our experiments, we define a slightly adapted version of the measure:

$$A3R'^{d_i}_{a_p} = \frac{SR^{d_i}_{a_p}}{\sqrt[r]{T^{d_i}_{a_p}}} \tag{3}$$

where $SR^{d_i}_{a_p}$ is the predictive accuracy (success rate) of classifier a_p on dataset d_i, $T^{d_i}_{a_p}$ is the run time of classifier a_p on dataset d_i and r is a parameter controlled by the user, influencing the importance of time. Indeed, setting this value lower results in a higher emphasize on time.

4 Experiments

We have evaluated the method using 53 classifiers and 39 data sets from OpenML[1]. The classifiers implementations come from Weka 3.7.12 [5], and include (but are not limited to) Decision Trees, Bayesian Networks, Support Vector Machines, Bagging, and Boosting. The data sets have between 540 and 48,842 observations, and between 5 and 241 attributes. All classifiers are run on all data sets.

Figure 1 shows how the run time of all classifiers increases as the sample size increases. The run time is the sum of the training time of all classifiers, averaged over all datasets. There seems to be a linear connection between the two. The drop can be explained by a subset of high-dimensional data sets, which take longer to train but contain only 2,000 observations.

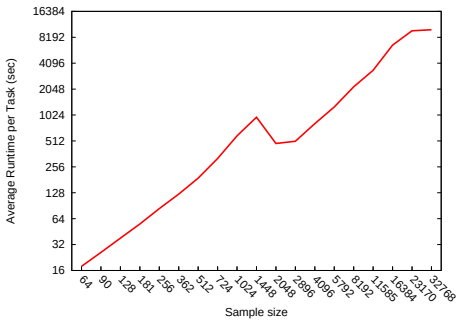


Fig. 1. Average run time of all classifiers per task per sample

We will use two strong baseline methods to compare our method to. `Best on Sample` runs all classifiers using a given sample size, and ranks the classifiers in the order of performance on that sample [10]. The `Average Rank` is a commonly used method in ranking and meta-learning [3, 9]. It ranks the classifiers in the order of their average rank on previously seen datasets, and has proven to be quite accurate. Although comparing the methods also against a meta-feature based approach seems interesting, configuring the latter takes much time, giving an unfair advantage to the sample based approaches.

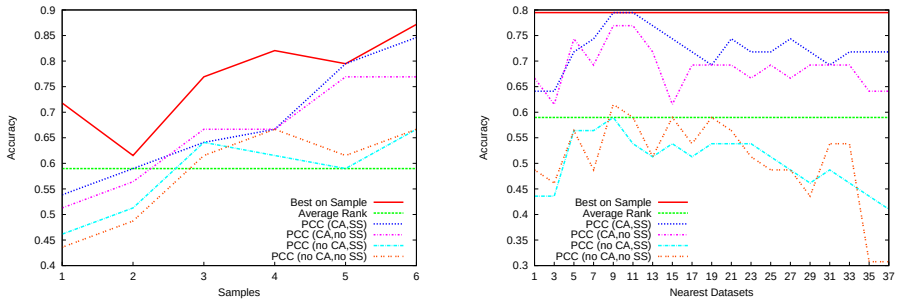
Section 4.1 and Section 4.2 describe traditional experiments focussing solely on accuracy; Section 4.3 describes our main contribution, novel experiments incorporating both accuracy and run times, yielding significant improvements.

4.1 Predicting the Best Classifier

In the first experiment we aim to establish how well the meta-algorithm performs when the task is to recommend the best available classifier. A recommendation is considered correct if there was no statistically significant difference between the absolute best classifier and the recommended classifier, as is done in, e.g., [8].

Our proposed method has several parameters. Most importantly, n (number of samples used) and k (number of nearest data sets to be identified). Furthermore, we seek to explore the effect of the proposed Curve Adaptation (CA) and smaller sample size (SS) by comparing instances having this option enabled against instances without.

