8

Setting Up Configuration Spaces and Experiments

Summary. This chapter discusses the issues relative to so-called configuration spaces that need to be set up before initiating the search for a solution. It starts by introducing some basic concepts, such as discrete and continuous subspaces. Then it discusses certain criteria that help us to determine whether the given configuration space is (or is not) adequate for the tasks at hand. One important topic which is addressed here is *hyperparameter importance*, as it helps us to determine which hyperparameters have a high influence on the performance and should therefore be optimized. This chapter also discusses some methods for reducing the configuration space. This is important as it can speed up the process of finding the potentially best workflow for the new task. One problem that current systems face nowadays is that the number of alternatives in a given configuration space can be so large that it is virtually impossible to gather complete metadata. This chapter discusses the issue of whether the system can still function satisfactorily even when the metadata is incomplete. The final part of this chapter discusses some strategies that can be used for gathering metadata that originated in the area of multi-armed bandits, including, for instance, SoftMax, upper confidence bound (UCB) and pricing strategies.

8.1 Introduction

The configuration space includes all possible workflows (pipelines) that can be constructed by combining the given set of base-level algorithms with all admissible configurations of their hyperparameters.

The search space is of great influence to the result of the hyperparameter optimization algorithm (Yu et al., 2020; Yang et al., 2020). Designing this configuration space too small can be harmful, as the search procedure might not yield good workflows for some datasets. On the other hand, designing this configuration space too large can also be problematic, as the search procedure might require a lot of time before it converges to something good. This chapter aims to address the issue of how to set up an adequate configuration space.

Organization of this chapter

Section 8.2 clarifies some basic notions that are useful when discussing configuration spaces. First, it discusses the difference between discrete and continuous subspaces. Then

P. Brazdil et al., *Metalearning*, Cognitive Technologies,
https://doi.org/10.1007/978-3-030-67024-5_8

the issue of sampling the continuous subspaces is taken up. Finally, we address the issue of how configuration spaces can be described.

Section 8.3 discusses certain criteria that enable us to affirm whether the given configuration space is (or is not) adequate for the tasks at hand.

Section 8.4 discusses the notion of *hyperparameter importance*. This notion helps us to determine which hyperparameters have a high influence on performance and should therefore be optimized. Hyperparameters that have a relatively small influence on performance could potentially be neglected, or else, less resources could be allocated to them.

Section 8.5 discusses some methods for reducing the configuration space. This is important as it can speed up the process of finding the potentially best workflow for each new task. This is simply because the system does not need to consider the alternatives that do not make a difference. Having a simple set up also has other advantages. Many users may want to know why a given recommendation system has come up with a particular solution, following the trend called *explainable AI* (Došilovič et al., 2018). Obviously, if the system is simpler, so are the explanations that can be given.

The success of a particular recommendation system also depends on the datasets used in the process of generating metadata. The issue of which datasets are needed is discussed in Section 8.7. One problem that current systems face nowadays is that the number of alternatives in a given configuration space can be so large that it is virtually impossible to gather complete metadata.

Section 8.8 discusses the issue of whether the system can still function satisfactorily even when the metadata is incomplete.

The final section (Section 8.9) discusses some strategies that can be used for gathering metadata that originated in the area of multi-armed bandits.

8.2 Types of Configuration Spaces

The term *configuration space* is used here to refer to the search space associated with a particular metalearning task. In this context we can distinguish among the following metalearning tasks:

- algorithm selection
- hyperparameter optimization and the combined hyperparameter optimization and algorithm selection (CASH) problem
- workflow design

Each of these tasks involves certain configuration space. The details on each are given in the following subsections.

8.2.1 Configuration spaces associated with algorithm selection

The configuration space associated with base-level algorithm selection consists of a set of base-level algorithms, which is usually referred to as a *portfolio*. The number of items in this portfolio determines the size of this space. In practical applications, there is a finite number of possibilities. As such, this is a *discrete* search space. Typically, there is only a finite number of values that can be transferred from one situation to another.

