5

Metalearning Approaches for Algorithm Selection II

Summary. This chapter discusses different types of metalearning models, including regression, classification and relative performance models. Regression models use a suitable regression algorithm, which is trained on the metadata and used to predict the performance of given base-level algorithms. The predictions can in turn be used to order the base-level algorithms and hence identify the best one. These models also play an important role in the search for the potentially best hyperparameter configuration discussed in the next chapter. Classification models identify which base-level algorithms are *applicable* or *non-applicable* to the target classification task. Probabilistic classifiers can be used to construct a ranking of potentially useful alternatives. Relative performance models exploit information regarding the relative performance of base-level models, which can be either in the form of rankings or pairwise comparisons. This chapter discusses various methods that use this information in the search for the potentially best algorithm for the target task.

5.1 Introduction

In this chapter we discuss different approaches to algorithm selection. Some of them were used in early studies involving metalearning, but many methods have been upgraded and represent quite competitive variants.[1]

In Section 5.2 we discuss the use of suitable regression models within the algorithm recommendation system. If we can predict the performance of a given set of base-level algorithm (e.g., classifiers), we can order them and generate, in effect, a ranking. Then we can use just the topmost element, or else use the top-N strategy discussed in Chapter 2, to search for the potentially best algorithm.

An alternative approach is discussed in Section 5.3 which uses a classification algorithm at a meta-level. Here the aim is to identify which of the base-level algorithms are simply applicable to the target machine learning (e.g., classification) task.

If we use probabilistic classifiers at the meta-level, which provide not only the class (e.g., applicable or non-applicable) but also numeric values related to the probability of classification, then the probabilities can again be used to elaborate a ranking. As in the previous case, the potentially best possible base-level algorithm can be identified.

[1] This part is based on pages 36–42 in the first edition of this book. The text has been extended and upgraded.

P. Brazdil et al., *Metalearning*, Cognitive Technologies,
https://doi.org/10.1007/978-3-030-67024-5_5

One important class of approaches is based on pairwise comparisons of the performance of the base-level algorithms. In these approaches the algorithm recommendation system converts the actual base-level performance into information that captures relative performance. Pairwise models are discussed in Sections 5.4, 5.5, and 5.6.

Section 5.7 describes a two-stage approach. First, pairwise models are used to generate features, which are subsequently used to generate predictions.

Some of the methods discussed above conduct the pairwise tests in a kind of preprocessing stage. However, this has a disadvantage, as not all of these tests may be needed in the process of identifying the potentially best algorithm. Some methods use a more intelligent way to identify the potentially useful tests. This method, referred to as *active testing*, is discussed in Section 5.8.

Section 5.9 discusses the possibility of using non-propositional (i.e., relational or ILP-based) representation for the description of datasets. In consequence, the corresponding methods need to be upgraded to deal with such representations.

5.2 Using Regression Models in Metalearning Systems

In Chapter 2 we discussed a metalearning system that converted base-level performance data into ranks, and the aim was to predict a ranking of base-level algorithms together with their configurations. Let us for simplicity represent each combination a_θ simply by $\theta \in \Theta$ and refer to it simply as a *configuration*. Symbol Θ is used to represent all possible configurations, or the distribution of all possible configurations.

In some approaches, the basic idea is to estimate the performance of each base-level configuration and select the one that leads to the best results. Some authors refer to these models as *empirical performance models* (EPMs) (Leyton-Brown et al., 2009; Hutter et al., 2014; Eggensperger et al., 2018).

5.2.1 Empirical Performance Models

The existing approaches can be divided into two groups, according to whether the regressor uses metadata obtained from only the current dataset, or from other datasets as well.

Using metadata from the current dataset

Let us analyze first the approaches that do not use metadata obtained on other datasets. This approach forms the basis of *Auto-WEKA* system (Thornton et al., 2013).

When no metadata from other datasets is available, the only source of metadata can be from earlier runs on the dataset itself. At first, several preselected configurations are tested and the outcome constitutes the resulting metadata. These configurations can either be selected at random or by initial design (see, e.g., Pfisterer et al. (2018); Feurer et al. (2018)). The metadata consists of n cases (meta-examples) of the form

$$MetaD \equiv (\theta_i, P_i)^n, \tag{5.1}$$

where $i = \{1, 2, \ldots, n\}$, θ_i represents the i^{th} algorithm configuration, and P_i is the performance of this configuration. A suitable regressor can be trained on this metadata, and this way we obtain a meta-level model $MetaM$:

$$MetaM \leftarrow Train(MetaD). \tag{5.2}$$

As in all learning tasks, the performance depends on how many cases are available. So we need a sufficient number of cases in $MetaD$ to obtain a reasonable performance. The empirical performance model $MetaM$ can be used to predict (more precisely, estimate) the performance for some new configuration that is not even a part of the existing metadata $MetaD$:

$$\hat{P}_k \leftarrow Predict(MetaM, \theta_k). \tag{5.3}$$

So different configurations can be queried, and the one with the estimated best performance selected and evaluated on the current base-level dataset. More details regarding this are given in Section 6.3.

Approaches that use metadata from other datasets

Some researchers used information on other datasets in the search for the best possible configuration (Gama and Brazdil, 1995; Sohn, 1999; Köpf et al., 2000; Bensusan and Kalousis, 2001; Feurer et al., 2014). Let us see how this works. First, let us extend the notation in Eq. 5.1 to include also different datasets,

$$MetaD \equiv (\theta_i, d_j, P_{i,j})_{i=1,j=1}^{n,\ m}, \tag{5.4}$$

where θ_i represents a configuration and d_j the dataset on which the performance $P_{i,j}$ is measured. Each dataset is characterized by metafeatures $mf(d_j)$.[2] An exhaustive overview of possible metafeatures is presented in Chapter 4.

The meta-level model, $MetaM$, can be trained as shown in Eq. 5.2. Obviously, constructing such a model is more complex, as in the previous case, as it involves various datasets. Predictions can be done by invoking

$$\hat{P}_{k,t} \leftarrow Predict(MetaM, \theta_k, d_t). \tag{5.5}$$

The prediction task has two parameters, if we disregard $MetaM$ itself. One is the configuration θ_k. The aim is to find a configuration $\theta*$ that achieves the highest possible performance on the target dataset d_t. So one naive way to obtain this is to consider various configurations generated by the meta-model and select the best one.

Various researchers have devised other, more effective approaches. Typically, the search would be done in an iterative manner. The best configuration found in one iteration is used to update the meta-level model or else to condition the future search in some way. This way the method requires less time to identify the potentially best solution. Some metalearning methods that follow this line are discussed further on in this chapter (Sections 5.6 and 5.8) and in Chapter 6 (Sections 6.3 and 6.4).

Let us come back to Eq. 5.5 describing how the prediction is obtained. Another parameter in this process is the target dataset d_k characterized by metafeatures $mf(d_k)$. Although this parameter is fixed, it affects the performance of the meta-level model. So one question is — how can it be exploited to boost up the performance?

This can be achieved by *adapting* the meta-level model $MetaM$, constructed on the basis of the test results on different datasets in $D \equiv (d_i)^m$. So one important part of

[2]Note that the above notation implies that the configurations are represented in the form of a finite list and hence can be enumerated. It is not required that a performance result exists for every configuration and dataset pair.

the adaptation process is to consider the similarity between the target dataset d_t and the individual datasets in D. Let us represent the similarity by $Sim(d_t, d_j)$. Chapter 4 (Section 4.10) shows how the similarity can be calculated using the dataset measures.

Chapter 2 (Section 2.2) shows how the similarity can be exploited to adapt the ranking approach to provide a specific ranking. As Brazdil et al. (2003) have shown, this leads to enhanced performance. A similar adaptation process is described in Subsection 5.8.2 in relation to the *active testing* (AT) method.