[1] All classifiers, data sets and experiments used and generated in this study can be found online on: `http://www.openml.org/project/tag/LearningCurves/u/1`



(a) Varied value T, fixed $k = 9$ (b) Varied value k, fixed $T = 5$

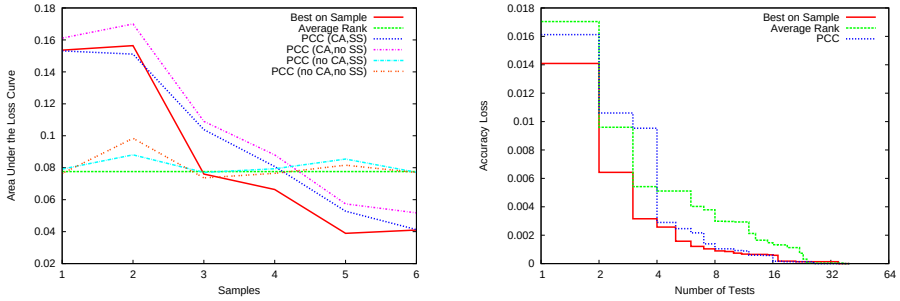
Fig. 2. Performance on predicting the best classifier.

Figure 2(a) shows the effect of varying the number of (increasingly large) samples when computing the partial learning curve. It can be seen that using more samples results almost consistently in better performance, as was expected. There are some drops in performance, which can probably be attributed to characteristics of the specific data sets used (e.g., dimensionality). Figure 2(b) shows the effect of varying the number of nearest neighbours, using odd numbers. `Average Rank` remains constant, as it does not use samples nor identify nearest datasets. Setting k around 9 seem very suitable in this case, but presumably this depends on the size of the meta-dataset. Setting this value too low might lead to instable behaviour, whereas setting it too high might result in including many data sets which are not similar enough.

Both figures show similar trends. `Best on Sample` dominates the other techniques in most of the cases, even though this method is rather simple. Furthermore, both `Pairwise Curve Comparison` instances using Curve Adaptation (CA) outperform the instances without Curve Adaptation. Smaller Sample Size (SS) also seems to improve the prediction quality, although the difference is less prominent. In all, both `Best on Sample` and `Pairwise Curve Comparison` obtain very reasonable performance, advising a (statistically) correct classifier in more than 85% of the cases.

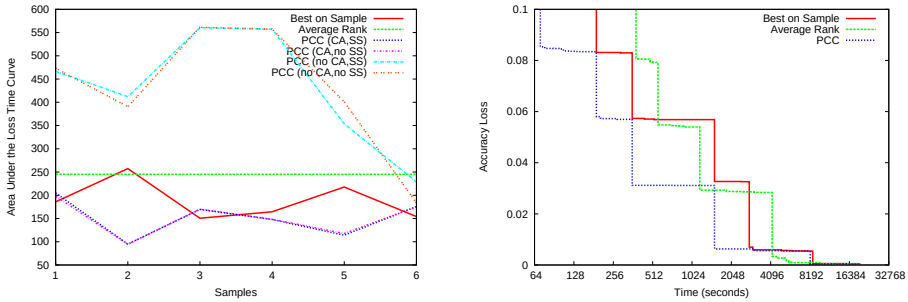
4.2 Ranking of Classifiers

In many meta-learning applications it is not enough to simply predict the single best classifier. Instead, a list of promising alternatives should be recommended so that the user can make an informed decision about which models to try, based on the available time and resources. One way to do this is to return a ranking. The standard approach to evaluate such a ranking is to compute the Spearman Correlation Coefficient [15]. However, it has a drawback: it penalizes every wrongly ranked classifier equally, whereas we typically do not care about incorrect ranked classifiers after the best one has been identified. An alternative



(a) Area under Loss Curve, $k = 9$ (b) Loss Curve, $T = 5$, $k = 9$

Fig. 3. Performance of ranking of classifiers in Loss space.

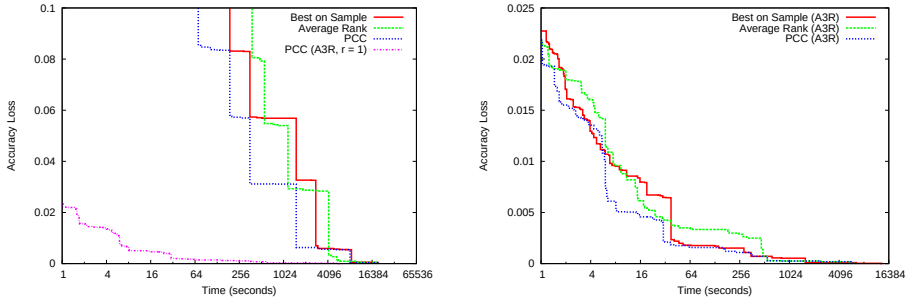


(a) Area under Loss Time Curve, $k = 9$ (b) Loss Time Curve, $T = 5$, $k = 9$

Fig. 4. Performance of ranking of classifiers in Loss Time space.

