2
Metalearning Approaches for Algorithm Selection I (Exploiting Rankings)

Summary. This chapter discusses an approach to the problem of algorithm selection, which exploits the performance metadata of algorithms (workflows) on prior tasks to generate recommendations for a given target dataset. The recommendations are in the form of rankings of candidate algorithms. The methodology involves two phases. In the first one, rankings of algorithms/workflows are elaborated on the basis of historical performance data on different datasets. These are subsequently aggregated into a single ranking (e.g. average ranking). In the second phase, the average ranking is used to schedule tests on the target dataset with the objective of identifying the best performing algorithm. This approach requires that an appropriate evaluation measure, such as *accuracy*, is set beforehand. In this chapter we also describe a method that builds this ranking based on a combination of accuracy and runtime, yielding good anytime performance. While this approach is rather simple, it can still provide good recommendations to the user. Although the examples in this chapter are from the classification domain, this approach can be applied to other tasks besides algorithm selection, namely hyperparameter optimization (HPO), as well as the combined algorithm selection and hyperparameter optimization (CASH) problem. As this approach works with discrete data, continuous hyperparameters need to be discretized first.

2.1 Introduction

This chapter follows the basic scheme discussed in the introduction which was illustrated in Figures 1.1 and 1.2. However, we focus on a method that exploits a specific kind of metadata that captures performance results of algorithms (workflows) on past datasets, namely rankings. Ranking methods usually rely on some form of metadata, that is, knowledge about how a discrete set of algorithms have performed on a set of historical datasets. This chapter discusses a standard approach that is capable of converting this metadata into a static ranking. The ranking helps users by suggesting an order of algorithms to apply when confronted with a new dataset. The approach is rather simple, but can still provide excellent recommendations to the user. Therefore, we have decided to discuss this approach before various other approaches described in subsequent chapters. Although it can be applied to various domains, all the examples in this chapter are from the classification domain.

The ranking approach can be applied to the algorithm selection (AS) task, hyperparameter optimization (HPO), as well as the combined algorithm selection and hyperpa-

P. Brazdil et al., *Metalearning*, Cognitive Technologies,
https://doi.org/10.1007/978-3-030-67024-5_2

rameter optimization (CASH) problem. Note that this approach works always with discrete data. Therefore, when addressing the HPO or CASH problems using this approach, continuous hyperparameters need to be discretized first.

Organization of this chapter

Section 2.2 discusses a rather general topic, which is concerned with different forms of recommendation. The system can recommend just a single item, or several items, or a ranked list of items.

Section 2.3 explains the methodology used to construct a ranking of algorithms, based on the available metadata. The methodology involves two phases. In the first one, rankings of algorithms/workflows are elaborated on the basis of historical performance data on different datasets. These are subsequently aggregated into a single ranking (e.g. average ranking). The details are described in Subsection 2.3.1. In the second phase, the average ranking is used to schedule tests on the target dataset with the objective of identifying the best performing algorithm. The details are explained in Subsection 2.3.2. This procedure represents a kind of standard and has been used in many metalearning and AutoML research papers.

Section 2.4 describes a method that incorporates a measure that combines both accuracy and runtime. Indeed, as the aim is to identify well-performing algorithms as soon as possible, this method tests fast algorithms first, and proceeds to slower ones later. The final section (2.5) describes various extensions of the basic approach.

2.2 Different Forms of Recommendation

Before explaining the approach that exploits rankings, let us analyze different types of recommendation that a system can provide. The system can recommend the user to apply/explore:

1. Best algorithm in a set,
2. Subset of the top algorithms,
3. Linear ranking,
4. Quasi-linear (weak) ranking,
5. Incomplete ranking.

Although *complete* and *incomplete* rankings can also be referred to as *total* and *partial* rankings, we prefer to use the former terminology here. Table 2.1 illustrates what characterizes each case. In our example, it is assumed that the given portfolio of algorithms includes $\{a_1, a_2, \ldots, a_6\}$. Figure 2.1 complements this information. Hasse diagrams provide a simple visual representation of rankings (Pavan and Todeschini, 2004), where each node represents an algorithm and directed edges represent the relation "significantly better than". The figure on the left (part (a)) shows an example of a complete linear ranking. The figure in the center (part (b)) shows an example of complete quasi-linear ranking. The figure on the right (part (c)) shows an example of an incomplete linear ranking. Each of these figures corresponds to the rankings in rows 3 to 5 of Table 2.1. More details about each form are provided in the following subsections.

Table 2.1: Examples of different forms of recommendation

1. Best in a set			a_3			
2. Subset			$\{a_3, a_1, a_5\}$			
	Rank					
	1	2	3	4	5	6
3. Linear and complete ranking	a_3	a_1	a_5	a_6	a_4	a_2
4. Quasi-linear and complete ranking	$\overline{a_3 \; a_1}$		a_5	$\overline{a_6 \; a_4}$		a_2
5. Linear and incomplete ranking		a_3	a_1	a_5	a_6	

