# 6 Combining Accuracy and Run Time

Meta-learning focuses on finding classifiers and parameter settings that work well on a given dataset. Evaluating all possible combinations typically takes too much time, hence many solutions have been proposed that attempt to predict which classifiers are most promising to try. As discussed in Chapter 3, the first recommended classifier is not always the correct choice, so multiple recommendations should be made, making this a ranking problem rather than a classification problem.

Even though this is a well studied problem, in the meta-learning literature there is no common way of evaluating these. We advocate the use of Loss Time Curves, as used in the field of optimization. These visualize the amount of budget (time) needed to converge to an acceptable solution. We investigate two methods that utilize the measured performances of classifiers on small samples of data to make such recommendation, and adapt it so that these works well in Loss Time space. Experimental results show that this method converges extremely fast to an acceptable solution.

OpenML was used as an experiment repository. The datasets and tasks that were used take many resources (time and memory) to model, so this work was greatly accelerated by including prior results that were collaboratively generated.

## 6.1 Introduction

When presented with a new classification problem, a key challenge is to identify a classifier and parameter settings that obtain good predictive performance. This problem is known as the *Algorithm Selection Problem* [119]. 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 and suboptimal, for the reasons mentioned in Chapter 3. The field of *meta-learning* attempts to solve this by learning from prior examples. Typically, a set of classifiers is recommended based on the performance on similar datasets.

The meta-learning method SAM [87] identifies similar datasets based on the *learning curves* of classifiers trained on them, and recommends the classifier that performs best on these similar datasets. A learning curve is an ordered set of performance scores of a classifier on data samples of increasing size [113]. 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, making it unable to distinguish between fast and slow classifiers. 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.

Another popular meta-learning technique is *Active Testing* [88]. It recommends a ranking of classifiers, advising in which order these should be cross-validated to check their applicability to the inspected dataset. This ranking is dynamically updated, based on results from earlier performed cross-validation tests on the dataset at hand. This method also does not take the training time of the models into account, making it unable to distinguish between fast and slow classifiers.

This chapter covers the following contributions. We extend the aforementioned techniques so that these produce a ranking of classifiers and takes into account the run times of classifiers. For this, a new evaluation measure is explored, $A3R'$, capable of trading of accuracy and run time. Furthermore, we study the performance of this method in Loss space and Loss Time space. We will argue that Loss Curves as presented in [88] are biased, and propose the use of Loss Time Curves, as commonly used in Optimization literature (e.g., [74]). 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 [52]. Our proposed technique dominates the baseline methods in some scenarios. Moreover, our results suggest that a simple, sample-based baseline technique has been mistakenly neglected in the literature. Finally, we will see that meta-learning techniques that adopt $A3R'$ improve their performance in Loss Time space.

This chapter is organized as follows. Chapter 6.2 surveys related work. Chapter 6.3 formalizes both methods, and adapt them to work in Loss Time space. Chapter 6.4 contains experiments. Chapter 6.5 concludes.

## 6.2 Related Work

Meta-learning aims to learn which learning techniques work well on what data [141]. A common task, known as the Algorithm Selection Problem [119], 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* [20], 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 (e.g., number of examples, number of attributes), statistical (e.g., mean standard deviation of attributes, mean skewness of attributes), information theoretic (e.g., class entropy, mean mutual information) or landmarkers [108] (performance evaluations of simple classifiers). Many meta-learning studies follow this approach [125, 133, 144, 158, 163].

As argued in Chapter 3, meta-feature-based approaches have some intrinsic limitations. First, it is hard to construct a meta-feature set that adequately characterizes the problem space [86]. Second, the most successful meta-features, landmarkers, can be computationally expensive, limiting the options [108]. Finally, because not all classifiers can model 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 [86, 87] identify similar datasets based on partial learning curves. In this particular method, a *partial learning curve* is computed, using small samples, to identify similar datasets and use performance information from those datasets to extrapolate the learning curve. As such, running classifiers on these samples is rather cheap. There is also a clear connection with multi-armed bandit strategies [89], where results on a small sample determine whether it is worthwhile to continue learning.

Alternatively, the Best on Sample method uses the performance estimates of classifiers on a small subset (sample) of the data, and recommends the classifiers which perform best on this sample, in descending order [107]. The smaller this sample is, the fewer time this method takes to execute. Prior work is inconclusive about its performance. The authors of [107] suggest that this technique should be used as a baseline method in meta-learning research. The authors of [52] show that this information is not useful as a 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 [86]. However, this happens less often as the sample size increases, making this method quite reliable when using the right sample size, as we will see in Chapter 6.4.

These three methods all aim to recommend an algorithm with high accuracy. However, in a setting where multiple classifiers will be tried sequentially and the budget is time, it might make sense to first try many fast algorithms, rather than a few slow ones. This can be done by selecting classifiers based on a trade-off between accuracy and run time. Brazdil et al. [21] proposes *adjusted ratio of ratios* (ARR), which is defined as:

$$ARR^{d_i}_{a_p,a_q} = \frac{\dfrac{SR^{d_i}_{a_p}}{SR^{d_i}_{a_q}}}{1 + AccD \cdot \log_2\left(\dfrac{T^{d_i}_{a_p}}{T^{d_i}_{a_q}}\right)} \tag{6.1}$$

where $SR^{d_i}_{a_p}$ and $SR^{d_i}_{a_q}$ are the predictive accuracy (success rate) of classifiers $a_p$ and $a_q$ (respectively) on dataset $d_i$. Likewise, $T^{d_i}_{a_p}$ and $T^{d_i}_{a_q}$ are the run times of classifiers $a_p$ and $a_q$ (respectively) on dataset $d_i$. Finally, $AccD$ is a parameter controlled by the user, influencing the relative importance of accuracy to run time.

As was pointed out by [1], there are some problems with this measure: it is not monotonic, and even approaches infinity at some point. Therefore, the measure $A3R$ was introduced:

$$A3R^{d_i}_{a_p,a_q} = \frac{\dfrac{SR^{d_i}_{a_p}}{SR^{d_i}_{a_q}}}{\sqrt[r]{T^{d_i}_{a_p}/T^{d_i}_{a_q}}} \tag{6.2}$$

where, similar to $ARR$, $SR^{d_i}_{a_p}$ and $SR^{d_i}_{a_q}$ are the predictive accuracy (success rate) of classifiers $a_p$ and $a_q$ (respectively) on dataset $d_i$. Likewise, $T^{d_i}_{a_p}$ and $T^{d_i}_{a_q}$ are the run times of classifiers $a_p$ and $a_q$ (respectively) on dataset $d_i$. Finally, $r$ is a parameter controlled by the user, influencing the relative importance of accuracy versus run time.

Although both $ARR$ and $A3R$ are suitable for finding fast classifiers, these have not been used as such before. Experimental evaluations have focused on recommending classifiers that work well on this criterion. This seems a bit arbitrary. Indeed, we are still interested in finding the algorithm with the highest accuracy, however we want to find it as fast as possible (i.e., with fewest number of cross-validation test or time). In this respect our approach differs from earlier meta-learning approaches by using this measure to build a ranking of classifiers that finds a reasonable classifier as fast as possible.

## 6.3 Methods

In this chapter we describe two methods that extend the work of [86, 87, 88] in several ways. We consider a set $A$ of classifiers, $a_m$ ($m = 1, 2, 3, \ldots, M$). We also consider a set $D$ of datasets, $d_n$ ($n = 1, 2, 3, \ldots, N$), on which we have information on the performance of the classifiers in $A$ ($d_{new}$ is not in $D$). The total amount of trainings instances available for dataset $d_n$ is denoted as $|d_n|$. Let $P_{m,n,s}$ and $P'_{m,n,s}$ denote the performance of classifier $a_m$ on dataset $d_n$, for a given evaluation measure (e.g., predictive accuracy), using a sample size of $s$. Furthermore, $\Omega$ denotes the size of a dataset given by the context. Hence, $P_{m,n,\Omega}$ (equals $P_{m,n,|d_n|}$) denotes the performance of classifier $a_m$ on the full dataset $d_n$.

### 6.3.1 Pairwise Curve Comparison

Various methods that are successful at recommending algorithms make use of so-called learning curves. A learning curve is an ordered set of performance scores of a classifier on data samples of increasing size [113]. The method proposed by [86] builds a partial learning curve for a pair of algorithms, and finds among earlier seen datasets (on which these algorithms were also run) the one that has the most similar partial learning curves. Based on the performance of the algorithms on that full dataset, it makes a recommendation. This idea is extended to multiple classifiers (rather than a pair) in [87].