approach is to use loss curves (see, e.g., [9]). As shown in Figure 3(b), a loss curve tracks the accuracy loss incurred by selecting a suboptimal classifier, as we iteratively evaluate more classifiers on the full data set, for instance by going down a ranking, selecting the current best classifier in each iteration. Typically, this is repeated over many data sets and the average loss curve is reported. Similarly to ROC Curves for which commonly an Area Under the ROC Curve is calculated, we also can calculate the *Area Under the Loss Curve*, in which low values are preferred over high values. Although this measure is less informative than the Loss Curve itself, it can be used to show certain trends, e.g. the effect of an algorithm parameter.

Figure 3(a) plots the Area Under the Loss Curve against the number of samples. Using more (larger) samples typically results in an improved Area Under the Loss Curve score for `Best on Sample` and `Pairwise Curve Comparison` instances using Curve Adaptation. Again, `Average Rank` remains constant, and there seems to be no improvement for `Pairwise Curve Comparison` instances without Curve Adaptation. Figure 3(b) shows the Loss Curves. In order not



(a) Loss Time Curves, $T = 5$, $k = 9$ (b) Loss Time Curves, $T = 5$, $k = 9$, $A3R'$

Fig. 5. Classifier ranking performance in Loss Time space using the $A3R'$ criterion.

to overload the figure, we only include the baselines and the `Pairwise Curve Comparison` instance using both Curve Adaptation and Smaller Sample Size. The `Best on Sample` technique again dominates the other techniques.

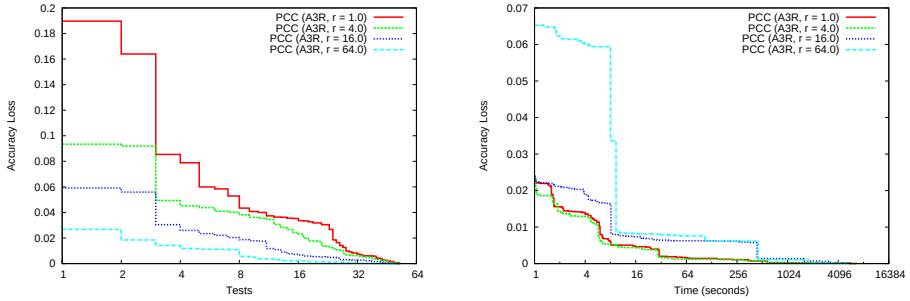
Loss Curves assume that every test will take the same amount of time, which is not realistic. For example, Multilayer Perceptrons take longer to train than Naive Bayes classifiers. Therefore, it is better to use *Loss Time Curves*, which plot the average loss against the time needed to obtain this loss, instead of the number of tests. They have been used before in the Optimization literature [6].

Figure 4 shows the results of the same experiment in Loss Time space. Figure 4(b) shows the Loss Time Curve, scaled to the part where the average loss is lower than 10%. Compared to Figure 3(b), it shows that while `Pairwise Curve Comparison` needs more tests to converge to an acceptable loss, it does so in less time. However, the results for other values of T (number of tests), shown in Figure 4(a), are less conclusive. Indeed, adding more samples does not lead to better results in Loss Time space. The reason for this is that none of the involved methods are taking training times into account when building a ranking.

4.3 Incorporating Run Times

Next, our aim is to establish that by involving run times we can improve the performance of the ranking in Loss Time space. Indeed, we need to trade off accuracy and speed. Naively ranking the classifiers in the order of run times yields bad results. Therefore, we adjust `Pairwise Curve Comparison` to compare and select classifiers based on their $A3R'$ scores, as introduced in Section 3. Both baseline methods are adjusted in a similar way.

Figure 5(a) compares the ranking obtained by `Pairwise Curve Comparison` using $A3R'$ against all methods building the ranking based on accuracy. The gain in performance is striking. `Pairwise Curve Comparison` using $A3R'$ converges to an acceptable loss level orders of magnitude faster than its counterparts without. When we also introduce our novel $A3R'$ selection criterion in the other



(a) Loss Curves, $T = 5$, $k = 9$ (b) Loss Time Curves, $T = 5$, $k = 9$

Fig. 6. Performance of `Pairwise Curve Comparison`, varying values for r.

methods, we obtain the results shown in Figure 5(b). Indeed, the $A3R'$ criterion is useful in these methods as well, all reducing the accuracy loss drastically faster than before. However, the `Pairwise Curve Comparison` method still dominates the other techniques, though the differences are smaller.