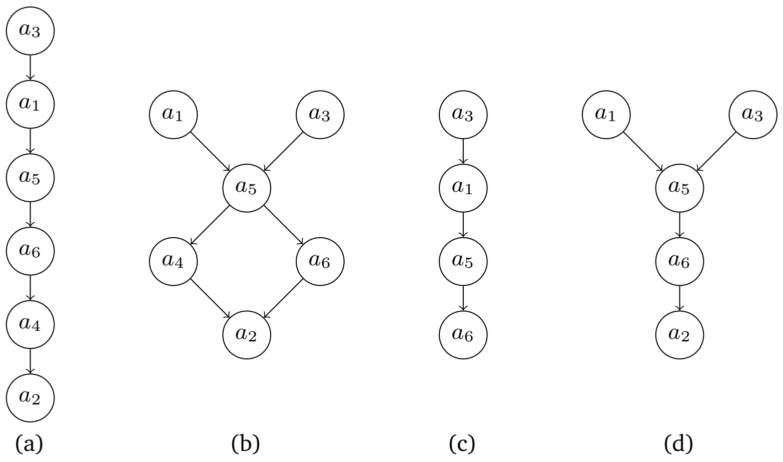

Fig. 2.1: Representation of rankings using Hasse diagrams: (a) complete linear ranking; (b) complete quasi-linear ranking; (c) incomplete linear ranking; (d) incomplete quasi-linear ranking

2.2.1 Best algorithm in a set

The first form consists of identifying the algorithm that is expected to obtain the best performance in the set of base-level algorithms (Pfahringer et al., 2000; Kalousis, 2002). Note that this is formally a ranking of size one. Intuitively, most data scientists make use of such ranking, when applying their favorite algorithm first on a new dataset. One way of doing this is by identifying the best algorithm for each dataset. Then the information gathered needs to be aggregated. One possibility is to use the algorithm that was the best one on most datasets.

We note that this is similar to selecting one of the top items in the linear ranking. As this strategy does not use search, there is a possibility that the algorithm selected may not be the truly best one. Consequently, we may get a substandard result.

2.2.2 Subset of the top algorithms

Methods that use this form of recommendation suggest a (usually small) subset of algorithms that are expected to perform well on the given problem (Todorovski and Džeroski, 1999; Kalousis and Theoharis, 1999; Kalousis, 2002). One way of determining this subset is by identifying the best algorithm for each dataset. If the best algorithm is tied with others, we may simply select the first one in the order as they appear. Then the information gathered can be aggregated by taking the union of all the algorithms identified. So, supposing that the training datasets include n datasets, we will end up with a subset with at most n elements. We note that this method ignores all algorithms that achieve a comparable performance to the best algorithm. Consequently, there is a chance that the subset selected may not include the truly best one.

To increase the chances, we may use a more elaborate strategy. It involves identifying the best algorithm for each dataset and all other algorithms that also perform equally well. The notion of *performing well* on a given dataset is typically defined in relative terms. For example, having a model that makes predictions that are correct in 50% of the cases is considered very good on some datasets, whereas on others this might be considered mediocre. The following subsection discusses the details.

Identifying algorithms with comparable performance

Assuming that the best algorithm has been identified ($a*$) a question arises whether there are other algorithms (e.g., a_c) with comparable performance. One possibility is to carry out a suitable statistical test to determine whether the performance of some candidate algorithms is significantly worse than the performance of the best one (Kalousis and Theoharis, 1999; Kalousis, 2002). An algorithm is included among the "good" subset, if it is not significantly worse than the best one. In practice, researchers have applied both parametric test (e.g., t-test) and non-parametric tests (e.g., Wilcoxon signed-rank test (Neave and Worthington, 1992)). This approach requires that the tests of the algorithms are carried out using cross-validation (CV), as it requires information gathered in different folds of the CV procedure.

If the fold information is not available, it is possible to use an approximate method that may still provide a quite satisfactory solution. This approach involves establishing a *margin* relative to the performance of the best algorithm on that dataset. All the algorithms with a performance within the margin are considered to perform well too. In classification, the margin can be defined in the following way (Brazdil et al., 1994; Gama and Brazdil, 1995; Todorovski and Džeroski, 1999):

$$\left(e_{min}, e_{min} + k\sqrt{\frac{e_{min}\,(1 - e_{min}))}{n}} \right), \tag{2.1}$$

where e_{min} is the error of the best algorithm, n is the number of examples, and k is a user-defined confidence level that affects the size of the margin. We note that this approach is based on an assumption that errors are normally distributed.

Both approaches are related, because the interval of confidence of the first method can be related to the margin used in the second one. Thus, any algorithm with a performance within this margin can be considered to be not significantly worse than the best one. This method results in a small subset of algorithms for each dataset (those that perform well).

Aggregating subsets

The subsets generated in the previous step need to be aggregated. This can be done, for instance, by taking a union. The algorithms in the final subset can be ordered according to how many datasets they are involved in. If a particular algorithm a_i appears in several subsets, while a_j only once, a_i could be attributed a higher rank than a_j, as the probability that a_i will achieve better performance on the target dataset is higher when compared with a_j.

This approach has the advantage that the search phase involves more than one algorithm and, consequently, the chance that the truly best algorithm is included in it is higher.

This topic is related to the problem of reducing the set of algorithm in a given portfolio, which is discussed in Chapter 8.

2.2.3 Linear ranking

Rankings have been used by many researches in the past (Brazdil et al., 1994; Soares and Brazdil, 2000; Keller et al., 2000; Brazdil et al., 2003). Typically, the order indicated in the ranking is the order that should be followed in the experimentation phase. Many systems tend to use *linear and complete* ranking. It is shown in row 3 of Table 2.1 and also in Figure 2.1(a). It is referred to as *linear ranking* because the ranks are different for all algorithms. Additionally, it is a *complete ranking* because all the algorithms $a_1, \ldots, a_6$ have their rank defined (Cook et al., 2007).