In this chapter, we propose a novel method that extends the method as defined by [87] in three ways. First, it recommends a ranking of classifiers, rather than just a single best classifier. Second, it can take arbitrary evaluation measures into account, such as run time. Lastly, we introduce an optimization called *Smaller Sample*, that improves performance when the sizes of datasets differ a lot.

Let $S$ be the set of samples of dataset $d$ of increasing size $s_t = 2^{5.5+0.5 \cdot t}$ with $t = (1, 2, 3, \ldots, T)$, and $T$ being a parameter set by the user such that $1 \leq T \leq 2 \cdot (\lfloor \log_2 |d_n| \rfloor - 5.5)$. This ensures that the biggest data sample never exceeds the available amount of training data. The samples follow a geometric increase, as suggested by [113]. When using a higher value for $T$, larger samples are calculated, presumably yielding more accurate estimates at the expense of higher run times.

Figure 6.1 shows learning curves of all model types introduced in Chapter 2.4. The $x$-axis is displayed on a logarithmic scale, to show the geometrical increase in sample sizes. It shows some typical learning curve behaviour. First, when presented with more data, the classifiers typically perform better. However, this is not always the case. Sometimes a classifier handles a new batch of data not so well, and accuracy decreases. Furthermore, there is also a trend of diminishing increases. At the begin-

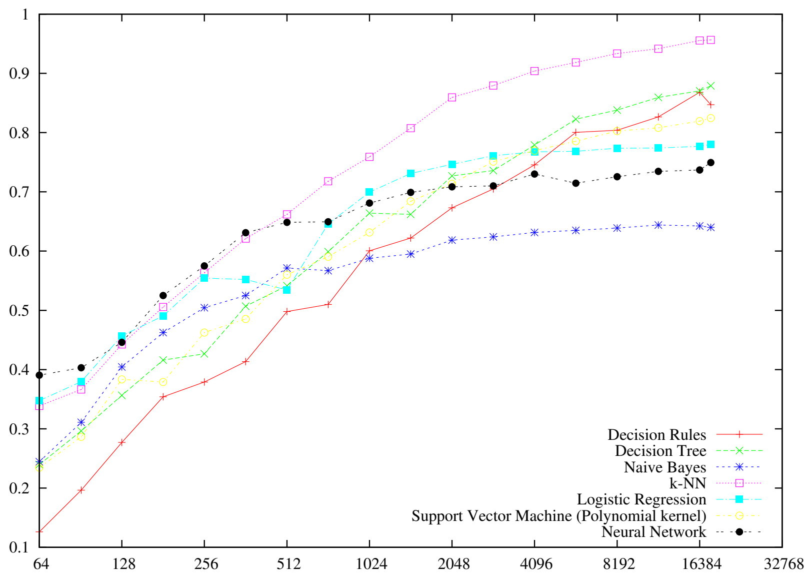


Figure 6.1: Learning curves on the 'letter' dataset.

ning the accuracy gain of adding more data is considerable, whereas at some point it flattens out. This is especially visible for models like Naive Bayes and Logistic Regression. Lastly, as already noted by [86], learning curves can cross. This also happens quite often for small samples, but less often for bigger samples.

The distance between two datasets $d_i$ and $d_j$ can be determined using the following function [86]:

$$dist(d_i, d_j, a_p, a_q, T) = \sum_{t=1}^{T}(P_{p,i,s_t} - P_{p,j,s_t})^2 + \sum_{t=1}^{T}(P_{q,i,s_t} - P_{q,j,s_t})^2 \qquad (6.3)$$

This distance function is related to the Euclidean distance. It gives a measure of how similar two datasets are, based on the learning curves of the two classifiers. Other work proposes a distance function that measures the Manhattan distance between learning curves, but experiments show that the difference in performance between these variants is negligible [87].

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 likely perform better on the new dataset. Controversially, it has been remarked that as the number of used samples increases, the performance of this technique decreases [86]. The authors of [86] 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. This is done because some datasets are simply harder to model, hence the whole curve will be higher or lower, and that is being corrected for. 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:

$$f(d_i, d_r, a_p, T) = \frac{\sum_{t=1}^{T}(P_{p,i,s_t} \cdot P_{p,r,s_t} \cdot s_t^2)}{\sum_{t=1}^{T}((P_{p,r,s_t})^2 \cdot s_t^2)} \tag{6.4}$$

The resulting coefficient $f$ can be used to scale the performance of the retrieved learning curve of dataset $d_r$ to match the partial learning curve on dataset $d_i$, making the final points of $d_r$ (that are not available in $d_i$) more realistic.

A novel optimization that could potentially improve performance is the *Smaller Sample* technique. As not all datasets are of the same size, it is possible that a retrieved dataset has a bigger size than the new dataset, which might give an unfair advantage to particular learners. Suppose that the retrieved dataset contains a high number of observations, and from a certain sample size on the learning curve of one algorithm outperforms all other algorithms. If the new dataset is much smaller than the retrieved dataset, this information might be irrelevant and potentially obfuscates the prediction for the new dataset. In that case it might be beneficial to use the performance of the classifiers at a sample size close to the full size of the new dataset. More formally, when reasoning over the performance of a given classifier $a$ on dataset $d_{new}$ based on a retrieved dataset $d_r$, if $|d_{new}| < |d_r|$, it might be more informative to use the performance of algorithm $a$ on a subset of $d_r$, rather than the full dataset $d_r$.

Algorithm 6.1 shows the full 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 available to build the partial learning curve), and boolean parameters indicating whether to use the Curve Adaptation and Smaller Sample technique. 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 an arbitrary 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 (recall that $d_{new}$ is not amongst those). Line 9 uses Equation 6.3 to retrieve the nearest dataset. Lines 12–15 show how Curve Adaptation shifts the retrieved learning curve to the partial learning curve, using Equation 6.4. Lines 16–18 show how the Smaller Sample option utilizes learning curves of a size close to the size of the new dataset. The classifier that performed best on the retrieved

**Algorithm 6.1** Pairwise Curve Comparison (PCC)

```
Require: $d_{new}$, $k \in \mathbb{N}^{+}$, $T \in \mathbb{N}^{+}$, $CurveAdaptation \in \{0,1\}$, $SmallerSample \in \{0,1\}$
 1: Initialize $A$ as a set of all classifiers
 2: Initialize $R$ as empty list
 3: while $|A| > 0$ do
 4:     $a_{best} \leftarrow$ Arbitrary element from $A$
 5:     for all $a_{comp} \in A : a_{comp} \neq a_{best}$ do
 6:         Initialize $D$ as the set of all datasets on which $a_{best}$ and $a_{comp}$ were ran
 7:         $votesBest = votesComp = 0$
 8:         while $votesBest + votesComp < k$ do
 9:             $d_{sim} \leftarrow \arg\min_{d_i \in D} dist(d_{new}, d_i, a_{best}, a_{comp}, T)$
10:             $coeff_{best} = coeff_{comp} = 1$
11:             $samp \leftarrow \Omega$
12:             if $CurveAdaptation = 1$ then
13:                 $coeff_{best} \leftarrow f(d_{new}, d_{sim}, a_{best}, T)$
14:                 $coeff_{comp} \leftarrow f(d_{new}, d_{sim}, a_{comp}, T)$
15:             end if
16:             if $SmallerSample = 1$ and $|d_{new}| < |d_{sim}|$ then
17:                 $samp \leftarrow |d_{new}|$
18:             end if
19:             if $coeff_{best} \cdot P'_{best,d_{sim},samp} > coeff_{comp} \cdot P'_{comp,d_{sim},samp}$ then
20:                 $votesBest \leftarrow votesBest + 1$
21:             else
22:                 $votesComp \leftarrow votesComp + 1$
23:             end if
24:             $D \leftarrow D - d_{sim}$
25:         end while
26:         if $votesBest < votesComp$ then
27:             $a_{best} \leftarrow a_{comp}$
28:         end if
29:     end for
30:     $R \leftarrow R + a_{best}$
31:     $A \leftarrow A - a_{best}$
32: end while
33: return $R$ {Ranking of classifiers in decreasing order}
```

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. Note that the algorithm potentially utilizes two different evaluation scores, denoted by $P$ and $P'$, but these can also be the same. The scores of one evaluation measure are used for identifying similar datasets and Curve Adaptation (i.e., the one denoted by $P$); the scores of the other evaluation measure are used for selecting an appropriate classifier (i.e., the one denoted by $P'$). This is useful because not all evaluation measures are suitable for both tasks. For example, measures that trade-off accuracy and run time (e.g., $ARR$, $A3R$) are very suitable for selecting appropriate algorithms, however learning curves that are built upon these measures are typically neither informative nor stable.