5.2.2 Normalizing performance

Earlier we have shown the format of metadata $MetaD$ (see Eq. 5.1). We see that it involves performance results obtained on different datasets. However, as has already been noted in Chapter 3, the range of performance values (in absolute terms) may vary substantially for different datasets. An accuracy of 90% may be quite high on a classification problem but low on another one. If the aim is to exploit metadata from different datasets in a metalearning system, it is useful to normalize the performance values in a kind of preprocessing step. Chapter 3 (Section 3.1) describes some common approaches that can be used to do this.

5.2.3 Performance models

Performance models can be represented in an extensional form or in an intensional one (e.g. using a function). The former corresponds to enumeration of different cases, as in instance-based learning (IBL) approaches to machine learning. This representation has been used in many ranking approaches, where the cases are reordered according to the chosen performance value.

Our aim in this chapter is to discuss models that use an intensional form. In principle, it is possible to use any regression algorithms for the task of predicting performance. However, some have proved more useful than others, when judged by the quality of predictions.

In one rather early work by Gama and Brazdil (1995), the authors used linear regression, regression trees, model trees, and IBL to estimate the error of a large number of base-level algorithms.

In another study, a comparison between *Cubist*[3] and a *kernel method* was carried out on a problem of estimating the error of ten classification algorithms (Bensusan and Kalousis, 2001). Results showed a slight advantage for the kernel method.

Some researchers have adopted *Gaussian processes* (e.g., Rasmussen and Williams (2006)), others *random forests* (RFs) (e.g. Eggensperger et al. (2018); Hutter et al. (2014); Leyton-Brown et al. (2009)). As both types of meta-level models are discussed in more detail in Chapter 6, we do not provide more details here.

Some proposals combined several models at the meta-level. One early metalearning approach has been discussed by Gama and Brazdil (1995). They showed that a linear combination of the meta-level models yields better results than any of these models considered individually.

[3]Cubist combines rules with regression trees and can be seen as an extension of the M5 model tree (Quinlan, 1992).

Table 5.1: Examples of different forms of recommendation

	Algorithms					
	a_1	a_2	a_3	a_4	a_5	a_6
(1) Estimates of performance	0.89	0.68	0.90	0.74	0.81	0.75

	Rank					
	1	2	3	4	5	6
(2) Ranking (linear and complete)	a_3	a_1	a_5	a_6	a_4	a_2

5.2.4 Clustering trees

Clustering trees can be used to induce a single model for several target variables and hence provide multitarget prediction (Blockeel et al., 1998). In our case, the performance of multiple base-algorithms represented in the form of the multiple targets.

Clustering trees are obtained with a common algorithm for top-down induction of decision trees (TDIDT) that tries to minimize the variance of the target variables for the cases in every leaf (and maximize the variance across different leaves). They have been applied to the problem of estimating the performance of several algorithms (Todorovski et al., 2002). The decision nodes represent tests on the values of metafeatures, and the leaf nodes represent sets of performance estimates, one for each algorithm.

The results obtained with clustering trees are comparable to those obtained with the approach using separate models, with the advantage of improved readability, because the former approach generates a single model rather than several models (Todorovski et al., 2002).

5.2.5 Transforming performance predictions into rankings

A set of performance estimates of different base-level algorithms can be transformed into rankings by ordering the algorithms according to their expected performance (Sohn, 1999; Bensusan and Kalousis, 2001). This is illustrated in Table 5.1. The performance values (line 1) can be used to reorder the corresponding algorithms to generate a ranking (line 2). This ranking can be followed in a top-*n* fashion (as described in Chapter 2, Section 2.2).

It could be argued that, in many cases, the user simply requires to identify the potentially best algorithm and hence detailed information on performance has simply an auxiliary role.

Not much work has been dedicated to empirical comparisons of different forms of recommendation discussed here and in Chapter 2 (Section 2.1). One early study was done by Bensusan and Kalousis (2001). However, it would be useful to conduct a new study which would incorporate up-to-date approaches and methods.

5.2.6 Predicting performance for each example

A different approach to estimate the performance of an algorithm is based on the use of metalearning model to predict the performance of a base-level algorithm on each individual base-level example (Tsuda et al., 2001). For instance, in a classification problem,

the metalearning model predicts whether the base-level algorithm correctly predicts the class for each test example. A prediction of the performance of the algorithm on the dataset is obtained by aggregating the set of individual predictions that are obtained with the metalearning model. In the classification setting, the predicted performance of the algorithm could be given by the predicted accuracy.

5.2.7 Advantages and disadvantages of performance predictions

Advantages

By providing performance estimates for each algorithm rather than, for instance, a rank, more information is provided. As we have argued earlier (Subsection 5.2.5), the performance estimates can be converted into rankings quite easily.

Additionally, each regression problem can be solved independently, generating one model for each base-algorithm. With this approach, it is easier to change the portfolio comprising the set of base-algorithms. Removing an algorithm simply means eliminating the corresponding meta-level model, while inserting a new one can be done by generating the corresponding meta-level model. In both cases, the meta-level models of the remaining algorithms are not affected.

Disadvantages

Generating as many meta-level models as there are algorithms has a disadvantage. It is not trivial to understand when an algorithm performs better than another one and vice versa. Take, for instance, the rules presented in Table 5.2 that were selected from models that predict the error of `C4.5` and `CN2` (Gama and Brazdil, 1995). The metafeatures are: $fract1$, the first normalized eigenvalues of canonical discriminant matrix; $cost$, a Boolean value indicating if errors have different costs; and Ha, the entropy of attributes. These models do not describe directly the conditions when `C4.5` is better than `CN2` and vice versa.

Table 5.2: Four sample rules that predict the error of `C4.5` and `CN2` (Gama and Brazdil, 1995).

Algorithm	Estimated Error	Conditions
`C4.5`	22.5	$\leftarrow fract1 > 0.2 \wedge cost > 0$
`c4.5`	58.2	$\leftarrow fract1 < 0.2$
`CN2`	8.5	$\leftarrow Ha \leq 5.6$
`CN2`	60.4	$\leftarrow Ha > 5.6 \wedge cost > 0$

It can be expected that predicting individual performance values is much harder than discriminating between a finite number of classes or predicting rankings.

Additionally, the fact that several regression problems are solved independently can be regarded as a disadvantage. In fact, the error in itself is not so important as the question of whether it affects the relative order of the algorithms. This issue was addressed by other researchers who have provided models for pairs of algorithms. One work on this line is discussed in Section 5.4.

Another disadvantage of this approach was that it provided a pointwise prediction. It is well known that the performance of a given algorithm normally varies substantially across different datasets. Some examples of this are shown in Chapter 16. Consequently, interval prediction provides information about the variability of the performance, which can be exploited in algorithm selection and configuration. The methods discussed in Chapter 6 do just that.

5.3 Using Classification at Meta-level for the Prediction of Applicability

In one early work of Brazdil et al. (1994), the aim was to predict whether a given base-level classification algorithm was applicable (with relatively low error) or non-applicable (with relatively high error) to a given target dataset. Following the notation introduced in Section 5.2.1, the metadata $MetaD$ of the form $(\theta_i, d_j, P_{i,j})_{i=1,j=1}^{n,\ m}$ was transformed into metadata suitable for a classification algorithm. That is, each $P_{i,j}$ was discretized into two possible categorical values, *applicable* and *non-applicable*.

A suitable classification algorithm was then used at the meta-level. Each dataset was described using dataset characteristics (see Chapter 4 for details), and then dataset similarity was also explored to generate a specific recommendation for the target dataset.

The fact that the prediction is categorical class value (e.g. applicable/non-applicable) may seem as a limiting factor, as it is not possible to determine the order in which the base-algorithms should be tried out. However, if we included $P_{i,j}$ as a feature and opted for a probabilistic classifier at the meta-level, this limitation would be mitigated. The class value generated by the probabilistic classifier could be accompanied by the estimate of probability. Hence, it would be possible to order the base-level algorithms according to this probability. So the effects could be similar to those obtained with classical ranking approaches (see Chapter 2).