This type of ranking has a disadvantage, as it cannot represent the case when two algorithms are tied on a given dataset (i.e., their performance is not significantly different).

2.2.4 Quasi-linear (weak) ranking

Whenever two or more algorithms are tied, a *quasi-linear* (sometimes also called *weak*) ranking can be used (Cook et al., 1996). An example is shown in Table 2.1 (row 4). The line above the algorithm names (as in $\overline{a_3\ a_1}$) indicates that the performance of the corresponding algorithms is not significantly different. An alternative representation is shown in Figure 2.1(b).

Quasi-linear rankings arise when there is not enough data permitting to distinguish their (relative) performance on the dataset at hand, or if the algorithms are truly indistinguishable. In this case, the problem can be resolved by assigning the same rank to all tied algorithms.

A meta-learning method that provides recommendations in the form of quasi-linear rankings is proposed in Brazdil et al. (2001). The method is an adaptation of the k-NN ranking approach discussed in the next section (2.3). It identifies algorithms with equivalent performance and includes only one of the algorithms in the recommendation.

2.2.5 Incomplete ranking

Both linear and quasi-linear rankings can be incomplete, as only some algorithms were used in the tests. So a question arises on what to do. In our view we need to distinguish the following two rather different situations. The first one arises when some algorithms were excluded from consideration for a particular reason (e.g., they sometimes crash; they are difficult to use; they are rather slow etc.). In this case, we should just resort to the incomplete ranking, as if it were complete.

The second situation occurs when new algorithms were developed and so they need to be added to the existing algorithm set (portfolio). Obviously, it is necessary to run tests to extend the existing metadata. The metadata does not necessarily need to be complete. The topic of complete vs. incomplete metadata is discussed further in Chapter 8 (Section 8.8). If the metalearning method in question can work with incomplete metadata, a question arises regarding which tests should be conducted in preference to others. Chapter 8 (Section 8.9) describes some strategies developed in the area of *multi-armed bandits* that can be used for this purpose.

2.2.6 Searching for the best algorithm within a given budget

Rankings are particularly suitable for algorithm recommendation, because the metalearning system can be developed without any information about how many base-algorithms the user will try out. This number depends on the available computational resources (i.e., budget) and the importance of achieving good performance (e.g., accuracy) on the target problem. If time is the critical factor, only very few alternatives should be selected. On the other hand, if the critical factor is, say, accuracy, then more algorithms should be examined, as it increases the chance of obtaining the potentially best result. This was confirmed by various experimental studies (e.g., Brazdil et al. (2003)).

2.3 Ranking Models for Algorithm Selection

The approach described in this chapter is based on the following assumption: if the aim is to identify a well-performing algorithm, it is not as important to accurately predict their *true performance*, as it is to predict their *relative performance*. The task of algorithm recommendation can thus be defined as the task of ranking algorithms according to their predicted performance.

To address this problem with the help of machine learning, we follow the two-phase approach described in introductory Chapter 1. In the first phase, it is necessary to collect data describing the performance of algorithms, referred to as *performance metadata*. Some approaches also exploit certain characteristics of base-level tasks, referred to as *task/dataset metadata*. The metadata permits to generate a meta-level model. In the approach discussed in this chapter, the meta-level model is in the form of a ranked list of algorithms (workflows). More details about this process are provided in Subsection 2.3.1.

After the meta-level model has been generated, it is possible to advance to the second phase. The meta-level model can be used to obtain recommendation for the target dataset. More details about this are given in Subsection 2.3.2.

2.3.1 Generating a meta-model in the form of rankings

The process of generating a meta-model involves the following steps:

1. Evaluate all algorithms on all datasets.
2. Use dataset similarity to identify the relevant parts of metadata.

3. Use all performance results to elaborate a ranking of all algorithms, representing a meta-model.

This process is illustrated in Figure 2.2.

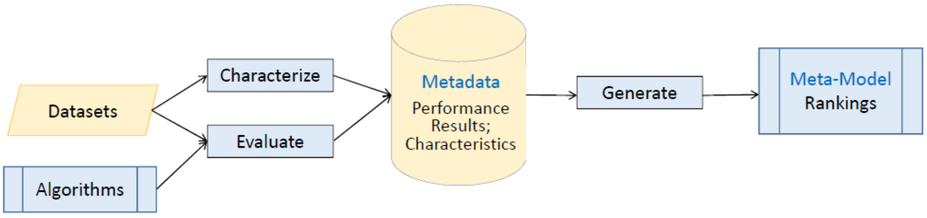

Fig. 2.2: Generating a meta-model in the form of a ranking

Gathering performance results

This step consists of running tests to collect performance results (performance metadata). We assume that the performance results are stored in a *performance matrix* P, where rows represent datasets and columns algorithms. More precisely, the labels (names) of the rows are the names of the dataset used, i.e., $D = \{d_1, \cdots, d_k\}$. Similarly, the labels (names) of the columns are the algorithm names ($A = \{a_1, \cdots, a_n\}$). Each slot $P(i, j)$ holds the performance of algorithm j on dataset i after the respective evaluation was carried out.

Let us clarify what kind of performance measures can be used here. In the classification domain, some common measures are accuracy, AUC, F1, microF1 and macroF1, among others, described in books on machine learning (ML) (e.g., Mitchell (1997); Hand et al. (2001)). In the examples in this chapter we use mainly *predictive accuracy*, which is defined as the proportion of test examples that were classified correctly by the model.