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, it assumes that 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. However, this is also the case for conventional meta-learning techniques (e.g., when using landmarkers). Therefore we will not consider these additional costs in the results.

### 6.3.2 Active Testing

Active Testing was introduced in [88]. It is an algorithm selection method that combines grid search with meta-knowledge. It recommends a ranking of classifiers (the order in which they should be cross-validated), but it also updates the ranking of the not yet cross-validated classifiers after every test.

Active Testing iteratively chooses new promising algorithms; those that have a good possibility to outperform a current best algorithm. For each candidate algorithm, it finds the historic datasets on which the candidate algorithm outperforms the current best. This is done by the notion of relative landmarkers, which are defined as:

$$RL(a_t, a_{best}, d_i, P) = b(P_{a_t,d_i,\Omega} > P_{a_{best},d_i,\Omega}) \cdot (P_{a_t,d_i,\Omega} - P_{a_{best},d_i,\Omega}) \tag{6.5}$$

where $P_{a_k,d_i,\Omega}$ is the evaluation measure (obtained by cross-validation) of algorithm $a_k$ ($a_k \in \{best, t\}$) on dataset $d_i$, and $b$ is an indicator function returning $1$ if the condition is true and $0$ otherwise. Active Testing operates on the full dataset rather than learning curves or sub-samples; the $\Omega$ subscript is maintained in the notation for consistency. Basically, the relative landmakers measure the accuracy difference between algorithms on datasets where the current best algorithm was outperformed,

**Algorithm 6.2** Active Testing (AT)

```
Require: $d_{new}$, $a_{best} \in A$, $P$
 1: Initialize $A$ as a set of all classifiers except $a_{best}$
 2: Initialize $R$ as list containing $a_{best}$
 3: $P'_{best,new,\Omega} = CV(a_{best}, d_{new})$
 4: while $|A| > 0$ do
 5:     $a_{comp} = \arg\max_{a_i \in A} \sum_{d_i \in D} RL(a_t, a_{best}, d_i, P) \cdot Sim(d_{new}, d_i)$
 6:     $A \leftarrow A - a_{comp}$
 7:     $P'_{comp,new,\Omega} = CV(a_{comp}, d_{new})$
 8:     if $P'_{comp,new,\Omega} > P'_{best,new,\Omega}$ then
 9:         $a_{best} = a_{comp}$
10:         $P'_{best,new,\Omega} = P'_{comp,new,\Omega}$
11:     end if
12:     $R \leftarrow R + a_{comp}$
13: end while
14: return $a_{best}$ {Best classifier based on measure $P'$}
```

and neglect the accuracy difference on datasets where the current best algorithm was better. For each new cross-validation test, we are interested in the method that obtained the highest relative landmarker score on datasets similar to the current one. As such, we are optimizing:

$$a_{comp} = \underset{a_t \in A}{\arg\max} \sum_{d_i \in D} RL(a_t, a_{best}, d_i, P) \cdot Sim(d_{new}, d_i) \tag{6.6}$$

where $Sim(d_{new}, d_i)$ is a measure of similarity between dataset $d_{new}$ and dataset $d_i$. Several similarity measures are proposed in [88]. The method *Active Testing 0* naively assumes that all datasets are equally similar, hence its corresponding similarity function always returns 1. The method *Active Testing 1* determines the similarity based on only the last cross-validation test. If the last cross-validation test shows that $a_{comp}$ is better than the best algorithm until that moment $a_{best}$ (i.e., $P_{a_{comp},d_{new},\Omega} > P_{a_{best},d_{new},\Omega}$), then each dataset $d_i \in D$ that satisfies $P_{a_{comp},d_i,\Omega} > P_{a_{best},d_i,\Omega}$ is considered similar to dataset $d_{new}$.

Algorithm 6.2 puts this all together. It requires the user to choose a measure $P$, the evaluation measure that should be measured by the cross-validation test. Leite et al. [88] considered only predictive accuracy, but it is clear that any measure can be used for this, e.g., $A3R$. Note that although we are using the measure determined by $P$ to determine the order in which we search, we ultimately evaluate all algorithms based on a measure $P'$, which is typically predictive accuracy or area under the ROC curve. The two are not necessarily identical.

Furthermore, this method requires an arbitrary classifier to be selected first, $a_{best}$. In the work of [88], the top ranked algorithm from the global ranking was used. This algorithm will be cross-validated first (line 3). Lines 4–13 show a while-loop, which at each iteration removes an algorithm from $A$ (line 6), cross-validates it (line 7) and adds it to $R$ (line 12). Which algorithm that is, is determined using Eq. 6.6 on line 5. If that algorithm performed better than the best algorithm so far, it is considered the new best algorithm (line 9–10). Finally, on line 14 the best algorithm found is returned. Note that we do not need to return a ranking, as the algorithms are already evaluated using a cross-validation test. We are guaranteed that the returned algorithm is the best on the determined criterion.

### 6.3.3 Combining Accuracy and Run Time

Both Active Testing and Pairwise Curve Comparison select classifiers based on their predictive accuracy on similar datasets. Both are designed such that instead of predictive accuracy any measure can be used for this selection. Because our experiments focus on both accuracy and run time, we will experiment with *A3R*, which combines predictive accuracy and run time [1]. *A3R* compares the run times and accuracy of two classifiers on a dataset, so it could be used directly into methods that work based on pairwise comparisons. However, in order to make it useful for methods that do not compare classifiers pairwise, and allow a fair comparison in the experiments, we define a slightly adapted version of the measure:

$$A3R'^{d_i}_{a_p} = \frac{SR^{d_i}_{a_p}}{\sqrt[r]{T^{d_i}_{a_p}}} \tag{6.7}$$

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 $d_i$; finally, $r$ is a parameter controlled by the user, influencing the importance of time. Indeed, a lower value results in a higher emphasize on time. The higher the $A3R'$ score, the more suitable the classifier is on the combination of accuracy and run time.

Note that both Pairwise Curve Comparison and Active Testing can natively work with *A3R*, as they work based on pairwise comparisons. However, as this does not hold for some baseline methods, it is good to have the $A3R'$ as an alternative. It is less complex and when comparing two classifiers it ranks these the same.

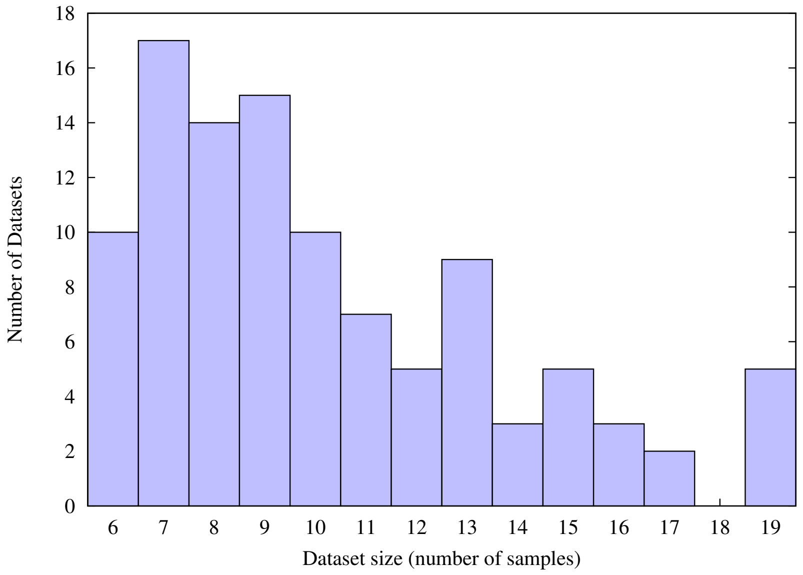


Figure 6.2: Number of datasets with maximum number of learning curve samples.