5.3.1 Classification algorithms used at meta-level

Given their wide availability, many different classification algorithms have been used in meta-level learning. Early studies focused on the decision tree classifier (Brazdil et al., 1994) or k-NN classifier (Brazdil et al., 2003).

An extreme example is to use the same set of base-level algorithms also at the meta-level (Pfahringer et al., 2000; Bensusan and Giraud-Carrier, 2000). In this work, the ten classification algorithms used were quite diverse, including decision trees, a linear discriminant, and neural networks, among others. The authors compared pairs of meta-level classifiers on several metalearning problems. The comparisons were not conclusive in one of the studies (Bensusan and Giraud-Carrier, 2000), while in another study (Pfahringer et al., 2000) the results indicated that decision tree- and rule-based models lead to better metalearning results. In another study (Kalousis, 2002; Kalousis and Hilario, 2000), the authors compared four variants of decision tree classifiers, IBL, and boosted C5.0 at the meta-level. The best results were obtained by boosted C5.0.

5.4 Methods Based on Pairwise Comparisons

Various authors have studied the issue of using pairwise models for the task of predicting the potentially best algorithm, or alternatively, a ranking of algorithms. Some early work

focused on the problem of predicting which of the two algorithms is likely to perform better on a new dataset (Pfahringer et al., 2000; Fürnkranz and Petrak, 2001; Soares et al., 2001; Leite and Brazdil, 2004, 2005).

In subsequent years, some researchers have addressed the issue of how the method of pairwise comparisons could be incorporated into a method that provides a ranking of all algorithms, or else identify the first element, i.e., the potentially best-performing algorithms (plus possibly all algorithms with equivalent performance).

Some of the methods generate various pairwise models, which are used afterwards to obtain a ranking (Leite and Brazdil, 2010; Sun and Pfahringer, 2012, 2013).

Other methods use an iterative method which conducts pairwise tests as the method proceeds. When this process is terminated, the potentially best algorithm is returned (Leite and Brazdil, 2007; van Rijn et al., 2015). More details about these methods are given in the following subsections.

5.4.1 Pairwise tests that exploit landmarkers

Pfahringer et al. (2000) proposed the use of simplified and fast versions of algorithms referred to as *landmarkers* in pairwise comparisons (see Chapter 4, Section 4.2 for more details on landmarkers). The authors have shown that landmarkers can be used for the problem of predicting which of the two algorithms is likely to perform better on a new dataset.

Other researchers have proposed that, apart from simplified versions of algorithms, one could also use simplified versions of datasets. This concept was referred to as *subsampling landmarkers* (Fürnkranz and Petrak, 2001; Soares et al., 2001).

The experimental results carried out at the time did not seem to show a clear advantage of this approach. However, as was shown later (van Rijn et al., 2015; van Rijn, 2016), the method that uses the performance on a single subsample of the data to select the classifier that performs best is rather competitive and beats many more complex methods. This method is referred to as the *best-on-sample* method. In other communities it is referred to as *constant prediction* or *constant liar*. This simple baseline gave rise to complex methods, such as *successive halving* (Jamieson and Talwalkar, 2016) and *Hyperband* (Li et al., 2017).

One surprising result of the early work cited earlier was that using more information in the form of more samples did not lead to marked improvement. This led some researchers (Leite and Brazdil, 2004, 2005) to investigate the reasons for this. This study led to a new variant of the method which is discussed in the next subsection.

5.4.2 Pairwise method oriented towards partial learning curves

Leite and Brazdil (2004, 2005) proposed a variant of the pairwise method that used a series of samples as data characteristics. The series of samples used represents, in effect, a *partial learning curve*. The method relies on a meta-dataset that includes full learning curves of all algorithms on historical datasets. On the new target dataset it is necessary to obtain partial learning curves. This process typically takes less time than obtaining the performance value on a full dataset.

Based on the partial learning curve on the target dataset, and full learning curves on historical datasets, the meta-algorithm can determine which algorithm to recommend.

Figure 5.1 shows an example of learning curves. The x-axis shows the number of data points; the y-axis shows the measured performance. We can see some typical behavior

of learning curves: (i) most classifiers typically perform better when larger data samples are available, (ii) this is not always the case, as sometimes performance may drop when more data is available, and (iii) learning curves can cross, as some classifiers that do not perform well on small samples can attain higher performance on larger data samples.

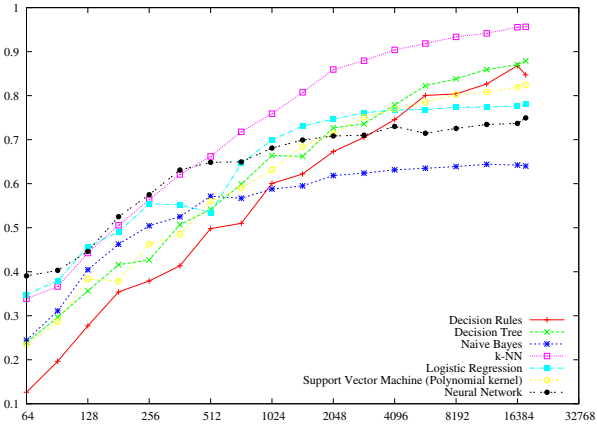

Fig. 5.1: Learning curves of various Weka classifiers on the "letter" dataset. Image taken from van Rijn (2016).

The method described can be used to solve the following problem: Given two algorithms a_p and a_q, which of them will most likely perform best on the current dataset? The method involves the following steps:

- Determine how to represent learning curves;
- Obtain a partial learning curve of both algorithms a_p and a_q on the target dataset;
- Identify datasets with most similar curves while examining the meta-database of past experiments;
- Use the retrieved curves to predict the performance a_p and a_q for the target dataset. Predict which algorithm will achieve a higher performance.

The main idea of this method is to save time. A more expensive CV test is substituted by a cheaper estimate, as is done when using surrogate models discussed in Chapter 6.

As this idea was reused in follow-up work (e.g., Leite and Brazdil (2010); van Rijn et al. (2015)), the method is explained in some detail in the following subsections.

Representation of partial learning curves

Each partial learning curve is represented by a sequence of performance values of a given algorithm on a given dataset on samples of increasing size. Formally, partial learning curve P_{a_p,d_i} is represented by a sequence of elements of the form $p_{a_p,d_i,k}$, where a_p represents an algorithm, d_i a dataset, and k an index of a sample which varies from 1 to k_{max}.

Following previous work of Provost et al. (1999), the sizes follow a geometric progression. Leite and Brazdil (2005) have set the size of the k-th sample to the rounded

value of $2^{6+0.5\times k}$. So, the size of the first sample is set to $2^{6.5}$, giving 91 after rounding, and the second sample is set to 2^7, i.e., 128, etc. This scheme was reused in various subsequent papers.

Conducting tests on the target dataset

Tests are conducted with the aim of obtaining a partial learning curve of both algorithm a_p and a_q on the target dataset.

Identification of the most similar learning curve

In this step the aim is to identify datasets whose learning curves are most similar to the partial learning curve elaborated for the target dataset. Once such curves have been identified, the performance for any sample size can be retrieved and used for prediction.

The similarity between two learning curves is judged by an adaptation of the nearest neighbor algorithm (*k*-NN). The distance measure used between datasets d_i and d_j for a given pair of algorithms a_p and a_q is defined by the sum of two distances, one for algorithm a_p and the other for a_q:

$$d_{a_p,a_q}(d_i,d_j) = d_{a_p}(d_i,d_j) + d_{a_q}(d_i,d_j). \tag{5.6}$$

The first term determines the distance between two learning curves of algorithm a_p, one on dataset d_i and the other on dataset d_j. It is calculated as follows:

$$d_{a_p}(i,j) = \sum_{k=1}^{k_{max}} (p_{a_p,i,k} - p_{a_p,j,k}). \tag{5.7}$$

As the learning curve has k points, Eq. 5.7 sums the individual distances for all k points. The second term, i.e. $d_{a_q}(d_i,d_j)$, is calculated in a similar way.