Details of this process are shown in Algorithm 2.1. To simplify the description, we assume that the initial performance matrix P_0, which is initially empty, is given. The aim is to generate a rank matrix R, which has a similar format to the performance matrix P, but instead of performance values it includes ranks. Table 2.3 shows an example of test results converted to ranks for 3 datasets and 10 algorithms.

The conversion to ranks is simple. The best algorithm is assigned rank 1, the runner-up is assigned rank 2, and so on.

Note that cross-validating each algorithm on each dataset is a costly procedure, and only feasible for a relatively small number of algorithms and datasets. In many studies, this information is assumed to be readily available, e.g., by using an existing source of such metadata, such as OpenML discussed in Chapter 16.

```
input : $P_0$ (empty performance matrix)
output: $R$ (ranking matrix)
begin
    $P \leftarrow P_0$
end
foreach row (dataset) $i$ in $P$ do
    foreach column (algorithm) $j$ in $P$ do
        Evaluate the algorithm $j$ on dataset $i$ using cross-validation (CV):
        $P(i, j) \leftarrow CV(j, i)$
    end
end
foreach column (algorithm) $j$ in $P$ do
    Convert the performance vector into a ranking:
    $R(, j) \leftarrow rank(P(, j))$
end
```

Algorithm 2.1: Constructing performance and rank matrices

Aggregating performance results into a single ranking

This subsection includes a description of the process of aggregation of set of rankings obtained in different tests into a single aggregated ranking. The aggregation is done on the basis of a particular ranking criterion that can be chosen. Different criteria exist:

- average rank,
- median rank,
- rank based on significant wins and/or losses.

Aggregating by average rank: This method can be regarded as a variant of Borda's method (Lin, 2010). This method was inspired by Friedman's *M statistic* (Neave and Worthington, 1992). The method based on average ranks is referred to as the *average ranking (AR)* method. It requires that we have, for each dataset, a ranking of all algorithms based on performance results.

Let $R_{i,j}$ be the rank of base-algorithm a_j ($j = 1, \ldots, n$) on dataset i, where n is the number of algorithms. The *average rank* for each a_j is

$$\bar{R}_j = \frac{\sum_{i=1}^{n} R_{i,j}}{k}, \tag{2.2}$$

where k represents the number of datasets. The final ranking is obtained by ordering the average ranks and assigning ranks to the algorithms accordingly. An example is given further on.

Aggregating by median rank: This method is similar to the one just described. Instead of calculating the mean rank using Eq. 2.2, it is necessary to obtain the median value. The method based on median ranks is referred to as the *median ranking (MR)* method. Cachada (2017) compared the two methods — AR and MR — on a set-up that included test results of 368 different workflows on 37 datasets. The results showed that MR achieved somewhat better results than AR, although the differences were not statistically significant.

Table 2.2: Classification algorithms

C5b	Boosted decision trees (C5.0)
C5r	Decision tree-based rule set (C5.0)
C5t	Decision tree (C5.0)
IB1	1-Nearest neighbor (MLC++)
LD	Linear discriminant
Lt	Decision trees with linear combination of attributes
MLP	Multilayer perceptron (Clementine)
NB	Naïve Bayes
RBFN	Radial basis function network (Clementine)
RIP	Rule sets (RIPPER)

Table 2.3: Example of an average ranking based on three datasets

Algorithm:	C5b	C5r	C5t	MLP	RBFN	LD	Lt	IB1	NB	RIP
byzantine	2	6	7	10	9	5	4	1	3	8
isolet	2	5	7	10	9	1	6	4	3	8
pendigits	2	4	6	7	10	8	3	1	9	5
Average rank scores $\bar{R}_i$	2.0	5.0	6.7	9.0	9.3	4.7	4.3	2.0	5.0	7.0
Average ranking	1.5	5.5	7	9	10	4	3	1.5	5.5	8

Aggregating by the number of significant wins and/or losses: This method establishes the rank of each algorithm a_i and takes into account the number of significant wins and/or losses over other algorithms. A *significant win* of algorithm a_i over algorithm a_j is defined as a performance difference that is statistically significant. This method was explored by various researchers in the past (Brazdil et al., 2003; Leite and Brazdil, 2010).

Example: elaborating an average ranking

The use of the average ranking method for the problem of algorithm recommendation is illustrated here on an example. The metadata used captures the performance of 10 classification algorithms (see Table 2.2) and 57 datasets from the UCI repository (Asuncion and Newman, 2007). More information about the experimental set-up can be found elsewhere (Brazdil et al., 2003).

The goal here is to construct the ranking of the algorithms on the basis of rankings obtained on three datasets listed in Table 2.3. The corresponding average rank scores, $\bar{R}_j$, obtained by aggregating the individual rankings are shown in that table. The rank scores can be used to reorder the algorithms and, this way, obtain the recommended ranking (`C5b`, `IB1` .. `RBFN`). This ranking provides guidance concerning the future experiments to be carried out on the target dataset.

We note that the average ranking contains two pairs of ties. One of them involves `C5b` and `IB1`, which share the first two ranks and hence have been attributed rank 1.5 in our table. A tie means that there is no evidence that either of the algorithms (in this case C5b and IB1) would achieve different performance, based on the metadata used. The user can carry our random selection, or else use some other criterion in the selection process (e.g., runtime).