## 6.4 Experiments

To evaluate the proposed algorithm selection strategies, we used 30 classifiers and 105 datasets from OpenML [154]. Table 6.1 shows all datasets used in this experiment. The datasets have between 500 and 48,842 observations, and between 5 and 10,937 attributes.The size of the dataset is of importance to Pairwise Curve Comparison. In order to construct a learning curve of a given number of such samples, the dataset has to be sufficiently large (i.e., contain enough observations). In fact, to be able to construct a learning curve of 7 samples, the training set of the dataset has to contain at least 512 observations. Figure 6.2 shows the maximum number of samples that a learning curve can contain per dataset. Note that because we use 10-fold cross-validation in the experiments, a dataset actually needs 563 observations in order to guarantee a trainings set of 512 observations.

The algorithms are all from Weka 3.7.13 [61], and include all model types from Chapter 2 and all ensemble types from Chapter 3. The same algorithms are used as those used in Figure 4.10 (page 61) and Figure 4.11 (page 62). All classifiers were run on all datasets.

We will use two strong baseline methods to compare our method to. The *Best on Sample* method runs all classifiers using a given sample size, and ranks the classifiers

Table 6.1: Datasets used for the experiment.

| name | obs. | atts. | cls. |
|---|---:|---:|---:|
| irish | 500 | 6 | 2 |
| autoUniv-au7-500 | 500 | 13 | 5 |
| collins | 500 | 24 | 15 |
| kc2 | 522 | 22 | 2 |
| climate-model-simulation | 540 | 21 | 2 |
| cylinder-bands | 540 | 40 | 2 |
| AP_Breast_Ovary | 542 | 10,937 | 2 |
| AP_Colon_Kidney | 546 | 10,937 | 2 |
| monks-problems-3 | 554 | 7 | 2 |
| monks-problems-1 | 556 | 7 | 2 |
| ilpd | 583 | 11 | 2 |
| synthetic_control | 600 | 62 | 6 |
| monks-problems-2 | 601 | 7 | 2 |
| AP_Breast_Kidney | 604 | 10,937 | 2 |
| balance-scale | 625 | 5 | 3 |
| AP_Breast_Colon | 630 | 10,937 | 2 |
| profb | 672 | 10 | 2 |
| soybean | 683 | 36 | 19 |
| Australian | 690 | 15 | 2 |
| credit-a | 690 | 16 | 2 |
| breast-w | 699 | 10 | 2 |
| autoUniv-au7-700 | 700 | 13 | 3 |
| eucalyptus | 736 | 20 | 5 |
| blood-transfusion-serv. | 748 | 5 | 2 |
| autoUniv-au6-750 | 750 | 41 | 8 |
| diabetes | 768 | 9 | 2 |
| analcatdata_dmft | 797 | 5 | 6 |
| analcatdata_authorship | 841 | 71 | 4 |
| vehicle | 846 | 19 | 4 |
| anneal | 898 | 39 | 6 |
| oh15.wc | 913 | 3,101 | 10 |
| oh5.wc | 918 | 3,013 | 10 |
| vowel | 990 | 13 | 11 |
| credit-g | 1,000 | 21 | 2 |
| autoUniv-au1-1000 | 1,000 | 21 | 2 |
| autoUniv-au6-1000 | 1,000 | 41 | 8 |
| oh0.wc | 1,003 | 3,183 | 10 |
| qsar-biodeg | 1,055 | 42 | 2 |
| MiceProtein | 1,080 | 82 | 8 |
| autoUniv-au7-1100 | 1,100 | 13 | 5 |
| pc1 | 1,109 | 22 | 2 |
| banknote-authentication | 1,372 | 5 | 2 |
| pc4 | 1,458 | 38 | 2 |
| cmc | 1,473 | 10 | 3 |
| OVA_Breast | 1,545 | 10,937 | 2 |
| OVA_Colon | 1,545 | 10,937 | 2 |
| OVA_Kidney | 1,545 | 10,937 | 2 |
| OVA_Lung | 1,545 | 10,937 | 2 |
| OVA_Omentum | 1,545 | 10,937 | 2 |
| OVA_Ovary | 1,545 | 10,937 | 2 |
| OVA_Uterus | 1,545 | 10,937 | 2 |
| pc3 | 1,563 | 38 | 2 |
| semeion | 1,593 | 257 | 10 |
| 100-plants-texture | 1,599 | 65 | 100 |
| 100-plants-margin | 1,600 | 65 | 100 |
| 100-plants-shape | 1,600 | 65 | 100 |
| car | 1,728 | 7 | 4 |
| steel-plates-fault | 1,941 | 34 | 2 |
| mfeat-morphological | 2,000 | 7 | 10 |
| mfeat-zernike | 2,000 | 48 | 10 |
| mfeat-karhunen | 2,000 | 65 | 10 |
| mfeat-fourier | 2,000 | 77 | 10 |
| mfeat-factors | 2,000 | 217 | 10 |
| mfeat-pixel | 2,000 | 241 | 10 |
| kc1 | 2,109 | 22 | 2 |
| cardiotocography | 2,126 | 36 | 10 |
| segment | 2,310 | 20 | 7 |
| scene | 2,407 | 300 | 2 |
| autoUniv-au4-2500 | 2,500 | 101 | 3 |
| ozone-level-8hr | 2,534 | 73 | 2 |
| cjs | 2,796 | 35 | 6 |
| splice | 3,190 | 62 | 3 |
| kr-vs-kp | 3,196 | 37 | 2 |
| Internet-Advertisements | 3,279 | 1,559 | 2 |
| gina_agnostic | 3,468 | 971 | 2 |
| Bioresponse | 3,751 | 1,777 | 2 |
| sick | 3,772 | 30 | 2 |
| abalone | 4,177 | 9 | 29 |
| ada_agnostic | 4,562 | 49 | 2 |
| spambase | 4,601 | 58 | 2 |
| wilt | 4,839 | 6 | 2 |
| waveform-5000 | 5,000 | 41 | 3 |
| phoneme | 5,404 | 6 | 2 |
| wall-robot-navigation | 5,456 | 25 | 4 |
| optdigits | 5,620 | 65 | 10 |
| first-order-theorem-proving | 6,118 | 52 | 6 |
| satimage | 6,430 | 37 | 6 |
| musk | 6,598 | 170 | 2 |
| isolet | 7,797 | 618 | 26 |
| mushroom | 8,124 | 23 | 2 |
| Gest.Ph. Segmentation Proc. | 9,873 | 33 | 5 |
| JapaneseVowels | 9,961 | 15 | 9 |
| jm1 | 10,885 | 22 | 2 |
| pendigits | 10,992 | 17 | 10 |
| PhishingWebsites | 11,055 | 31 | 2 |
| sylva_agnostic | 14,395 | 217 | 2 |
| eeg-eye-state | 14,980 | 15 | 2 |
| mozilla4 | 15,545 | 6 | 2 |
| MagicTelescope | 19,020 | 12 | 2 |
| letter | 20,000 | 17 | 26 |
| webdata_wXa | 36,974 | 124 | 2 |
| Click_prediction_small | 39,948 | 12 | 2 |
| electricity | 45,312 | 9 | 2 |
| tamilnadu-electricity | 45,781 | 4 | 20 |
| adult | 48,842 | 15 | 2 |

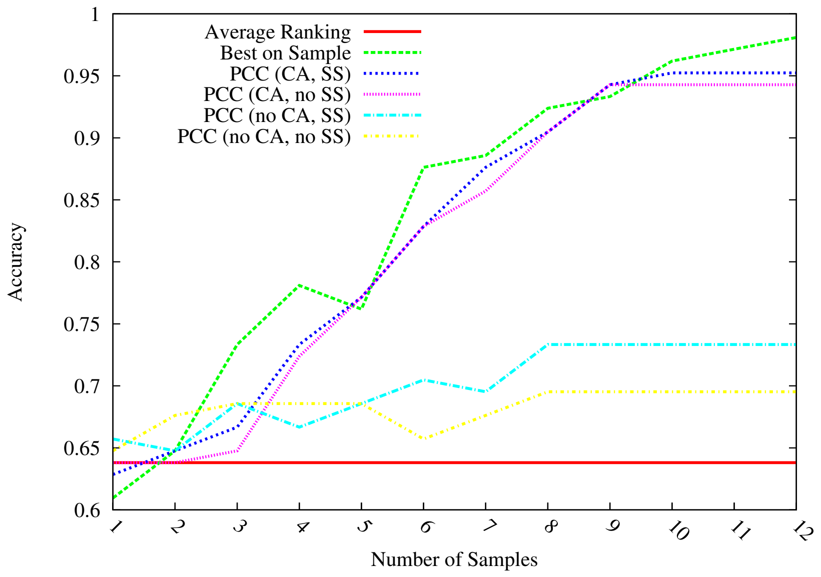
in the order of performance on that sample [107]. The *Average Ranking* method ranks the classifiers in the order of their average rank on previously seen datasets [19, 88]. The Average Ranking method represents the typical approach that human experts in machine learning wield. For each dataset, a set of favourite algorithms is selected and tried in a static order. Both baseline methods have proven to be quite accurate in previous studies.