Adaptation of retrieved curves

Leite and Brazdil (2005) have observed that, although the retrieved learning curve may often have the same shape, it may be shifted up (or down) with respect to the curve on the target dataset. They have concluded that this may impair the quality of prediction. To avoid this shortcoming the authors have proposed an additional *adaptation* step, related to the notion of *adaptation* in case-based reasoning (Kolodner, 1993; Leake, 1996).

Here the adaptation procedure involves moving the retrieved performance curve P_r to the partial performance curve P_t generated for the target dataset and obtaining P'_r. Adaptation is done by multiplying each item in P_r by the scale coefficient f. So the k-th item in the curve relative to algorithm a is transformed as follows: $p'_{a,r,k} = f \times p_{a,r,k}$. The scale coefficient f is designed to minimize the Euclidean distance between the two curves on the initial segment. The authors have used different weights for the different items in the curve in accordance with the corresponding sample size. The following equation shows how f is calculated:

$$f = \frac{\sum_{k=1}^{k_{max}} (p_{a,t,k} \times p_{a,r,k} \times w_k^2)}{\sum_{k=1}^{k_{max}} (p_{a,r,k}^2 \times w_k^2)}. \tag{5.8}$$

The process of adaptation is illustrated in Figure 5.2, reproduced from earlier work (Leite and Brazdil, 2005).

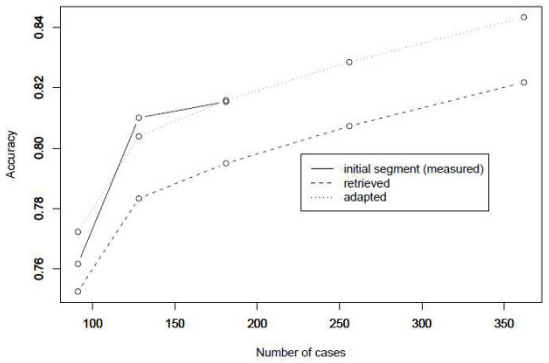

Fig. 5.2: Adaptation of the retrieved learning curve

Carrying out predictions for k nearest datasets

Assuming that k nearest datasets have been identified for algorithm a_p using the method described above, the next step is to obtain k predictions for the sample size S_t corresponding to the size of the target dataset. The authors then calculate a mean of the k values, which is then used as the predicted performance value of a_p. The predicted performance value of a_q is calculated in a similar way. The two values are compared to determine which of the two algorithms is better. In the experiments carried out the setting of $k_{max} = 3$ was used, determining that the partial learning curve should include performance values for three data samples.

Main results

The authors have studied a pair of algorithms, which included C5 (Quinlan, 1998) and SVM with radial basis kernel (the implementation in e1071 in R (Dimitriadou et al., 2004)). The authors have compared the classification accuracy of the proposed method (A-MDS) with the classical approach that used a set of dataset characteristics (MDC). The performance of MDS that used just one sample (91 cases) was better than the one obtained with MDC.

The authors have also shown that, if a learning curve with more samples was used, the performance would increase. This is not true for a variant that does not use adaptation in the process. This explains why Soares et al. (2001) obtained somewhat discouraging results earlier. As adaptation was not used, using a larger sample in the matching process did not lead to improved results.

Recent work suggests that learning curves can also be used model-free to speed up a cross-validation procedure (Domhan et al., 2015; Mohr and van Rijn, 2021).

5.5 Pairwise Approach for a Set of Algorithms

The approach described in the previous subsection was incorporated into various other methods that dealt with N algorithms. In this subsection we describe one of them, referred to as the SAM method (Leite and Brazdil, 2010). This method involves the following steps:

1. Select a pair of classification algorithms and determine which one of the two is better using a pairwise method.
2. Repeat this for all pairs of algorithms and generate a partial order (a graph).
3. Process the partial order to identify the best algorithm(s).

The details are given in the following subsections.

Select a pair of algorithms and determine which one is better

This step can be achieved by a cross-validation test followed by a statistical significance test. This method is referred to as CV_{ST} by the authors. This method can be seen as a function which returns +1 (–1) if algorithm a_i achieves significantly better (worse) performance than a_j. The value 0 is returned if this cannot be established.

Repeat for all pairs and generate a partial order

This step involves applying this method to all algorithm pairs and constructing a graph. When using the output of SAM, link $a_i \leftarrow a_j$ is drawn if a_i is significantly better than a_j. An example of a partial ranking of algorithms generated this way is shown in Figure 5.3. In Chapter 2 this type of ranking was referred to as *quasi-linear ranking*. We note that the figure could include various other links that could be obtained by applying the rule of transitivity. They were omitted so as not to overload the figure with too many links.

Identify the best algorithm(s)

This step concerns the analysis of the partially ordered set of items with the aim of identifying the topmost item(s). The authors have identified three different aggregation measures corresponding to three different strategies of aggregating the detailed information:

- W, the sum of all wins (outgoing arrows)
- L, the sum of all losses (incoming arrows), each as a negative sign
- W+L, the sum of wins and losses

Let us examine what happens if we process the example shown in Figure 5.3. The algorithm *MLP*, for instance, is significantly better than three other algorithms, and so the measure $W = 3$. As it is not beaten by any algorithm, $L = 0$. Consequently, *W+L* is 3. The results for this algorithm and all the others used in our example are shown in Table 5.3.
In this example all three aggregation methods identify the algorithms *MLP* and *LogD* as the topmost two algorithms.

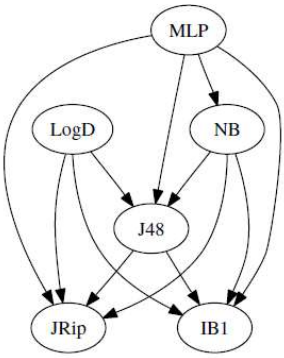

Fig. 5.3: Example of a partial order of six classification algorithms (reproduced from Leite and Brazdil (2010))

Table 5.3: Three different aggregation strategies applied to the example in Fig. 5.3

Algorithm	W	L	W+L
MLP	3	0	3
LogD	3	0	3
NB	3	-1	2
J48	2	-3	-1
JRip	0	-4	-4
IB1	0	-4	-4

Evaluation

Leite and Brazdil (2010) also discussed the problem of evaluating the method described above. We note that the method can, in general, return more than one algorithm. Let us use $\hat{\mathbf{A}}$ to refer to this set. In the example in Figure 5.3 the method returned two algorithms. This may not coincide with the set $\mathbf{A}$ that is considered "correct". So a question arises as to how we should proceed to calculate the appropriate measure of success, M_S.

The method of Leite and Brazdil (2010) is based on an assumption that the items (algorithms) in $\hat{\mathbf{A}}$ could be considered equivalent. Consequently, as soon as $\mathbf{A}$ includes at least one item of $\hat{\mathbf{A}}$, M_S returns 1. This is captured in the following equation:

$$M_S = \frac{|\hat{\mathbf{A}} \cap \mathbf{A}|}{|\hat{\mathbf{A}}|}. \tag{5.9}$$

So, for instance, in a situation when $\hat{\mathbf{A}} = \{a_1\}$ and $\mathbf{A} = \{a_1, a_2\}$, M_S returns 1. Should $\hat{\mathbf{A}}$ return several items, some of which are correct while others are not, M_S returns a

value corresponding to the proportion of correctly identified items in the set of correct items. So, for instance, in a situation when $\hat{\mathbf{A}} = \{a_1, a_2\}$ and $\mathbf{A} = \{a_1\}$, M_S returns 0.5. That is, if we chose one of the items from $\hat{\mathbf{A}}$ at random, the probability of guessing right would be 0.5.