A question that follows is whether the predicted (or recommended) ranking is an accurate prediction of the true ranking, i.e., of the relative performance of the algorithms on the target dataset. This issue is addressed in the next subsection (2.3.2) and also in Chapter 3. We observe that the two rankings are more or less similar. The largest error is made in the prediction of the ranks of `LD` and `NB` (four rank positions), but the majority of the errors are of two positions. Nevertheless, a proper evaluation methodology is necessary. That is, we need methods that enable us to quantify and compare the quality of rankings in a systematic way. This is explained in Section 2.3.3.

2.3.2 Using the ranking meta-model for predictions (top-*n* strategy)

The meta-model discussed in the previous subsection can be used to provide a recommendation regarding which algorithm to select for the target dataset. This scheme is illustrated in Figure 2.3. Algorithm 2.2 provides more details about the method.

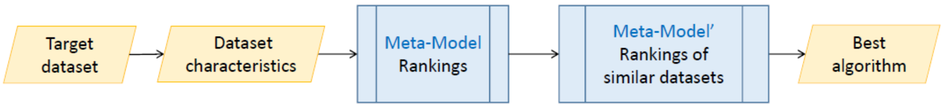

Fig. 2.3: Using the average ranking (AR) method for the prediction of the best algorithm

As the recommendation is in the form of a ranking, it is thus reasonable to expect that the order recommended will be followed by the user. The algorithm (workflow) ranked in the first position will most likely be considered first, followed by the one ranked second, and so on. This is done by cross-validating the algorithms in this order on the target dataset. After each cross-validation test, the performance is stored, and the algorithm with the highest stored performance is the winner. A question arises regarding how many algorithms the user should select.

A *top*-n execution can be used for this purpose (Brazdil et al., 2003). This method consists of simulating that the top n items will be selected. When studying the performance of the top-n scheme, we will normally let it run until the end. In other words, the parameter n will normally be set to the maximum value, corresponding to the number of algorithms. This has the advantage that we can inspect the results at different stages of execution. One other alternative to fixing n consists of fixing the time budget (see Section 2.4).

Example

The method is illustrated by following the recommended ranking presented in Table 2.4 and carrying out tests on `waveform40` dataset. The table also presents the accuracy obtained by each algorithm and the corresponding runtime. The first item listed in this table represents the default classification accuracy on this dataset. As the dataset `waveform40` includes three classes, we can assume that the mean accuracy will be 1/3, under the

```
input  : $A = \{a_1, \cdots, a_n\}$ (list of algorithms ordered by ranks)
         $d_{new}$ (target dataset)
         $n$ (number of algorithms to test)
output: $a*$ (algorithm with the best performance)
         $p*$ (performance of $a*$)
         $t_{accum}$ (time used)
begin
    $a* \leftarrow A[1]$ (initialize $a*$)
    Evaluate the first algorithm and initialize values:
    $(p*, t_{accum}) \leftarrow CV(A[1], d_{new})$
    foreach $i \in \{2, \cdots n\}$ do
        Evaluate the i-th algorithm:
        $(p_c, t_c) \leftarrow CV(A[i], d_{new})$
        if $p_c > p*$ then
            $a* \leftarrow A[i]$
        end
        $p* \leftarrow max(p_c, p*)$
        $t_{accum} \leftarrow t_c + t_{accum}$
    end
end
```

Algorithm 2.2: Top-*n* procedure

Table 2.4: Results of executing a given recommended ranking on `waveform40` dataset

Recommended	Def	MLP	RBFN	LD	Lt	C5b	NB	RIP	C5r	C5t	IB1
Ranking	0	1	2	3	4	5	6	7	8	9	10
Accuracy	0.33	0.81	0.85	0.86	0.84	0.82	0.80	0.79	0.78	0.76	0.70
Runtime	0	99.70	441.52	1.73	9.78	44.91	3.55	66.18	11.44	4.05	34.91
Runtime accum.	0	99.7	541.2	542.9	552.7	597.6	601.2	667.4	678.8	682.9	717.8

assumption that the classes are equally probable. We assume that determining this takes virtually no time.

Figure 2.4 shows how the accuracy evolves with the number of algorithms executed (*n*). The first algorithm executed is `MLP`, obtaining an accuracy of 81.4%. Once the next algorithm in the ranking (`RBFN`) is executed a significant increase in accuracy is obtained, reaching 85.1%. The execution of the next algorithm in the ranking, `LD`, yields a smaller increase (86.0%). The remaining algorithms do not alter this situation much. Note that when using the top-*n* strategy, the performance never goes down. In order to understand this, we need to revisit what it actually does. It measures the highest obtained performance in cross-validation tests so far. As the set of cross-validated algorithms grows, this value cannot decrease.

Figure 2.5 shows the evolution of accuracy on runtime. This plot provides more information that is relevant for the assessment of the recommended ranking. It shows that, although the execution of `RBFN` provides a significant improvement in accuracy, it does so at the cost of a comparatively much larger runtime (441 s.). The plot also shows that, although the gain obtained with `LD` is smaller, the corresponding runtime is quite small (less than 2 s.).