Chapter 6.4.1 describes an experiment that focuses solely on predicting the best classifier; here we attempt to reproduce the results obtained by [87] using a larger number of datasets and classifiers. In Chapter 6.4.2 we show how the meta-algorithms perform when predicting a ranking of classifiers in Loss space. Chapter 6.4.3 shows how the meta-algorithms perform in Loss Time space. Chapter 6.4.4 describes our main contribution, empirically evaluated on both accuracy and run times. This method yields significant improvements while trading of accuracy and run time.

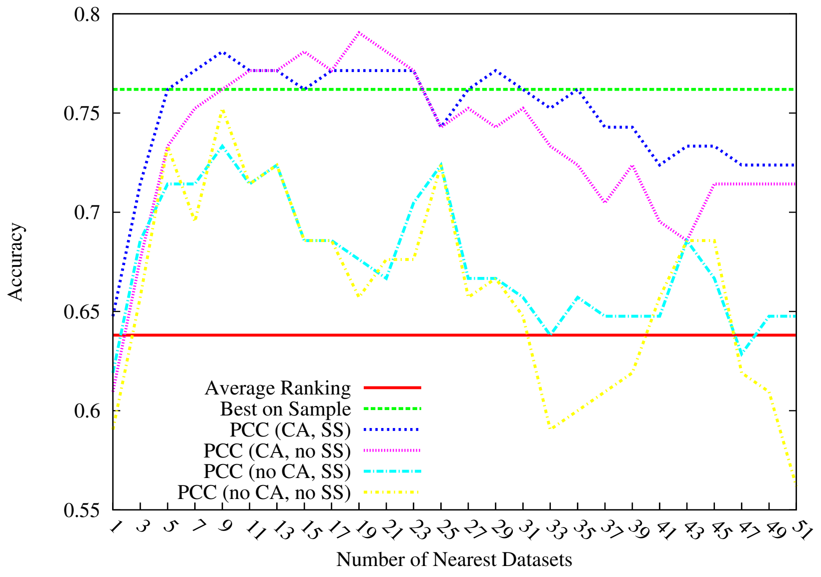
### 6.4.1 Predicting the Best Classifier

In the first experiment we aim to establish how well the meta-algorithm performs when the task is just 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 (similar to the evaluation by [87]). It uses predictive accuracy as the evaluation measure to identify similar datasets and select the best classifier (i.e., $P = P' = accuracy$). Pairwise Curve Comparison has several parameters. Most importantly, $T$ (the number of samples used to construct the learning curves) and $k$ (the number of nearest datasets to be identified). Furthermore, we explore the effect of Curve Adaptation (CA) and the Smaller Sample technique (SS) by comparing meta-algorithms having these options enabled against meta-algorithms having these options disabled.

Figure 6.3a shows the effect of varying the size of the learning curves. The $x$-axis shows the value of $T$ (number of samples used); the $y$-axis shows how often a given meta-learner predicted the best algorithm or an algorithm that was statistically equivalent to the best one. Note that not all the datasets allow for the construction of learning curves containing 7–12 samples, therefore sometimes predictions are made based on a smaller learning curve than the setup actually allowed. It can be seen that for most methods, using more samples results almost consistently in better performance, as is expected. All methods outperform the Average Ranking baseline already with a small number of samples, i.e., the sample-based techniques do not need much computational effort to perform better than the Average Ranking. It is clear that the Best on Sample and Pairwise Curve Comparison with Curve Adaptation perform clearly better, even when using only a small learning curve. There are some



(a) Varied value $T$, fixed $k = 17$



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

Figure 6.3: Performance of meta-algorithm on predicting the best classifier.

drops in performance, which can probably be attributed to characteristics of the specific datasets used (e.g., dimensionality). When the number of attributes of a dataset is in the same order or exceeds the number of observations, it becomes hard to learn from it and some (base-)classifiers might exhibit unstable behaviour.

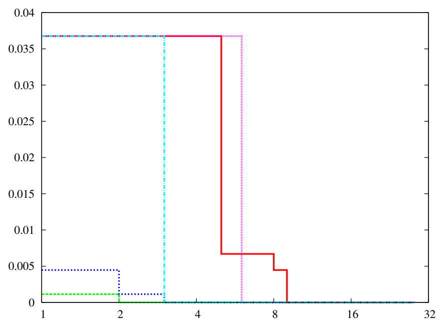
Figure 6.3b shows the effect of varying the number of nearest neighbours. In this setting, the number of samples for the learning curve $T$ was set to $5$, as this seems to be a reasonable small number. Hence, all predictions are made based on learning curves containing $5$ samples, with the largest sample consisting of $2^{5.5+0.5\cdot 5} = 256$ observations. Results obtained with higher values for $T$ are less interesting, as the time required for running the classifier on the consecutive samples increases. Average Ranking remains constant, as it does not use samples nor identifies the nearest datasets. Setting $k$ around $17$ 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 datasets which are not similar enough.

The Active Testing variants are not shown in both figures, as these are sequential methods that solely focus on the ranking of classifiers. It is undefined which classifier is advised first (see Algorithm 6.2). In our experiments, the classifier with the best Average Ranking is selected, so the accuracy of the first prediction of Active Testing would always be exactly the same as the Average Ranking method.

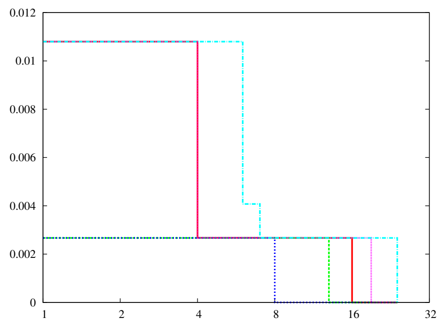
Both figures show similar trends. Best on Sample dominates the other techniques in most of the cases, even though this method is rather simple. Furthermore, both Pairwise Curve Comparison instances using Curve Adaptation (CA) outperform the instances without Curve Adaptation. Smaller Sample (SS) also 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) best or equally good classifier in more than $75\%$ of the cases, already when using a learning curve consisting of just $5$ samples.

## 6.4.2 Ranking of Classifiers

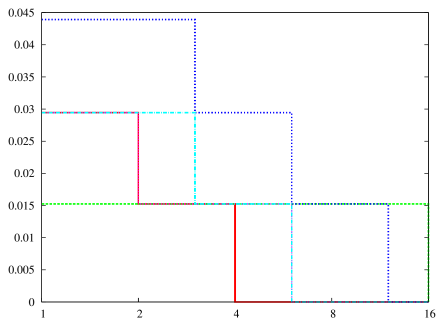
Sometimes, recommencing the best base-classifier in $75$ per cent of the cases is not good enough, hence we need to use a different approach. When the recommended base-classifier does not perform well enough, an alternative should be at hand. Rather than recommending a single classifier, a ranking should be created, ordering the classifiers on their likelihood of performing well on the dataset. This way, the user can make an informed decision about which models to try based on the available time and resources. The standard approach to evaluate such a ranking is to compute the Spearman Correlation Coefficient [144]. However, it has a drawback: it penalizes every



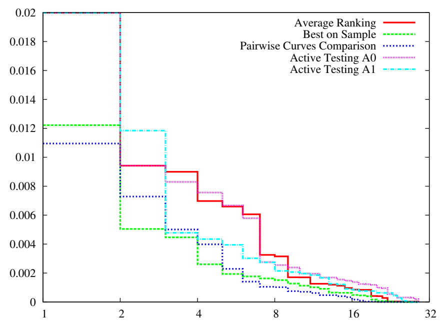
(a) 'Anneal'



(b) 'cmc'



(c) 'Electricity'



(d) Average over all datasets

Figure 6.4: Loss Curves. The $x$-axis shows the number of tests, the $y$-axis shows the loss.

wrongly ranked classifier equally, whereas we are mostly interested in classifiers at the top of the ranking. Furthermore, we do not care at all about incorrect ranked classifiers after the best one has been identified (although when applying the method in practise, we will not know when we have found the best method).