Disadvantages of this approach

One disadvantage of this approach is that the number of pairwise models needed is $(N \times (N-1))/2$, where N represents the number of algorithms. When N is 20, the total number of models is still manageable, i.e., 190. The method does not scale up well for larger numbers.

This problem can be mitigated by identifying a subset of algorithm, N_P as *pivot algorithms*. Then the N algorithms are compared with the pivot algorithms N_P. A study determining whether this could lead to improvements in terms of accuracy and time could be carried out in the future.

As is shown later in Section 5.6, this problem can be mitigated by adopting a search method that identifies the best possible pairwise test in each step. This way, many pairwise tests are actually skipped.

Using a partial ranking for top-n execution

Partial rankings can be used to schedule tests, leading to the top-n execution discussed in Chapter 2. Conceptually, we can view this as an activity that can be divided into two phases. In the first phase, the algorithms are characterized by one of the possible measures, such as *W+L* discussed earlier, and reordering all algorithms according to this measure. The second phase involves following this reordered set using the top-n execution (see Chapter 2).

Extending the average ranking method to partial rankings

The average ranking method discussed in Chapter 2 exploited a set of rankings, one per dataset, and constructed average ranking. A question arises as to how this could be extended to accept a set of partial rankings.

One possibility involves first annotating each item in each partial ranking with the *W+L* measure, and then calculating an average value for each item (algorithm).

5.6 Iterative Approach of Conducting Pairwise Tests

In Section 5.5 we discussed an approach that exploits pairwise tests to identify the potentially best algorithm in a set. In this approach, the pairwise tests were conducted in a kind of preprocessing stage. One disadvantage of this approach is that not all pairwise tests may be needed.

In a follow-up work (van Rijn et al., 2015; van Rijn, 2016), the pairwise approach was improved in two different aspects. First, this method used a search method with the aim of identifying the potentially best algorithm. In consequence, the pairwise tests are conducted on demand, as the method proceeds. The method has, at any time, a suggestion regarding what is the potentially best algorithm found so far. So time is not

wasted by carrying out useless pairwise tests in a preparatory phase. This aspect is similar to the active testing method discussed further on in Section 5.8.

The second very important improvement is that this method takes into account the runtime of tests. So relatively fast tests which are likely to lead to good-performing algorithms are preferred to the slower ones. The method referred to as *pairwise curve comparison* (PCC) by the authors involves the following steps:

1. Initialize the current best algorithm and elaborate partial learning curves (one per algorithm) on the target dataset;
2. Search for the best candidate algorithm to perform a test on;
   Conduct the selected test and update the current best algorithm;
3. Repeat the above until the termination condition has been satisfied.

More details about each step are given below.

Initialize the current best algorithm

This method used the concept of *current best* algorithm, a_{best}.[4] The authors have proposed to initialize it using a randomly chosen algorithm. However, it is not difficult to envisage an alternative approach that would choose some good candidate to initialize a_{best}. One such good candidate is the top element in the A3R-based average ranking discussed in Chapter 2.

After the algorithm a_{best} has been selected, it is necessary to elaborate partial learning curves on the target dataset. This involves conducting tests on different samples of the target dataset. Note that, so far, no tests have been executed on the target dataset.

Search for the best pairwise test

This step considers all possible candidate algorithms, one by one. Each of these represents a potential candidate a_{comp} (competitor) that can replace the current best algorithm, a_{best}. In order to determine whether the replacement should go ahead, it is necessary to predict its performance in relation to a_{best}.

The method uses k datasets with the most similar curves to both a_{comp} and a_{best} to do this. To determine which k datasets are most similar, Eq. 5.7 is used. After the learning curves have been retrieved, they are subject to *curve adaptation*, discussed earlier in Section 5.4.2.

The decision regarding whether algorithm a_{comp} should replace a_{best} depends on the predictions on k datasets. If a_{comp} was predominantly better on the full learning curve on these datasets than a_{best}, the replacement goes ahead.

This method can output a ranking to be used later. Once the comparison against all other algorithms has been carried out, the algorithm that is at that moment a_{best} is added to the ranking. At that point, it can be cross-validated. Note that this PCC can result in an *unstable ranking*: the resulting ranking can differ somewhat based on which algorithm is initialized as a_{best}.

[4] In some papers this algorithm is called the *incumbent* algorithm (the holder of a position). Note that the name a_{best} does not mean that is the best candidate, but rather the current best option at some point in time.

Upgrade to incorporate accuracy and time

As the authors have shown, the method can be upgraded quite easily to incorporate time into the method. The performance comparisons between a_{comp} and a_{best} can use a combined measure of accuracy and time, $A3R'$, which is a simplified version of $A3R$ discussed in Chapter 2:

$$A3R'^{d_i}_{a_j} = \frac{P^{d_i}_{a_j}}{(T^{d_i}_{a_j})^Q}, \tag{5.10}$$

where P represents performance (e.g., accuracy), T runtime, and Q is a scaling factor that controls the importance of runtime.

Main findings

This work confirmed that it is indeed useful to consider both accuracy and time as a measure of performance, when the aim is to obtain a ranking of classifiers. This strategy that gives preference to faster and relatively good methods earlier pays off. A similar conclusion was also reached in conjunction with two other similar approaches, AR*, discussed in Chapter 2, and AT*, discussed further on in Section 5.8.

The PCC method dominates two other approaches that were used in the comparative study. One of them was the *average ranking method* (AR), discussed in Chapter 2. The other method was the baseline method, referred to as *best on sample* (BoS). Quite surprisingly, method BoS is very competitive, if the evaluation is done with loss curves that portray how performance (accuracy) depends on the number of tests. The advantage of PCC becomes apparent when the evaluation is done with loss curves that portray how performance depends on runtime. This is, of course, what interests users. Their aim is to identify the potentially best algorithm as soon as possible, and they normally have an allocated time budget for a given task.

Relationship to surrogate models

The area of automatic hyperparameter configuration discussed in Chapter 6 introduces various useful concepts. One of these is the concept of *surrogate models*, which represent simplified algorithms that provide estimates of the predictions of the actual algorithms but are much faster to execute. An empirical performance model (EPM) is an example of such a surrogate model: instead of running a configuration on a dataset, the empirical performance model can assess whether the configuration is worth trying out. The prediction of an empirical performance model is typically cheaper in terms of runtime than running the actual configuration on a dataset.

The method discussed in this section that predicts the performance of a particular algorithm on the basis of a pair of learning curves (partial learning curve on the target dataset and a full learning curve on a similar dataset) can be related to surrogate models, in the sense that these represent simplified performance models of the underlying algorithm.

Another useful concept used in the area of hyperparameter configuration is the concept of *acquisition function*. This function selects the potentially best parameter configuration to test. We note that the method described in this section incorporates a method for selecting the next pair of algorithms to test. This method can be related to the acquisition function used in the area of hyperparameter configuration.

5.7 Using ART Trees and Forests

The approach of Sun and Pfahringer (2013) involves two phases:

1. Construct a set of pairwise models and generate features;
2. Use *approximate ranking trees* (ART) forests in conjunction with new features to generate predictions.

More details about each phase are given in the following subsections.

Construct a set of pairwise models

In this phase, a set of pairwise metalearning models is generated using a rule-based classifier (RIPPER) (Cohen, 1995). Each model is trained to predict whether to use algorithm a_i in preference to a_j or vice versa.

One interesting aspect of this work is that each pairwise model can use a specific set of base-level features. These may include statistical, information-theoretic, and landmarking features, such as AUC associated with a particular type of tree (REPTree.depth2).

The binary classifiers are then invoked to generate new metafeatures. In one of the variants of the method, each rule associated with a binary model gives rise to one new feature. These are appended to the more traditional features, including, besides some statistical and information-theoretic features, also landmarkers.

Use ART forests to generate predictions

In this phase, an *ART forest* is used to generate a prediction. ART forest can be viewed as a random forest of ART trees (*approximate ranking trees*). ART trees are based on the notion of predictive *clustering trees* for ranking (Blockeel et al., 1998) discussed earlier (see Subsection 5.2.4).