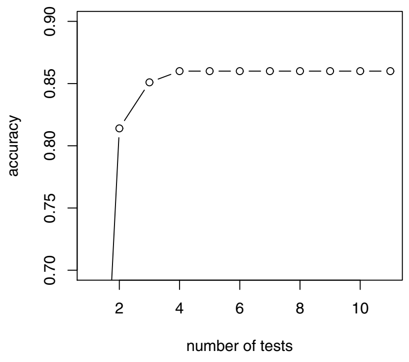

Fig. 2.4: Dependence of accuracy on number of tests with top-n execution

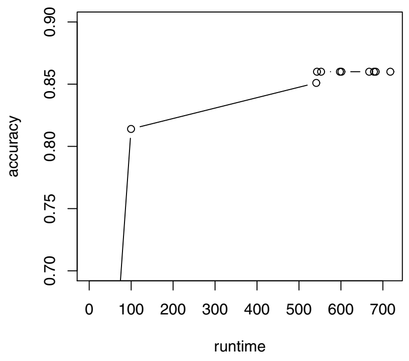

Fig. 2.5: Dependence of accuracy on runtime with top-n execution

Section 2.4 discusses another variant of the ranking method where runtime is incorporated into the algorithm. It is shown that this leads to marked improvements.

2.3.3 Evaluation of recommended rankings

An important question is how good/bad the recommendations are. Chapter 3 discusses the methodology that can be adopted to assess the quality of the recommendations generated by the system. It describes two different approaches. The first one aims to assess the quality by comparing it with the correct ranking, representing a golden standard. The second one aims to assess the effects on base-level performance when the ranking is followed.

2.4 Using a Combined Measure of Accuracy and Runtime

Rankings can be based on any performance measure we might wish to consider. Measures that combine accuracy (or AUC, F1, etc.) and runtime are of particular interest. Indeed, beforehand we do not know for sure which algorithms will perform well on the target dataset, and therefore a lot of time can be wasted on slower algorithms. Ideally, we want to schedule first the CV tests of fast, but relatively well-performing algorithms, before other, slower ones.

As Abdulrahman et al. (2018) have shown, this can lead to substantial speed-ups when seeking the best algorithm for a given target dataset. The concept of a combined measure that combines accuracy and runtime is not new. Various authors have proposed such a measure, including, for instance, Brazdil et al. (2003) who proposed the measure ARR. However, as was shown later (Abdulrahman et al., 2018), this measure is not monotonic. The authors introduced a measure A3R (shown in Chapter 5) that does not suffer from this shortcoming. Here we use a simplified version of this function, referred to as A3R' (van Rijn et al., 2015), which is defined as follows:

$$A3R'^{d_i}_{a_j} = \frac{P^{d_i}_{a_j}}{(T^{d_i}_{a_j})^Q}, \tag{2.3}$$

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. This function requires that a correct balance is established between the importance of accuracy and runtime. This is done by the parameter Q, which is in effect a scaling factor. Typically, Q would be a rather small number, such as $1/64$, representing in effect, the 64^{th} root. This is motivated by the fact that runtimes vary much more than accuracies. It is not uncommon that one particular algorithm is three orders of magnitude slower (or faster) than another. Obviously, we do not want the time ratios to completely dominate the equation.

For instance, when the setting is $Q = 1/64$, an algorithm that is 1000 times slower would yield a denominator of 1.114. This would thus be equivalent to the faster reference algorithm only if its accuracy were 11.4% higher than the reference algorithm.

A question arises regarding what the best setting for the parameter Q is. Abdulrahman et al. (2018) have investigated this issue. They have considered various settings for Q, including $1/4$, $1/16$, $1/64$, $1/128$ and $1/258$. They have shown that the setting $Q = 1/64$ was the best one, as it permitted to identify the good-performing algorithms earlier than the other options. This setting can be regarded as a useful default setting.

Average ranking is elaborated in the way described in Section 2.3. In addition to this, runtimes could be normalized for each dataset. Normalization is discussed in Chapter 3. For each dataset, the algorithms are ordered according to the performance measure chosen (here A3R) and ranks are assigned accordingly.

The average ranking is constructed by applying the Eq. 2.2. This upgrade has a rather dramatic effect on the loss curves, as can be seen in Figure 2.6 reproduced from Abdulrahman et al. (2018).

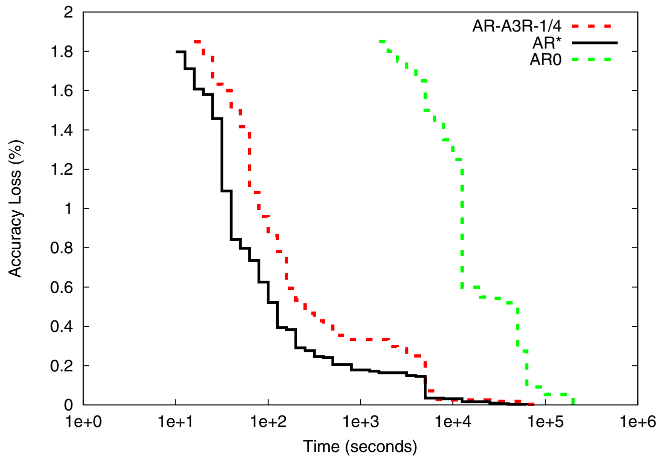

Fig. 2.6: Loss-time curves for A3R-based and accuracy-based average ranking

The loss curve of AR*, corresponding to the A3R-based average ranking method with the parameter setting $P = 1/64$, obtains a much better curve than the version AR0, corresponding to the case when only accuracy matters. With AR* the loss of 1% is achieved before reaching 100 seconds, while AR0 requires more than 10,000 seconds to obtain the same loss.