Chapters 2 and 5 described different ways of searching through this space. Section 8.5 describes a method for reducing this space, which is done by eliminating certain algorithms form the existing portfolio.

8.2.2 Configuration spaces associated with hyperparameter optimization and CASH

Base-level algorithms typically involve hyperparameters. Each algorithms has specific hyperparameters.

Types of hyperparameters

Some hyperparameters are categorical and hence discrete. Some examples of this type are: the type of SVM kernel, the sampling method of a random forest, or the distance function in a k-NN classifier.

Other hyperparameters are continuous. Some examples of continuous hyperparameters are: the kernel width of a SVM, the learning rate of a neural network, or the number of trees in a random forest.

Some algorithms include both categorical/discrete and continuous hyperparameters. In many systems, categorical and continuous hyperparameters are mixed together, as is done, for example, in the configuration space of Auto-sklearn (Feurer et al., 2015, 2019).

Continuous versus discrete spaces

The type of hyperparameter determines what kind of configuration space is involved in the associated task.

Discrete spaces consist of a fixed number of configurations, whereas continuous spaces consist potentially of an infinite number of configurations. Continuous spaces can be discretized. This has an advantage that it is relatively easy to gather metaknowledge on previous experiments, as there is only a finite number of configurations. The choice regarding whether to discretize (or not) is usually made by the designer of the configuration space.

So, typically we could say that hyperparameter optimization involves both *discrete* and *continuous spaces*.

Conditional hyperparameters and spaces

Sometimes, a set of hyperparameters is dependent on the specific value of another hyperparameter. These hyperparameters are called *conditional hyperparameters*. For example, many hyperparameters of support vector machines that control the kernel will only be relevant if a certain kernel is selected. As a more complex example, consider the Auto-sklearn search space (Feurer et al., 2015, 2019), where the set of hyperparameters is the union of the hyperparameters of all algorithms involved, plus additional hyperparameters determining which algorithms and preprocessing operators should be selected. All the hyperparameters are conditional on the values of the latter ones.

Sampling of continuous subspaces

Let us address a few issues that are relevant when dealing with numeric hyperparameters:

1. Type of sampling (uniform, log scale or other)
2. Level of detail

Continuous spaces are often sampled according to a chosen probability distribution. Some hyperparameters can be sampled uniformly, while others are sampled on a log scale, which seems more appropriate for others. For example, let us consider the task of selecting the number of trees for a gradient boosting classifier. We note that the change from 10 to 20 trees might produce a significant effect on performance, while the change from 1000 to 1010 trees would hardly lead to a significant effect. So this justifies the adoption of log-scale sampling for this parameter, which can form part of the set up. Snoek et al. (2014) proposed a method that determines what the optimal sampling rate is for every hyperparameter, freeing the user from this task.

Regarding the level of detail, numeric hyperparameters are often specified by the interval between the minimum and maximum value. Supposing we opt for uniform sampling in this range, the question is whether we should admit all possible values in this range or just some of them. On the one hand, we want to have a sufficiently fine resolution, so that we could observe all effects in performance. On the other hand, using too fine a resolution can complicate the search for the best setting.

Since the sampling is done according to a given probability distribution, sampling methods can keep running until a given time budget has been exhausted. The methods based on multi-armed bandits discussed in Chapter 8 (Section 8.9) provide other alternatives.

8.2.3 Configuration spaces associated with workflow design

Workflows (pipelines) are often defined as collections of steps (e.g., pre-processing steps, base-level algorithm) chained together. It is important to keep the size of this configuration space manageable. In general, certain formal structures including ontologies, grammars, or planning systems with operators can be used for this purpose (see Chapter 7). Each of these formal structures defines a configuration space. We note that Auto-sklearn (Feurer et al., 2015, 2019) uses a certain ontology that dictates the order and applicability of operators to include in a workflow (pipeline).

The aim is to design this space so that it would include all the required alternatives (here alternative workflows). It should not be unnecessarily "too big", as this could make the search for potentially best solution (here workflow) more difficult. This issue regarding whether the given configuration space is adequate for a given set of tasks is discussed in the next section.