The authors show that this approach achieves better predictive performance results than the classical *k*-NN approach. They also demonstrate that the addition of performance-based features associated with binary classifiers enhances the performance.

The disadvantage of this approach is as follows. As the method requires that all pairwise models be constructed beforehand, the method does not scale up well when the number of algorithms is large. One possible way to resolve this problem is by devising a method that determines the best possible pairwise test to carry out in each step. This method is discussed in the next section.

5.8 Active Testing

The algorithm selection methods, based on average ranking discussed in Chapter 2 suffer from one shortcoming. This is due to the fact that the schedule of tests is fixed. This has the disadvantage that, whenever a given set of algorithms includes many similar variants with similar performance, these could appear close to each other in the ranking. It does not make much sense to test them all in succession. It seems beneficial to try to use algorithms of different types that could hopefully yield better results. The average

ranking method, however, just follows the ranking, and hence is unable to skip very similar algorithms.

This problem is quite common, as similar variants can arise for instance when considering algorithms that include parameters, which may be set to different values. Quite often, many of the settings have a limited impact on performance. Even if we selected only some of all possible alternative parameter settings, we can end up with a large number of variants. Many of them exhibit rather similar performance. Clearly, it is desirable to have a more intelligent way of choosing algorithms from a given ranking. In other words, it is desirable to have a more intelligent schedule of tests. One method that does that is referred to as *active testing* (Leite et al., 2012; Abdulrahman et al., 2018) and is described in the next section. Two variants of the active testing method have been described in literature. The version AT of Leite et al. (2012) uses accuracy as the performance measure. The version AT*, discussed by Abdulrahman et al. (2018), uses a combined measure of accuracy and runtime as the performance measure. As AT can be regarded as a special case of AT*, where the effect of runtime is ignored, the focus here is on AT*, which is more general.

5.8.1 Active testing that considers accuracy and runtime

This active testing method involves two important concepts, *current best algorithm*, a_{best}, and *best competitor*, a_c. The method starts by initializing a_{best} to the topmost algorithm in the average ranking AR*. This algorithm can be run on the target dataset without problems.

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}. In the work of Abdulrahman et al. (2018), the estimate of performance gain, ΔP, was expressed in terms of A3R as follows:

$$\Delta P(a_j, a_{best}, d_i) = max(A3R^{d_i}_{a_{best},a_j} - 1, 0), \tag{5.11}$$

where $A3R$ is defined as

$$A3R^{d_i}_{a_{ref},a_j} = \frac{P^{d_i}_{a_j}/P^{d_i}_{a_{ref}}}{(T^{d_i}_{a_j}/T^{d_i}_{a_{ref}})^Q}, \tag{5.12}$$

where $P^{d_i}_{a_j}$ represents the performance (e.g., accuracy) of algorithm a_j on dataset d_i, and $T^{d_i}_{a_j}$ the corresponding runtime. The term a_{ref} is a reference algorithm and Q is a scaling factor that controls the importance of runtime.[5]

In one recent work Leite and Brazdil (2021) have shown that the following definition of ΔP, which is simpler than Eq. 5.11 shown before leads to better results:

$$\Delta P(a_j, a_{best}, d_i) = A3R^{d_i}_{a_{best},a_j}. \tag{5.13}$$

An illustrative example is shown in Fig. 5.4, which shows different values of ΔP for one competitor of a_{best} on all datasets. If another algorithm outperformed a_{best} on some datasets, it is regarded as a competitor. One important question here is to select the potentially best competitor. This is done by selecting the algorithm that maximizes the estimate of performance gain, as expressed by the following equation:

[5]The earlier variant of Leite et al. (2012) used just accuracy, without considering runtime. In this work, ΔP was defined as a difference, rather than a ratio.

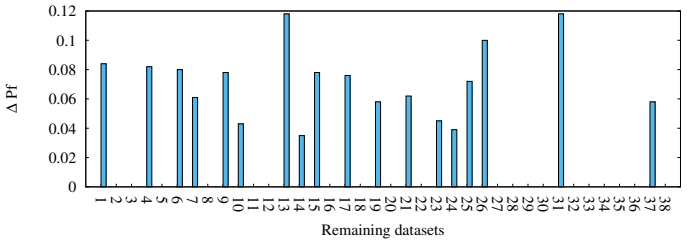

Fig. 5.4: Values of ΔP of a potential competitor with respect to a_{best} on all datasets (reproduced from (Abdulrahman et al., 2018)).

Require: Target dataset d_{new}; Top algorithm in AR* a_{best}; Datasets D_s;
Set of algorithms **A**; Parameter of importance of runtime Q

Initialize the loss curve $L_i \leftarrow ()$

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

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

  Compare the performance of a_c with a_{best} and carry out updates:
  if $P_{a_c}^{d_{new}} > P_{a_{best}}^{d_{new}}$ **then**
    $a_{best} \leftarrow a_c,\ P_{a_{best}}^{d_{new}} \leftarrow P_{a_c}^{d_{new}},\ T_{a_{best}}^{d_{new}} \leftarrow T_{a_c}^{d_{new}}$
  end if
end while
return Loss–time curve L_i and a_{best}

Algorithm 5.1: AT-A3R: active testing with A3R for dataset d_{new}

$$a_c = \underset{a_k}{\operatorname{argmax}} \sum_{d_i \in D_s} \Delta P(a_k, a_{best}, d_i). \tag{5.14}$$

After the best competitor has been identified, the method proceeds with a test on the new dataset to obtain the actual performance of the best competitor. The aim is to determine whether the best competitor achieves a better performance than the current best algorithm, a_{best}. If it does, it becomes the new a_{best}. The reader can consult Algorithm 5.1 for a detailed description of the method (adapted from (Abdulrahman et al., 2018)).

In order to use AT-A3R in practice, it is necessary to determine a good setting of the parameter Q in A3R. The authors have experimented with different values and recommend the setting $Q = 1/16$. It appears that the setting in the range from $P = 1$ to $Q = 1/64$ does not affect the results much.

However, the setting of $Q = 0$, which is outside this range and represents a situation when only accuracy matters, makes a difference. This is illustrated in Figure 5.5 (reproduced from the work referred to above). The optimized version of AT is referred to as AT*. The version AT0 uses the $P = 0$ setting and represents the version that does not take runtime into account. We note that AT0 has far worse performance than AT*. The similar loss is achieved at times much later. We note that the speed-up is huge: approximately two orders of magnitude!

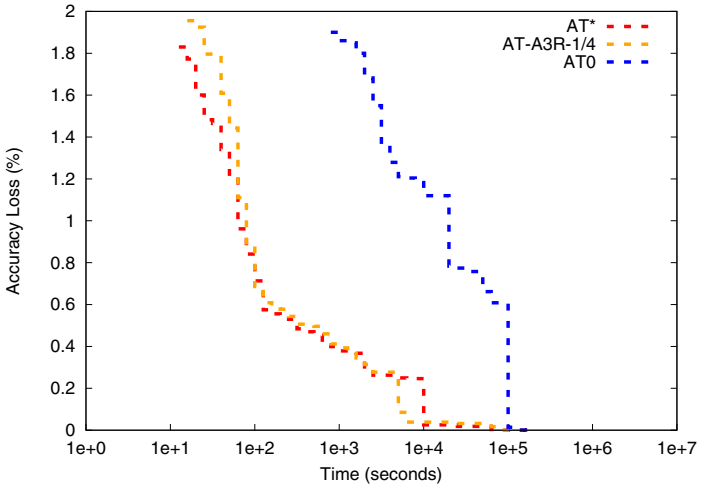

Fig. 5.5: Mean loss–time curves for different variants of AT-A3R

Expected performance gains and relative landmarks

In previous work by Leite et al. (2012) the estimate of performance gain, ΔP was referred to as the *relative landmark, RL*. The former term seems clearer than the second one, and this is why it has been adopted in this chapter. However, the relationship between the two concepts is interesting. All different dataset characteristics, including relative landmarkers, are discussed in Chapter 4.