Finally, $A3R'$ has a parameter r that lets users trade off the importance of run times. Increasing the value of r decreases the importance of run times when selecting classifiers. Figure 6(a) shows that the methods emphasizing accuracy converge to a low loss in few tests, since they focus on classifiers that are probably good, but potentially slow. However, Figure 6(b) shows that they do not converge faster in Loss Time space. Evaluating faster methods earlier clearly pays of, especially if there is limited time to select a classifier.

5 Conclusion

This paper addresses the problem of algorithm selection under a budget, where multiple algorithms can be run on the full data set until the budget expires.

We have extended the method presented in [8] such that it generates a ranking of classifiers, rather than just predicting the single best classifier, and evaluated it using a much larger amount of classifiers. Interestingly, a simple and elegant baseline method called `Best on Sample` outperformed this method in our experiments, selecting good classifiers in fewer tests. However, when tested in a more realistic setting where the budget is time, rather than a number of tests, and using a novel selection criterion, $A3R'$, that trades off accuracy and run times, the newly proposed method outperformed all baselines. This suggests that it is very suitable for algorithm selection applications with a limited time budget.

Another contribution of this work is the use of Loss Time Curves to study meta-learning algorithms, which to the best of our knowledge, have not been previously used in the meta-learning literature. Future work will focus on adapting other meta-learning techniques with the $A3R'$ criterion and/or evaluating them in Loss Time space, as this might lead to even more valuable insight.

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. Brazdil, P., Gama, J., Henery, B.: Characterizing the Applicability of Classification Algorithms Using Meta-Level Learning. In: Bergadano, F., De Raedt, L. (eds.) Machine Learning: ECML-94, Lecture Notes in Computer Science, vol. 784, pp. 83–102. Springer (1994)
3. Brazdil, P., Soares, C.: A Comparison of Ranking Methods for Classification Algorithm Selection. In: Machine Learning: ECML 2000, pp. 63–75. Springer (2000)
4. Fürnkranz, J., Petrak, J.: 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 (2001)
5. 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)
6. Hutter, F., Hoos, H.H., Leyton-Brown, K., Murphy, K.: Time-Bounded Sequential Parameter Optimization. In: Learning and intelligent optimization, pp. 281–298. Springer (2010)
7. Leite, R., Brazdil, P.: Predicting Relative Performance of Classifiers from Samples. In: Proceedings of the 22nd international conference on Machine learning. pp. 497–503. ACM (2005)
8. Leite, R., Brazdil, P.: Active Testing Strategy to Predict the Best Classification Algorithm via Sampling and Metalearning. In: ECAI. pp. 309–314 (2010)
9. 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)
10. Petrak, J.: Fast Subsampling Performance Estimates for Classification Algorithm Selection. In: Proceedings of the ECML-00 Workshop on Meta-Learning: Building Automatic Advice Strategies for Model Selection and Method Combination. pp. 3–14 (2000)
11. 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)
12. Provost, F., Jensen, D., Oates, T.: Efficient Progressive Sampling. In: Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 23–32. ACM (1999)
13. Rice, J.R.: The Algorithm Selection Problem. Advances in Computers 15, 65118 (1976)
14. Smith-Miles, K.A.: Cross-disciplinary Perspectives on Meta-Learning for Algorithm Selection. ACM Computing Surveys (CSUR) 41(1), 6 (2008)
15. Sun, Q., Pfahringer, B.: Pairwise meta-rules for better meta-learning-based algorithm ranking. Machine learning 93(1), 141–161 (2013)
16. Vanschoren, J., Blockeel, H., Pfahringer, B., Holmes, G.: Experiment databases. Machine Learning 87(2), 127–158 (2012)
17. Vanschoren, J., van Rijn, J.N., Bischl, B., Torgo, L.: OpenML: networked science in machine learning. ACM SIGKDD Explorations Newsletter 15(2), 49–60 (2014)
18. Vilalta, R., Drissi, Y.: A Perspective View and Survey of Meta-Learning. Artificial Intelligence Review 18(2), 77–95 (2002)
19. Wolpert, D.H.: Stacked generalization. Neural networks 5(2), 241–259 (1992)