8.3 Adequacy of Configuration Spaces for Given Tasks

Let T_C represent the tasks that can be solved adequately by a system that has access to configuration space C. Similarly, let T_R represent various tasks that we expect to encounter in the near future. Let us suppose, that to solve them in an adequate way, we would need configuration space R.

Note that R is a hypothetical concept meant for illustrative purposes. We can never know the exact contents of R, as we will not know what tasks we will encounter in the future. But we may know some instances of R, namely the tasks we have encountered until now. Having the knowledge of some instances enables us to make a guess regarding the underlying distribution of tasks. This way may cover also future tasks under the assumption of stationarity, i.e. that the distribution will not change in the future. The following situations can arise:

- $R \equiv C$
- $R \subset C$
- $C \subset R$
- $ExCond \wedge C \cap R \neq \oslash$

where $ExCond$ guarantees that the three above cases are excluded. This condition can be defined as $ExCond = (R \neq C) \wedge (R \not\subset C) \wedge (C \not\subset R)$.

Let us analyze each case separately.

Case $\mathbf{R \equiv C}$: If this case occurs, then we are well prepared for T_R. Nothing needs to be done.

Case $\mathbf{R \subset C}$: If this case arises, at first sight it seems that everything is fine, as in principle we could solve all the required tasks. However, as not all elements in C may be needed, time may be wasted in the search for the right algorithm. For instance, if the required task is to distinguish between two classes only and the configuration space includes methods suitable for any number of classes, it is possible to pre-select the appropriate subset.

However, our configuration space may also include algorithms that have sub-standard performance (non-competitive algorithms), or algorithms with good performance but which are redundant. So, the objective here is to identify a reduced configuration space C' which can still solve all the required tasks, namely $(R \equiv C') \ \wedge \ (C' \subset C)$. Reducing the configuration space is beneficial in general, as it reduces the time required in the search for the potentially best solution. However, this process incurs certain risks. If it is reduced too much, the optimal method (e.g., classifier) may be missed out. Section 8.5 discusses the reduction method in detail.

Case $\mathbf{C \subset R}$: In this case the problem is more serious, as we are unable to solve all the required tasks with recourse to the algorithms in the current configuration space. For instance, if our configuration space includes only methods for classification tasks and the required tasks include both classification and regression, then it is necessary to extend the current portfolio by the inclusion of appropriate methods.

Case $\mathbf{ExCond \wedge C \cap R \neq \oslash}$: Let us call this intersection C'. This means that only some problems in R can be solved with C' (as in case $C \subset R$). This means that only some problems in R can be solved with C' (as in case $C \subset R$). Moreover, C' includes some algorithms that are not really needed for R (as in case $R \subset C$).

8.3.1 General principles for the constitution of configuration spaces

It is possible to define a set of general principles that should be followed in the constitution of discrete configuration spaces. For continuous configurations, the situation is a bit different as it is defined by ranges, rather than a set of configurations.

Let us assume that a set of datasets and tasks is given. Let us consider a configuration space containing the following set of alternatives $C = c_1, \ldots c_m$, where each c_i represents a workflow with some specific hyperparameter settings. The general principles for including an element c_i are:

1. **Minimal relevance:** For most datasets there should be a c_i that obtains better performance than a suitable baseline. A *baseline* is a simple method which establishes a reference for minimally acceptable results. In supervised learning problems, for instance, it consists of predicting the most frequent class.
2. **Positive marginal contribution/individual relevance:** Each item c_i should provide some marginal contribution to the set C without C_i $(C - c_i)$. Effectively this means that c_i should be the best option for at least one task.

3. **Impossibility to improve on marginal contribution:** Given some preselected set C, the results cannot be further significantly improved by adding additional elements c_j to it.
4. **Impossibility to improve on individual relevance:** For every c_i there should not exist a c_j such that the performance of c_i is never significantly better than that of c_j for all tasks considered.

We note that the principles are oriented towards the available datasets and tasks. In other words, as we do not know the tasks that we will encounter, this is usually established for the metadata of previously seen tasks. If we want to be well prepared for future datasets, it is advisable to relax somewhat the last two principles. Some competitive algorithms may be useful in future tasks, even if they have not been included in the best equivalence group on any past dataset.