2.5 Extensions and Other Approaches

2.5.1 Using average ranking method to recommend workflows

Currently the attention of both researchers and practitioners is turning to the selection and configuration of workflows (pipelines) of operations. These typically include different preprocessing operations followed by the application of machine learning algorithms with appropriate hyperparameter configurations.

In this section we briefly mention the work of Cachada et al. (2017), which uses the variant AR* to recommend a workflow for a new dataset. The workflows may include a particular feature selection method (*correlation feature selection, CFS* (Hall, 1999)) and a particular classification algorithm selected from a given (62 in total). About half of these are ensembles. Besides, the authors also use different versions of some classification algorithms (algorithms with different settings of hyperparameters).

The authors show that AR* was able to select good-performing workflows. Their experiments also show that including feature selection and hyperparameter configurations as alternatives is, on the whole, beneficial. More details about this approach and other related approaches are given in Chapter 7.

2.5.2 Rankings may downgrade algorithms that are dataset experts

The rankings discussed in this chapter focus on algorithms that have high performance overall. Although this seems understandable, it also has a potential downside. Consider for example the following case, as shown in Table 2.5.

Table 2.5: Example metadataset, consisting of datasets $d_1 \ldots d_4$ and algorithms $a_1 \ldots a_4$. The table on the left shows the performance values of each algorithm on each dataset. The table on the right shows the ranks of each algorithm on each dataset

	d_1	d_2	d_3	d_4
a_1	0.66	0.63	**0.95**	0.65
a_2	**0.90**	**0.81**	0.89	**0.84**
a_3	0.82	0.79	0.83	0.83
a_4	0.74	0.76	0.84	0.77

	d_1	d_2	d_3	d_4
a_1	4	4	**1**	4
a_2	**1**	**1**	2	**1**
a_3	2	2	3	2
a_4	3	3	4	3

As can be seen, the complete ranking would be a_2, a_3, a_4, a_1, suggesting that a_1 is the worst algorithm to test. Looking at it from a different perspective, a_1 is, in fact, the only algorithm that manages to exceed the performance of a_2 on one dataset (d_3). In other words, when considering the performance on each dataset, algorithms a_2 and a_1 are the only algorithms that lie on the Pareto front.

The issue of how to identify and eliminate certain algorithms from a given set (which can be converted to a ranking) was addressed by Brazdil et al. (2001) and Abdulrahman et al. (2019). This approach is further detailed in Chapter 8 (Section 8.5).

Wistuba et al. (2015) investigated how to create a ranking of complementary algorithms. Pfisterer et al. (2018) showed that creating the optimal ranking based on metadata is an NP-complete problem, and proposed a greedy approach.

2.5.3 Approaches based on multi-criteria analysis with DEA

An alternative to designing a combined measure of two (or more) performance criteria is to use data envelopment analysis (DEA) (Charnes et al., 1978) for multi-criteria evaluation of learning algorithms (Nakhaeizadeh and Schnabl, 1997). One of the important characteristics of DEA is that the weights of the different criteria are determined by the method and not the user. However, this flexibility may not always be entirely suitable, and so Nakhaeizadeh and Schnabl (1998) have proposed a variant of DEA that enables to personalize the relative importance of different criteria. For instance, one user may prefer faster algorithms that generate interpretable models even if they are not so accurate.

2.5.4 Using dataset similarity to identify relevant parts of metadata

Section 2.3 described how to create a ranking model based on all metadata. However, not all metadata gathered in the experiments may be relevant to the task at hand. If

Table 2.6: Example of metadata with missing test results

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

the metadata includes test results on datasets that are rather different from the current task, using it may have an adverse effect on performance. So a question arises on how to identify which metadata is relevant for the given task.

One common approach involves using dataset characteristics to identify a subset of the most similar datasets to the target dataset and use the metadata associated with these datasets only. The approach presented here is motivated by the following hypothesis. If datasets are similar, then the algorithm rankings obtained on those datasets will be similar too. Dataset characteristics are sometimes called, in this context, *metafeatures*. Dataset characteristics are discussed in detail in Chapter 4.

We want to stress that a ranking approach can often be used without this step with quite satisfactory results. However, if dataset characteristics are not considered, the target dataset does not affect the order in which the algorithms are tested. In other words, the method follows a fixed schedule. Although this may not affect the final algorithm identified, more time may be needed to identify it. Having a flexible schedule may bring advantages. One such flexible schedule, different from the one discussed here, is presented in Chapter 5 (Section 5.8), which discusses an approach referred to as *active testing*.

2.5.5 Dealing with incomplete rankings

In practice, it sometimes happens that a certain proportion of test results is missing. That is, the test results of some algorithms on some datasets may be missing. So, if this happens, the resulting ranking will be incomplete. An example of an incomplete ranking is shown in row 5 in Table 2.1 and also in Figure 2.1(c). Table 2.6 shows an example with six algorithms (a_1 .. a_6) and six datasets (D_1 .. D_6). Note that in each column two out of the six results are missing.

Given that incomplete test results often arise in practice, a question arises regarding what to do. One simple and obvious answer is to complete the results. However, this may not always be possible, as a particular algorithm may have simply failed to run, or the know-how regarding how to run it is no more available. Also, running experiments with ML algorithms may often require substantial computational resources.