5.8.2 Active testing with focus on similar datasets

In the work of Leite et al. (2012), the performance gain was estimated by finding the *most similar datasets* and calculating the performance gain only for those datasets. For

the sake of simplicity of exposition, Algorithm 5.1 used all datasets without focusing on the most similar ones. However, it is not difficult to alter the method to focus only on relevant datasets. Let us examine some possibilities.

One possibility involves identifying similar datasets prior to invoking the AT algorithm (Algorithm 5.1). So the identification of similar datasets could be seen as a kind of preprocessing step of datasets. The similarity could be determined using a subset of dataset measures discussed in Chapter 4. These would necessarily have to exclude performance-based measures. This is because, if this was done prior to running any experiments on the target dataset, no performance-based data would be available. So, this approach has the disadvantage that it excludes the use of performance-based measures.

If we want to use performance-based measures, we need to incorporate the selection mechanism inside the AT algorithm. The best alternative seems to do this while searching for the algorithm (workflow) with the largest performance gain. This is captured by Equation 5.15.

$$a_c = \underset{a_k \in A_s}{\operatorname{argmax}} \sum_{d_i \in D_s} \Delta P(a_k, a_{best}, d_i) * Sim(d_{new}, d_i), \tag{5.15}$$

where A_s is a given portfolio of algorithms (workflows) and $Sim(d_{new}, d_i)$ is an appropriate measure of similarity between the dataset d_{new} and dataset d_i. Various alternatives arise regarding how the similarity is established:

1. Similarity based on polarity of performance difference
2. Cosine-based similarity of performance results
3. Correlation-based similarity of performance results

More details about the first measure in the list above (similarity based on polarity of performance difference) are given in the next subsection. Further details about the other two measures are given in Chapter 4 (Section 4.10).

Similarity based on polarity of performance difference

Leite et al. (2012) have described a variant of active testing referred to as *AT1*, where the similarity was based on performance (accuracy) difference. Suppose that, at some stage, a particular algorithm was identified as the current best candidate, a_{best^-}. Suppose that later on other algorithms were examined and algorithm a_{best} was identified as the best one. This implies that $\Delta P(a_{best}, a_{best^-}, d_{new}) > 0$, where ΔP represents the difference of accuracy. Then, in the subsequent iteration, all prior datasets d_k satisfying a similar condition are considered similar to d_{new}. So, the similarity can be defined by

$$\begin{array}{l} Sim_{pol}(d_{new}, d_k) = \\ \Delta P(a_{best}, a_{best^-}, d_{new}) > 0 \;\&\; \Delta P(a_{best}, a_{best^-}, d_k) > 0. \end{array} \tag{5.16}$$

We note that this similarity takes into account the performance of two algorithms only. Despite this limitation, the authors reported a speed-up of 2 with respect to the basic version of AT (Leite et al., 2012).

5.8.3 Discussion

Identifying best choice to test with acquisition functions

The process of identifying the best candidate algorithm to test in accordance with Eq. 5.11 can be compared to what *acquisition functions* do. In Bayesian optimization, the

acquisition function evaluates hyperparameter configurations based on their *expected utility* and selects the one with the highest utility. This issue is discussed in depth in Chapter 6.

Using the AT method for workflow selection and configuration

In many practical applications it is not sufficient to focus on a selection of a single algorithm, but rather construct a workflow (pipeline) of operation. Although this issue is discussed in Chapter 7 (Section 7.4), in this section we give a brief overview of the work of Ferreira and Brazdil (2018), who have examined whether the AT method discussed here could be extended to recommendation of workflows for text classification. Their meta-database included nearly 20,000 workflows. They have shown that the active testing approach somewhat surpassed the average ranking. They also used meta-level analysis, whose aim was to determine importance of different elements of the workflows.

Relationship of AT to reduction of configuration spaces

The AT method searches for the most competitive algorithms (workflows) in a given configuration space. Many non-competitive algorithms may not even be tried out. These algorithms are eliminated at run-time, as a side-effect of the AT search method.

This approach can be contrasted to another one, whose aim is to reduce the configuration space by eliminating non-competitive algorithms in a kind of preprocessing stage. That is, this operation can be carried out before invoking the AT method, or any other search method whose aim is to identify the best algorithm for the target dataset. This issue is addressed and discussed in detail in Chapter 8.

5.9 Non-propositional Approaches

The algorithms discussed so far are only able to deal with propositional representations of the metalearning problem. That is, they assume each meta-example is described by a fixed set of metafeatures, $\mathbf{x} = (x_1, x_2, ..., x_k)$. However, the problem is non-propositional. On the one hand, the size of the set of dataset characteristics varies for different datasets (e.g., depending on the number of features). On the other hand, information about the algorithms can also be useful for metalearning (e.g., the interpretability of the generated models). In spite of this, there are very few approaches that use relational learning approaches.

One approach that exploits the non-propositional dataset description is FOIL, a well-known inductive logic programming (ILP) algorithm (Quinlan and Cameron-Jones, 1993). With FOIL, models can be induced that contain existentially quantified rules, such as "*CN2* is applicable to datasets that contain a discrete feature with more than 2.3% missing values" (Todorovski and Džeroski, 1999).

A different approach uses a case-based reasoning tool, *CBR-Works Professional* (Lindner and Studer, 1999; Hilario and Kalousis, 2001). This can be viewed as a *k*-NN algorithm that not only allows a non-propositional description of datasets but also enables the use of information about the algorithms, independently of datasets. This work was extended later by the analysis of different distance measures for non-propositional representation (Kalousis and Hilario, 2003). Some of these measures enable the distance

between two datasets to be defined by a pair of individual features, e.g., the two features which are most similar in terms of one property such as *skewness*.

These papers usually compare their approaches against propositional methods. However, we have no knowledge of a comparison between different non-propositional methods.

References

Abdulrahman, S., Brazdil, P., van Rijn, J. N., and Vanschoren, J. (2018). Speeding up algorithm selection using average ranking and active testing by introducing runtime. *Machine Learning*, 107(1):79–108.

Bensusan, H. and Giraud-Carrier, C. (2000). Discovering task neighbourhoods through landmark learning performances. In Zighed, D. A., Komorowski, J., and Zytkow, J., editors, *Proceedings of the Fourth European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD 2000)*, pages 325–330. Springer.

Bensusan, H. and Kalousis, A. (2001). Estimating the predictive accuracy of a classifier. In Flach, P. and De Raedt, L., editors, *Proceedings of the 12th European Conference on Machine Learning*, pages 25–36. Springer.

Blockeel, H., De Raedt, L., and Ramon, J. (1998). Top-down induction of clustering trees. In *Proceedings of the 15th International Conference on Machine Learning*, ICML'98, pages 55–63, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Brazdil, P., Gama, J., and Henery, B. (1994). Characterizing the applicability of classification algorithms using meta-level learning. In Bergadano, F. and De Raedt, L., editors, *Proceedings of the European Conference on Machine Learning (ECML94)*, pages 83–102. Springer-Verlag.

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

Cohen, W. W. (1995). Fast effective rule induction. In Prieditis, A. and Russell, S., editors, *Proceedings of the 12th International Conference on Machine Learning*, ICML'95, pages 115–123. Morgan Kaufmann.

Dimitriadou, E., Hornik, K., Leisch, F., Meyer, D., and Weingessel, A. (2004). e1071: Misc functions of the Department of Statistics (e1071), R package version 1.5-1. Technical report, TU Wien.

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

Eggensperger, K., Lindauer, M., Hoos, H., Hutter, F., and Leyton-Brown, K. (2018). Efficient benchmarking of algorithm configuration procedures via model-based surrogates. *Special Issue on Metalearning and Algorithm Selection, Machine Learning*, 107(1).