Some of these issues are taken up again in subsequent subsections, where we discuss specific methods for constructing portfolios of algorithms (workflows).

8.4 Hyperparameter Importance and Marginal Contribution

In this section we cover the marginal contribution of certain elements in a given configuration space. Subsection 8.4.1 discusses the marginal contribution of algorithms (workflows), while Subsection 8.4.2 is oriented towards hyperparameter importance for a specific dataset. Subsection 8.4.3 generalizes the notion of hyperparameter importance across datasets.

8.4.1 Marginal contribution of algorithms (workflows)

Some researchers have investigated the issue of assessing the complementarity of algorithms. Xu et al. (2012), for instance, have defined a notion of *marginal contribution* to the performance, i.e., how much the performance of an existing portfolio is improved by adding a new algorithm to it. This approach has the disadvantage of being dependent on a fixed portfolio. A broader view of an algorithm contribution, which extends the marginal contribution analysis, involves a so-called *Shapley value* (Fréchette et al., 2016). This value determines the marginal contribution of an algorithm to any subset of the algorithm portfolio.

Shapley values come from the area of cooperative game theory. The setup involves a coalition of players who cooperate and obtain a certain overall gain from that cooperation. As some players may contribute more to the coalition than others or may possess different bargaining power (for example, threatening to destroy the whole surplus), a question arises of how important each player is to the overall cooperation and what the payoff is. The Shapley value provides one possible answer to this question.

The techniques described in this section work on continuous configuration spaces.

8.4.2 Determining hyperparameter importance on a given dataset

The problem of how to automatically optimize hyperparameters of algorithms has drawn a lot of attention in recent years. The reader can consult Chapter 6, where some of the more important methods are discussed. Most of these techniques require that the relevant hyperparameters for each algorithm are given and accompanied by a specification of the possible settings that can be considered. This can be done either by specifying the intervals of values or simply by enumerating all possible settings.

Various techniques exist that enable to identify, for a given dataset and algorithm, the most important hyperparameters. These include approaches using

- *Forward selection* (Hutter et al., 2013)
- *functional ANOVA* (Sobol, 1993; Hutter et al., 2014)
- *Ablation analysis* (Biedenkapp et al., 2017; Fawcett and Hoos, 2016)

The three techniques described further on enable us to determine the importance of hyperparameters for a given dataset. All require metadata about configurations and the corresponding performance values on the particular dataset. Note that these techniques operate on a continuous configuration space.

Forward selection

Forward selection (Hutter et al., 2013) trains a surrogate model to map hyperparameter values to performance values. The work assumes that including important hyperparameters as input to the surrogate model has a high positive impact on performance. This setting is similar to the experiment of Breiman (2001) on feature importance.

The method starts with an empty set, and greedily adds the hyperparameter that has the highest impact on the predictive performance of the surrogate model. This process continues in an iterative manner until no further improvement can be made. This yields a ranking of hyperparameters ordered by importance to this surrogate model.

Ablation analysis

Ablation analysis (Fawcett and Hoos, 2016) calculates a so-called *ablation trace*. The method first executes the default configuration on the dataset and establishes its performance. Afterwards, it determines the optimal hyperparameter setting using an optimization procedure, such as sequential model-based optimization (SMBO) discussed in Chapter 6 (Section 6.8). The goal is to determine which of the hyperparameters influence performance the most when changing the value from the default setting to the optimal setting. It starts with the optimal configuration, and considers all configurations that can be reached from the optimal configuration by switching one hyperparameter value to the default value. It continues in that fashion until all hyperparameters have been switched and the default configurations has been reached. This results in a ranking of hyperparameters in decreasing importance. This is called the ablation trace. Note that this method requires that, after the optimal hyperparameters have been determined, several models along the ablation trace are trained. This makes the procedure potentially time consuming. Biedenkapp et al. (2017) show how surrogate models can be used to avoid this and run ablation analysis faster.