So, the other possibility is to use the incomplete metadata in the process of identifying the potentially best algorithm for the target dataset. This issue was investigated by Abdulrahman et al. (2018). The authors have shown that the performance of the average ranking method AR* that uses the combined measure of accuracy and runtime (discussed in Section 2.4), is not affected even by 50% of omissions in the metadata. This has an important implication. It indicates that we do not need to carry out exhaustive testing to provide quite good metamodels.

Abdulrahman et al. (2018) have shown that the method for aggregating incomplete rankings needs to be modified. More details are provided in the following subsection.

Aggregating incomplete rankings

Many diverse methods exist that can be used to aggregate incomplete rankings. According to Lin (2010), these can be divided into three categories: heuristic algorithms, Markov chain methods, and stochastic optimization methods. The last category includes, for instance, *cross-entropy Monte Carlo* (CEMC) methods.

Merging incomplete rankings may involve rankings of different size. Some approaches require that these rankings be completed before aggregation. Let us consider a simple example. Suppose ranking R_1 represents four elements, namely (a_1, a_3, a_4, a_2), while R_2 represents just two elements (a_2, a_1). Some approaches would require that the missing elements in R_2 (i.e., a_3, a_4) be attributed a concrete rank (e.g., rank 3). For instance, this strategy is used in package *RankAggreg* of R (Pihur et al., 2009). This is not right, as one should not be forced to assume some information when in fact there is none.

Abdulrahman et al. (2018) have proposed a relatively simple method for aggregating incomplete rankings that avoids this shortcoming. The method is based on the following observation: If two rankings are of unequal length, the ranks in the shorter one provide much less information than the ranks in the longer ranking. It is quite easy to see why. A substandard algorithm may appear in the first position if it is compared with another similar algorithm. The authors provide experimental evidence that this method provides quite good results despite its simplicity.

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.

Abdulrahman, S., Brazdil, P., Zainon, W., and Alhassan, A. (2019). Simplifying the algorithm selection using reduction of rankings of classification algorithms. In *ICSCA '19, Proceedings of the 2019 8th Int. Conf. on Software and Computer Applications, Malaysia*, pages 140–148. ACM, New York.

Asuncion, A. and Newman, D. (2007). UCI machine learning repository.

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.

Brazdil, P., Soares, C., and Pereira, R. (2001). Reducing rankings of classifiers by eliminating redundant cases. In Brazdil, P. and Jorge, A., editors, *Proceedings of the 10th Portuguese Conference on Artificial Intelligence (EPIA2001)*. Springer.

Cachada, M. (2017). Ranking classification algorithms on past performance. Master's thesis, Faculty of Economics, University of Porto.

Cachada, M., Abdulrahman, S., and Brazdil, P. (2017). Combining feature and algorithm hyperparameter selection using some metalearning methods. In *Proc. of Workshop AutoML 2017, CEUR Proceedings Vol-1998*, pages 75–87.

Charnes, A., Cooper, W., and Rhodes, E. (1978). Measuring the efficiency of decision making units. *European Journal of Operational Research*, 2(6):429–444.

Cook, W. D., Golany, B., Penn, M., and Raviv, T. (2007). Creating a consensus ranking of proposals from reviewers' partial ordinal rankings. *Computers & Operations Research*, 34(4):954–965.

Cook, W. D., Kress, M., and Seiford, L. W. (1996). A general framework for distance-based consensus in ordinal ranking models. *European Journal of Operational Research*, 96(2):392–397.

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.

Hall, M. (1999). *Correlation-based feature selection for machine learning*. PhD thesis, University of Waikato.

Hand, D., Mannila, H., and Smyth, P. (2001). *Principles of Data Mining*. MIT Press.

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

Kalousis, A. and Theoharis, T. (1999). NOEMON: Design, implementation and performance results of an intelligent assistant for classifier selection. *Intelligent Data Analysis*, 3(5):319–337.

Keller, J., Paterson, I., and Berrer, H. (2000). An integrated concept for multi-criteria ranking of data-mining algorithms. In Keller, J. and Giraud-Carrier, C., editors, *Proceedings of the ECML Workshop on Meta-Learning: Building Automatic Advice Strategies for Model Selection and Method Combination*, pages 73–85.

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.

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

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill.

Nakhaeizadeh, G. and Schnabl, A. (1997). Development of multi-criteria metrics for evaluation of data mining algorithms. In *Proceedings of the Fourth International Conference on Knowledge Discovery in Databases & Data Mining*, pages 37–42. AAAI Press.

Nakhaeizadeh, G. and Schnabl, A. (1998). Towards the personalization of algorithms evaluation in data mining. In Agrawal, R. and Stolorz, P., editors, *Proceedings of the Third International Conference on Knowledge Discovery & Data Mining*, pages 289–293. AAAI Press.

Neave, H. R. and Worthington, P. L. (1992). *Distribution-Free Tests*. Routledge.

Pavan, M. and Todeschini, R. (2004). New indices for analysing partial ranking diagrams. *Analytica Chimica Acta*, 515(1):167–181.

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*.

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

Soares, C. and Brazdil, P. (2000). Zoomed ranking: Selection of classification algorithms based on relevant performance information. 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 126–135. Springer.

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.

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.

Wistuba, M., Schilling, N., and Schmidt-Thieme, L. (2015). Sequential model-free hyperparameter tuning. In *2015 IEEE International Conference on Data Mining*, pages 1033–1038.