An alternative approach is to use Loss Curves as done in, e.g., [88]. The authors define *loss* to be the difference in accuracy between the current best classifier and the global best classifier. In order to find the global best classifier on a dataset, we evaluate all classifiers on this dataset in a certain order, for example by going down the ranking. A Loss Curve plots the obtained loss against the number of classifiers that have been tested. The goal is to find a classifier that has a low loss in relatively few tests.

In the following experiments, Pairwise Curve Comparison was run with $T = 5$ and $k = 17$. Likewise, Best on Sample was run with $T = 5$ (the sample on which base-

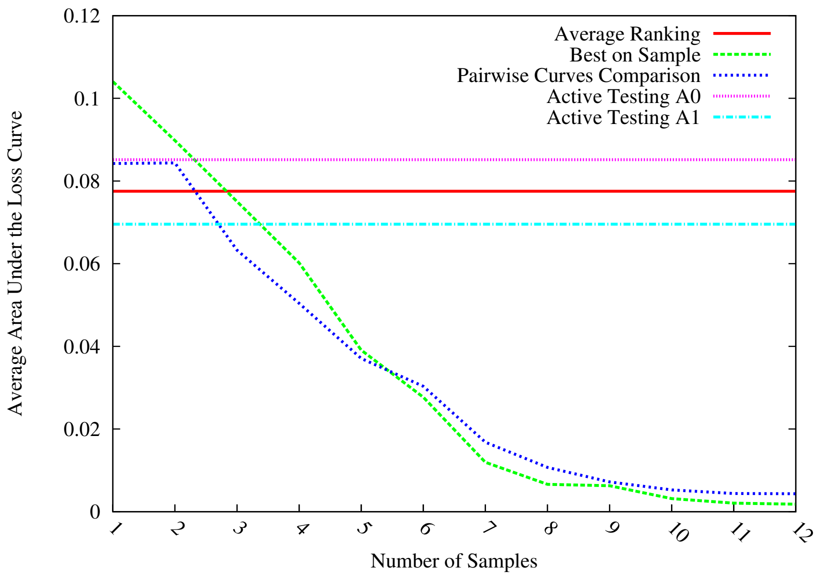


Figure 6.5: Average Area Under the Loss Curves for various meta-algorithms.

classifiers were evaluated contained 256 observations). Figure 6.4 shows loss curves of various strategies on some datasets. A loss curve is a visual evaluation measure showing how well strategies work after evaluating a given number of algorithms (tests). Similar to ROC Curves, for which commonly the 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 objectively compare various meta-algorithms. Usually, this is repeated over many datasets and an average Area Under the Loss Curve is reported, as is done in [127]. Sometimes an area of interest is defined, and the Area Under the Loss Curve for that interval is calculated [3]. This is useful when there is a specific budget (expressed in number of tests) for the meta-algorithm.

Figure 6.5 plots the effect of the learning curve size on the Area Under the Loss Curve. In order to not overload the figure, we omit the Pairwise Curve Comparison instances without Curve Adaptation or the Smaller Sample option. Pairwise Curve Comparison and Best on Sample techniques dominate the other techniques. Clearly, these are the only techniques that benefit from increasing the learning curve size. The other methods are by definition not influenced by this. Active Testing A1 outperforms the Average Ranking method. It can only compete with the sample-based methods that use a very small learning curve.

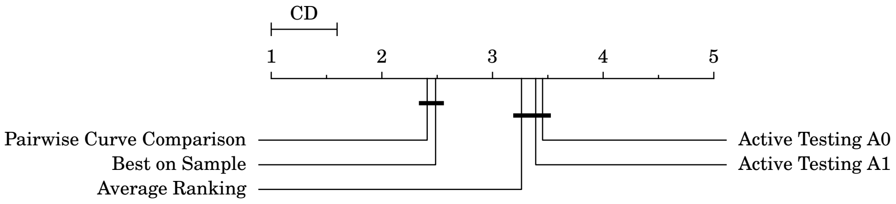


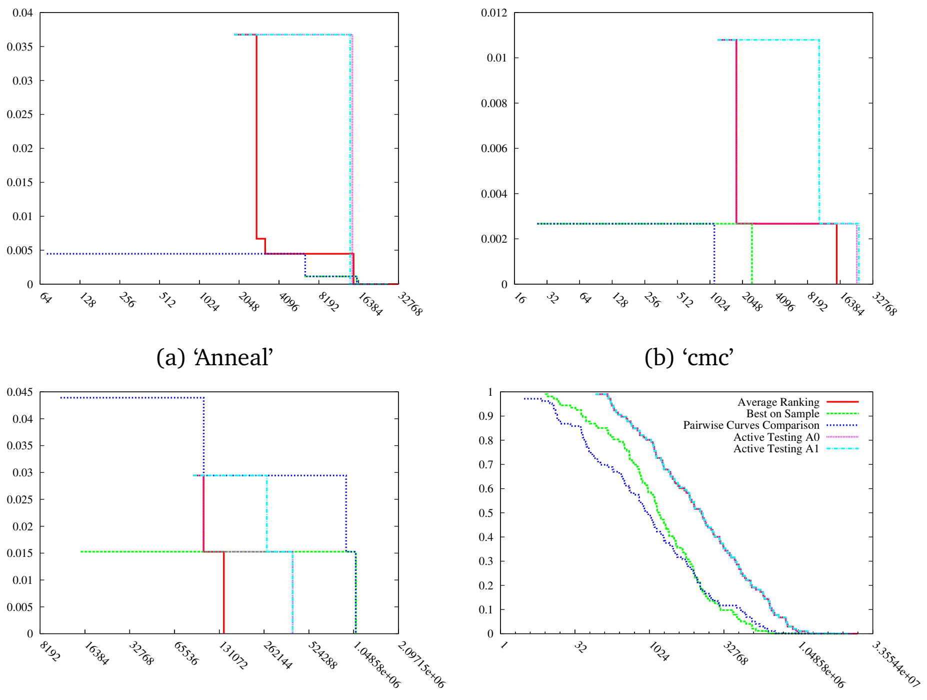
Figure 6.6: Results of Nemenyi test ($\alpha = 0.05$) on the Area Under the Loss Curve scores. Classifiers are sorted by their Average Ranking (lower is better).

Although interesting for observing general trends, we should be careful with drawing conclusions from the values from this plot: it can be dominated by outliers. Therefore, we also rank the meta-algorithms by their Area Under the Loss Curve per task, and calculate their average ranks over these. This shows which meta-algorithm performs best on most datasets, and enables us to do a statistical test over it, such as the Friedman test with post-hoc Nemenyi test. Figure 6.6 shows the result of the Nemenyi test. Pairwise Curve Comparison and Best on Sample perform statistically equivalent. The average ranks of these techniques are quite similar. Also Average Ranking, Active Testing A0 and Active Testing A1 perform statistically equivalent. Contrary to the results depicted in the Average Loss Curve (Figure 6.4d), it can be seen that the difference between Active Testing A1 and Active Testing A0 is minuscule. Because of the aforementioned reasons, conclusions based on the statistical test should have precedence over conclusions based on the average loss curve.

### 6.4.3 Loss Time Space

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. It describes how much time is needed on average to converge to a certain loss (lower is better). The faster such curve goes to a loss of zero, the better the technique is. They are commonly used in the Optimization literature (see, e.g., [74]).

Figure 6.7 shows the Loss Time Curves of various strategies on some datasets. The Loss Time Curves are drawn from the moment when the first cross-validation test has finished. As can be seen by the Loss Time Curves from Figure 6.7d, also in Loss Time space both Best on Sample and Pairwise Curve Comparison dominate the other techniques.



(a) 'Anneal'

(b) 'cmc'

(c) 'Electricity'

(d) Average over all datasets

Figure 6.7: Loss Time Curves. The $x$-axis shows the time in milliseconds, the $y$-axis shows the loss.