Ferreira, M. and Brazdil, P. (2018). Workflow recommendation for text classification with active testing method. In *Workshop AutoML 2018 @ ICML/IJCAI-ECAI*. Available at site https://sites.google.com/site/automl2018icml/accepted-papers.

Feurer, M., Eggensperger, K., Falkner, S., Lindauer, M., and Hutter, F. (2018). Practical automated machine learning for the AutoML challenge 2018. In *International Workshop on Automatic Machine Learning at ICML2018*, pages 1189–1232.

Feurer, M., Springenberg, J. T., and Hutter, F. (2014). Using meta-learning to initialize Bayesian optimization of hyperparameters. In *ECAI Workshop on Metalearning and Algorithm Selection (MetaSel)*, pages 3–10.

Fürnkranz, J. and Petrak, J. (2001). An evaluation of landmarking variants. In Giraud-Carrier, C., Lavrač, N., and Moyle, S., editors, *Working Notes of the ECML/PKDD 2000 Workshop on Integrating Aspects of Data Mining, Decision Support and Meta-Learning*, pages 57–68.

Gama, J. and Brazdil, P. (1995). Characterization of classification algorithms. In Pinto-Ferreira, C. and Mamede, N. J., editors, *Progress in Artificial Intelligence, Proceedings of the Seventh Portuguese Conference on Artificial Intelligence*, pages 189–200. Springer-Verlag.

Hilario, M. and Kalousis, A. (2001). Fusion of meta-knowledge and meta-data for case-based model selection. In Siebes, A. and De Raedt, L., editors, *Proceedings of the Fifth European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD01)*. Springer.

Hutter, F., Xu, L., Hoos, H., and Leyton-Brown, K. (2014). Algorithm runtime prediction: Methods and evaluation. *Artificial Intelligence*, 206:79–111.

Jamieson, K. and Talwalkar, A. (2016). Non-stochastic best arm identification and hyperparameter optimization. In *Artificial Intelligence and Statistics*, pages 240–248.

Kalousis, A. (2002). *Algorithm Selection via Meta-Learning*. PhD thesis, University of Geneva, Department of Computer Science.

Kalousis, A. and Hilario, M. (2000). Model selection via meta-learning: A comparative study. In *Proceedings of the 12th International IEEE Conference on Tools with AI*. IEEE Press.

Kalousis, A. and Hilario, M. (2003). Representational issues in meta-learning. In *Proceedings of the 20th International Conference on Machine Learning*, ICML'03, pages 313–320.

Kolodner, J. (1993). *Case-Based Reasoning*. Morgan Kaufmann Publishers.

Köpf, C., Taylor, C., and Keller, J. (2000). Meta-analysis: From data characterization for meta-learning to meta-regression. In Brazdil, P. and Jorge, A., editors, *Proceedings of the PKDD 2000 Workshop on Data Mining, Decision Support, Meta-Learning and ILP: Forum for Practical Problem Presentation and Prospective Solutions*, pages 15–26.

Leake, D. B. (1996). *Case-Based Reasoning: Experiences, Lessons & Future Directions*. AAAI Press.

Leite, R. and Brazdil, P. (2004). Improving progressive sampling via meta-learning on learning curves. In Boulicaut, J.-F., Esposito, F., Giannotti, F., and Pedreschi, D., editors, *Proc. of the 15th European Conf. on Machine Learning (ECML2004)*, LNAI 3201, pages 250–261. Springer-Verlag.

Leite, R. and Brazdil, P. (2005). Predicting relative performance of classifiers from samples. In *Proceedings of the 22nd International Conference on Machine Learning*, ICML'05, pages 497–503, NY, USA. ACM Press.

Leite, R. and Brazdil, P. (2007). An iterative process for building learning curves and predicting relative performance of classifiers. In *Proceedings of the 13th Portuguese Conference on Artificial Intelligence (EPIA 2007)*, pages 87–98.

Leite, R. and Brazdil, P. (2010). Active testing strategy to predict the best classification algorithm via sampling and metalearning. In *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI)*, pages 309–314.

Leite, R. and Brazdil, P. (2021). Exploiting performance-based similarity between datasets in metalearning. In Guyon, I., van Rijn, J. N., Treguer, S., and Vanschoren, J., editors, *AAAI Workshop on Meta-Learning and MetaDL Challenge*, volume 140, pages 90–99. PMLR.

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

Leyton-Brown, K., Nudelman, E., and Shoham, Y. (2009). Empirical hardness models: Methodology and a case study on combinatorial auctions. *Journal of the ACM*, 56(4).

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

Lindner, G. and Studer, R. (1999). AST: Support for algorithm selection with a CBR approach. In Giraud-Carrier, C. and Pfahringer, B., editors, *Recent Advances in Meta-Learning and Future Work*, pages 38–47. J. Stefan Institute.

Mohr, F. and van Rijn, J. N. (2021). Towards model selection using learning curve cross-validation. In *8th ICML Workshop on Automated Machine Learning (AutoML)*.

Pfahringer, B., Bensusan, H., and Giraud-Carrier, C. (2000). Meta-learning by landmarking various learning algorithms. In Langley, P., editor, *Proceedings of the 17th International Conference on Machine Learning*, ICML'00, pages 743–750.

Pfisterer, F., van Rijn, J. N., Probst, P., Müller, A., and Bischl, B. (2018). Learning multiple defaults for machine learning algorithms. *arXiv preprint arXiv:1811.09409*.

Provost, F., Jensen, D., and Oates, T. (1999). Efficient progressive sampling. In Chaudhuri, S. and Madigan, D., editors, *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM.

Quinlan, J. (1992). Learning with continuous classes. In Adams and Sterling, editors, *AI'92*, pages 343–348. Singapore: World Scientific.

Quinlan, R. (1998). *C5.0: An Informal Tutorial*. RuleQuest. `http://www.rulequest.com/see5-unix.html`.

Quinlan, R. and Cameron-Jones, R. (1993). FOIL: A midterm report. In Brazdil, P., editor, *Proc. of the Sixth European Conf. on Machine Learning*, volume 667 of *LNAI*, pages 3–20. Springer-Verlag.

Rasmussen, C. and Williams, C. (2006). *Gaussian Processes for Machine Learning*. The MIT Press.

Soares, C., Petrak, J., and Brazdil, P. (2001). Sampling-based relative landmarks: Systematically test-driving algorithms before choosing. In Brazdil, P. and Jorge, A., editors, *Proceedings of the 10th Portuguese Conference on Artificial Intelligence (EPIA2001)*, pages 88–94. Springer.

Sohn, S. Y. (1999). Meta analysis of classification algorithms for pattern recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(11):1137–1144.

Sun, Q. and Pfahringer, B. (2012). Bagging ensemble selection for regression. In *Proceedings of the 25th Australasian Joint Conference on Artificial Intelligence*, pages 695–706.

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

Thornton, C., Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2013). Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 847–855. ACM.

Todorovski, L., Blockeel, H., and Džeroski, S. (2002). Ranking with predictive clustering trees. In Elomaa, T., Mannila, H., and Toivonen, H., editors, *Proc. of the 13th European Conf. on Machine Learning*, number 2430 in LNAI, pages 444–455. Springer-Verlag.

Todorovski, L. and Džeroski, S. (1999). Experiments in meta-level learning with ILP. In Rauch, J. and Zytkow, J., editors, *Proceedings of the Third European Conference on*

Principles and Practice of Knowledge Discovery in Databases (PKDD99), pages 98–106. Springer.

Tsuda, K., Rätsch, G., Mika, S., and Müller, K. (2001). Learning to predict the leave-one-out error of kernel based classifiers. In *ICANN*, pages 331–338. Springer-Verlag.

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

van Rijn, J. N., Abdulrahman, S., Brazdil, P., and Vanschoren, J. (2015). Fast algorithm selection using learning curves. In *International Symposium on Intelligent Data Analysis XIV*, pages 298–309.