Similar to the Area Under the Loss Curve, we can calculate the Area Under the Loss Time Curve. One important aspect to consider is what loss is defined before the first cross-validation test has finished. In this work we use a loss of $1$ (which is the maximum possible loss), but also other values could be chosen. For example, [3] uses *default loss*, which is defined as the difference between the default accuracy of a dataset and the accuracy of the best performing algorithm on that dataset. Figure 6.8 plots the effect of the learning curve size on the Area Under the Loss Time Curve. The legend is omitted for readability, but the meta-algorithms have the same colours as in Figure 6.7. This again confirms the dominance of Pairwise Curve Comparison and Best on Sample. However, opposed to Figure 6.5, there is no general trend that Pairwise Curve Comparison and Best on Sample benefit from using more and bigger samples. One of the explanations for this is that although we are evaluating the meta-algorithms based on accuracy and run time, internally the meta-algorithms are

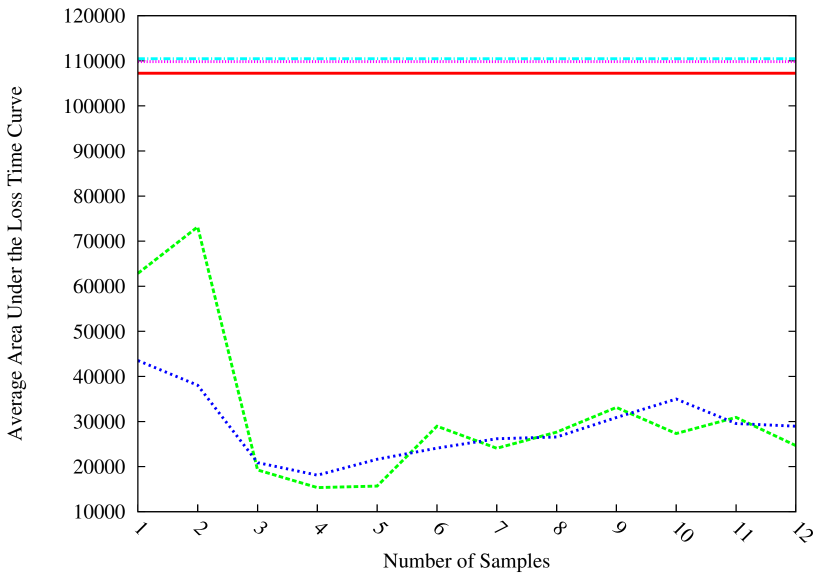


Figure 6.8: Average Area under the Loss Time Curve scores for the various meta-algorithms. The legend is omitted for readability.

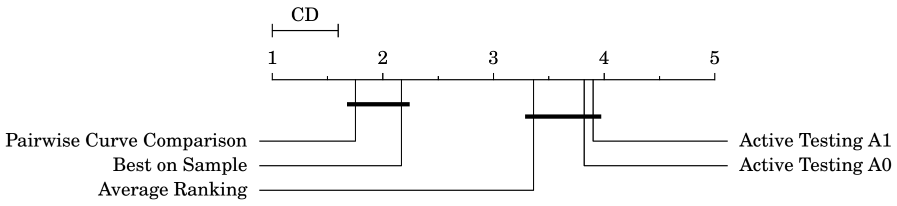


Figure 6.9: Results of Nemenyi test ($\alpha = 0.05$) on the Area Under the Loss Time Curves scores. Classifiers are sorted by their average rank (lower is better). Classifiers that are connected by a horizontal line are statistically equivalent.

not aware of this and consider only accuracy. They build a ranking solely based on accuracy, neglecting the run times, whereas high run times are punished by loss time curves.

As done before, we can rank the meta-algorithms for each dataset based on Area Under the Loss Time Curves. This enables us to do a statistical test on the performance in Loss Time space. Figure 6.9 shows the result of the corresponding Friedman with post-hoc Nemenyi test. The results seem similar to the results on the Area Under the

Loss Curves: Pairwise Curve Comparison and Best on Sample perform statistically equivalent, as well as Average Ranking, Active Testing A0 and Active Testing A1. However, much performance gain can still be obtained by making the meta-algorithms also consider run times, as we will see next.

### 6.4.4 Optimizing on Accuracy and Run Time

Next, our aim is to involve run times in the classifier selection process and evaluate whether this improves the performance of the meta-algorithm in Loss Time space. In this experiment, we will trade-off accuracy and run time. Recall that the meta-algorithms potentially use different evaluation measures to identify similar datasets and select classifiers. We adjust the methods to compare and select classifiers based on their $A3R'$ scores, as introduced in Chapter 6.3. Pairwise curve comparison still builds learning curves based on accuracy, but selects the most promising algorithm to test next based on a higher $A3R'$ score. Active Testing identifies similar datasets based on accuracy, but selects the most promising algorithm to test next based on a higher $A3R'$ score. Formally, evaluation measure $P = accuracy$, evaluation measure $P' = A3R'$. This way, decent classifiers that require a low run time are preferred over better classifier that require a high run time. The baseline methods can be adapted in a similar way such that they select classifiers based on $A3R'$.

Figure 6.10 compares the Average Loss Time Curves obtained using $A3R'$ with the Average Loss Time Curves based solely on accuracy. For example, in Figure 6.10a the red dashed line is exactly the same as the red line depicted in Figure 6.7d. These represent the 'vanilla' version of the Average Ranking method, that solely uses accuracy to build a ranking. In contrast, the solid red line is the version of the Average Ranking method that considers both accuracy and run time, by means of *A3R'*. The gain in performance is eminent. All methods using $A3R'$ converge to an acceptable loss orders of magnitude faster than the ones based on solely accuracy. For example, Pairwise Curve Comparison with $A3R'$ converges on average in $5$ seconds to a loss of less than $0.001$, whereas vanilla Pairwise Curve Comparison takes on average $454$ seconds for this. This can be very useful in practise, when data needs to be processed at high speed.

Again, we do a Friedman with post-hoc Nemenyi test, based on the scores for the Area Under the Loss Time Curves per task. Figure 6.11 shows the results. All the meta-algorithms that rely on *A3R'* to construct the ranking perform statistically significant better than their vanilla counterparts. Furthermore, all meta-algorithms that construct a ranking based on *A3R'* perform better than the meta-algorithms that just use accuracy. The average ranking method scores surprisingly well in practise, which is confirmed by a similar study [3].

## 6.5 Conclusion

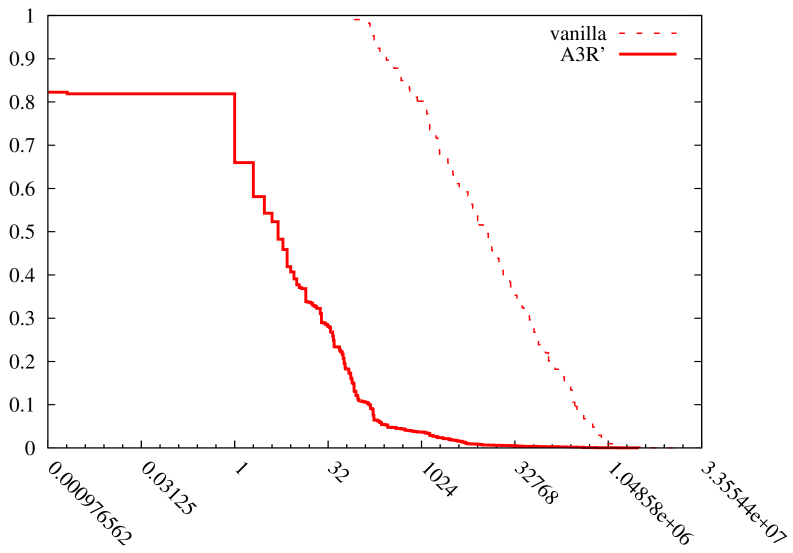
This chapter addresses the problem of algorithm selection under a budget, where multiple algorithms can be run on the full dataset until a given budget expires. This budget can be expressed as the number of cross-validation tests; in that case the user can only select a limited number of algorithms and parameter settings to evaluate. The budget can also be expressed in time, where there is a certain amount of time in which various algorithms with multiple parameter settings can be evaluated, after which the best must be selected.

We have extended the method presented in [87] such that it generates a ranking of classifiers, rather than just predicting the single best classifier. The ranking suggests in which order the classifiers should be tested. Based on such ranking a loss curve can be constructed, showing which meta-algorithms perform best after a given number of tests. In order to objectively compare various meta-learning algorithms and to enable statistical tests, the Area Under the Loss Curve can be calculated. Interestingly, a simple and elegant baseline method called Best on Sample performs equally well in our experiments, selecting a good classifier using only a few tests.

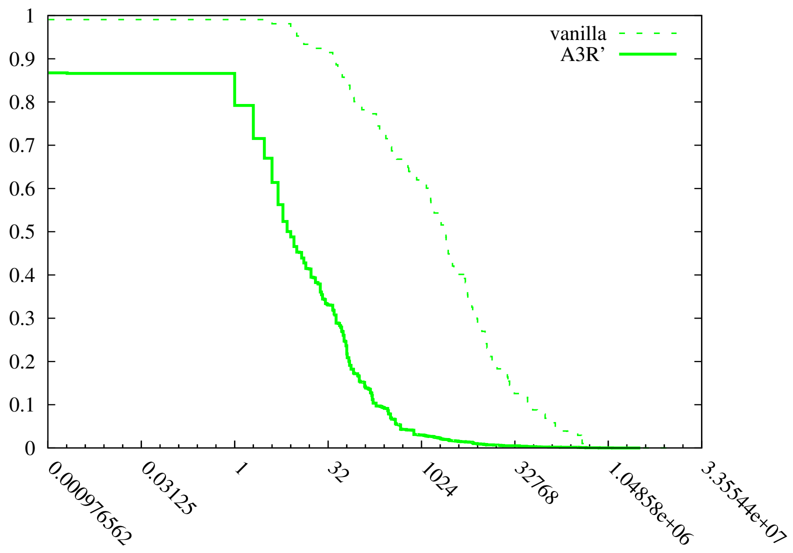
However, when tested in the more realistic setting where the budget is expressed in time, rather than a number of tests, the performance of meta-algorithms becomes unpredictable. This was reflected in Figure 6.8, where there was no general trend that Pairwise Curve Comparison and Best on Sample benefit from using more and bigger samples.

When evaluating in the time budget setting, the meta-algorithms should be aware of the run time of the classifiers. A measure such as *A3R'* (which trades off accuracy and run time) can provide this, although other measures are also capable of doing so as well (e.g., $ARR$). We evaluated all the meta-classifiers based on the Area Under the Loss Time Curve. The meta-algorithms using *A3R'* consistently outperformed the meta-algorithms that did not use this measure. This suggests that *A3R'* is very suitable for algorithm selection applications with a limited time budget. Apparently, it is better to try many reasonable (and fast) classifiers that a few potentially very good (but expensive) classifiers.

These results are not unexpected; when using an evaluation measure that partly considers run time (as the Area Under the Loss Time Curve does) the meta-algorithms should be aware of the run time of the algorithm configurations. Using the novel algorithm selection criterion, a reasonable performing classifier was typically selected orders of magnitude faster than otherwise. Recall that Pairwise Curve Comparison with $A3R'$ converges on average in $5$ seconds to a loss of less than $0.001$, whereas vanilla Pairwise Curve Comparison takes on average $454$ seconds for this; a speed-up of almost factor $100$.

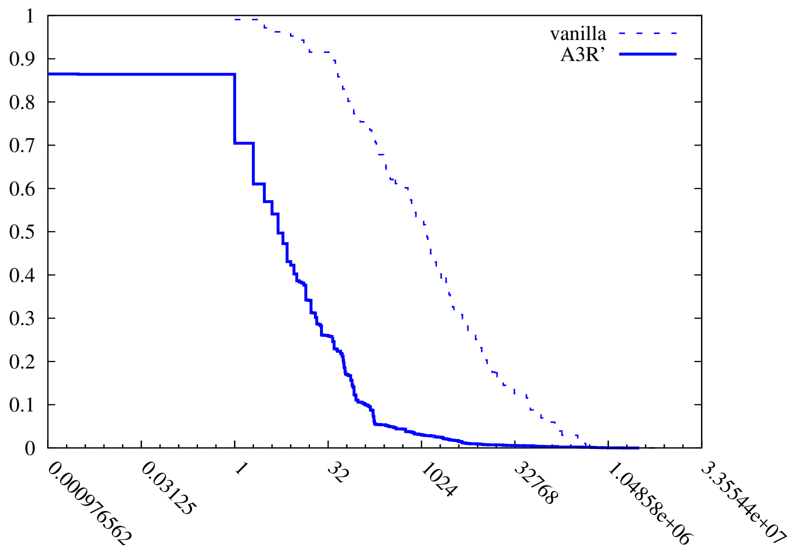


(a) Average Ranking

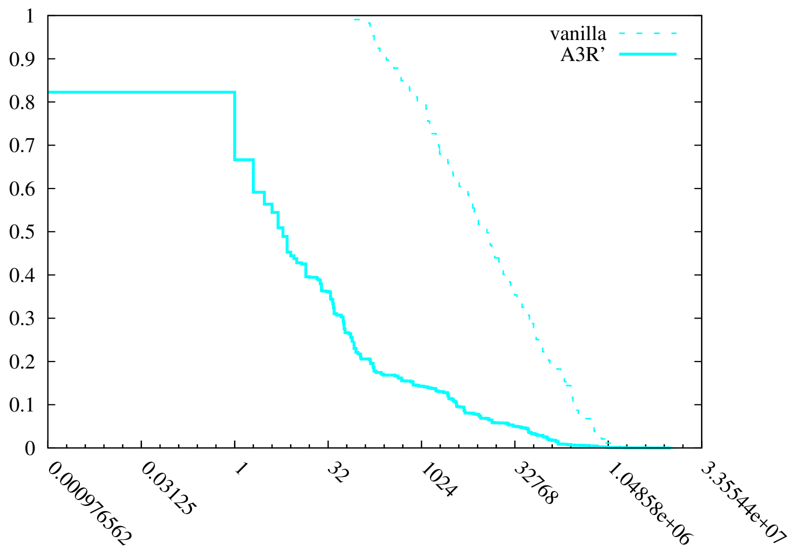


(b) Best on Sample

Figure 6.10: Loss Time Curves. The $x$-axis shows the time in milliseconds, the $y$-axis shows the loss.



(c) PCC



(d) Active Testing A1

Figure 6.10: Loss Time Curves. The $x$-axis shows the time in milliseconds, the $y$-axis shows the loss (continued).

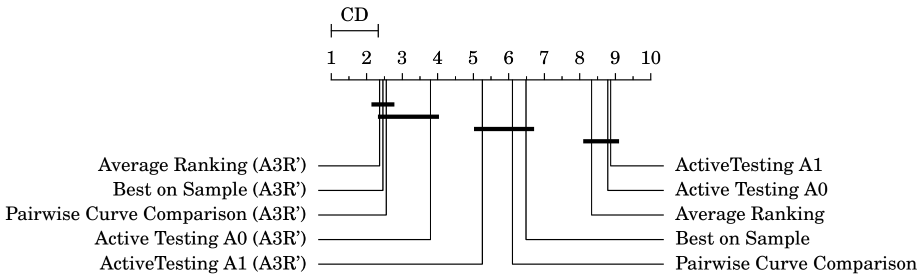


Figure 6.11: Results of Nemenyi test ($\alpha = 0.05$) on the Area Under the Loss Time Curve scores. Classifiers are sorted by their average rank (lower is better). Classifiers that are connected by a horizontal line are statistically equivalent. The parameters are fixed to $t = 5$, $k = 17$.

Future work should focus on applying this technique on the full meta-dataset of OpenML. Currently, OpenML already contains many datasets from various domains, e.g., data streams, text mining datasets and QSAR datasets. In this work we carefully selected a set of datasets to perform the meta-learning tasks on. However, by definition this approach embraces 'the strong assumption of Machine Learning' (as defined by Giraud-Carrier and Provost [60]). The strong assumption of Machine Learning is that the distribution of datasets that we will model is explicitly or implicitly known, at least to a useful approximation (see also Chapter 3). To the best of our knowledge, there is currently no work in Machine Learning that does not make this assumption. However, Machine Learning and meta-learning do not require such strong assumptions. In other words, one very promising area of future work would be setting up a meta-learning experiment that involves all datasets from OpenML. Such experiment would face many challenges, as one would be dealing with data and classifiers from various domains, but the rewards could be high. Trading off accuracy and run time using $A3R'$ might be a key aspect to this. This would be the first research that convincingly demonstrates the true power of meta-learning, without making any strong assumptions.