Functional ANOVA

Hutter et al. (2014) apply functional ANOVA (Sobol, 1993) to establish hyperparameter importance. Functional ANOVA determines how much each hyperparameter (and each combination of hyperparameters) contributes to the variance of the performance. It works on the concept of the *marginal* of a hyperparameter. In the context of machine learning, a marginal reflects the relation between the value of an hyperparameter and the performance of that algorithm. Specifically, for each value for that hyperparameter, it reflects how the algorithm would perform, averaged over all possible combinations of the other hyperparameters and their settings. This might not seem feasible, as there are an exponential number of combinations. However, Hutter et al. (2014) showed how this can be calculated efficiently using tree-based surrogate models trained on the performance data of configurations on the dataset. Hyperparameters that have a large variance across the marginal are deemed important. The opposite is also true: hyperparameters that have a low variance are considered unimportant. Note that functional ANOVA determines the importance of hyperparameters globally; the conclusions that are drawn do not depend on specific values of other hyperparameters.

8.4.3 Establishing hyperparameter importance across datasets

All three aforementioned techniques are post hoc techniques; i.e., when confronted with a new dataset, they do not reveal which hyperparameters are important prior to experimenting on that particular dataset. In this subsection, we describe efforts made towards establishing hyperparameter importance across datasets.

In order to gain a better understanding about which hyperparameters are important in general, van Rijn and Hutter (2018) and Probst et al. (2019) apply the following procedure:

1. Determine a suitable set of datasets;
2. Gather ample configurations and their performance on these datasets;
3. Apply a hyperparameter importance framework on it;
4. Aggregate the results in a human-understandable format.

For determining a suitable set of datasets, there are several considerations to take into account. On the one hand, it would be interesting to consider a broad set of datasets. For example, the OpenML-CC18 (Bischl et al., 2021) seems a suitable choice. However, in some specific studies, it makes sense to consider only a subset of datasets. For instance, Probst et al. (2019) are specifically interested in binary classification and hence use a subset of the “OpenML-100” with a binary target. Similarly, Sharma et al. (2019) are interested in image classification, and therefore define a set of ten image datasets.

As most methods for establishing hyperparameter importance rely on the use of a surrogate model, we need to have ample data gathered on the given datasets. The data consist of pairs of items, namely a configuration and the corresponding measure of performance. As surrogates become increasingly more accurate when trained on a larger number of configuration and performance pairs, we need a sufficient number of such pairs. As for the methods for establishing hyperparameter importance, all the aforementioned techniques (forward selection, ablation analysis, and functional ANOVA) can be used. However, all these methods are based on certain assumptions, and as such, the results may differ depending on which choice was made. Still, a quick investigation of the results shows that the methods based on tunability and functional ANOVA discussed earlier seem to agree on the most important hyperparameters (van Rijn and Hutter, 2018; Probst et al., 2019). Chapter 17 looks at this in more detail.

Regarding the aggregation of results, one could simply aggregate the results per hyperparameter and per dataset into a boxplot. Functional ANOVA returns a clear and interpretable fraction, representing the contribution to the overall variance. For ablation analysis and forward selection, the outcome is in the form of a ranking of hyperparameters according to their importance. Then a ranking-based aggregation can be used, which can lead to *critical distance* plots, as suggested by Demšar (2006).

8.5 Reducing Configuration Spaces

8.5.1 Reducing portfolios of algorithms/configurations

The issue of how to identify and eliminate certain algorithms (configurations, workflows) from a given portfolio and evaluate the effects was addressed by Brazdil et al. (2001) and Abdulrahman et al. (2019). The method involves two steps. The aim of the first step is to identify competitive algorithms. The non-competitive algorithms are effectively dropped at this stage. The aim of the second step is to seek a small number of *specialists/experts* for each dataset. This step results in the elimination of many potentially redundant algorithms.

More details on both steps are given in the following subsections. Both methods are defined for discrete configuration spaces.

Identifying competitive algorithms

The assumption followed here is that the non-competitive algorithms have little chance to achieve a competitive result on the new target dataset. This is done by applying Algorithm 8.1, which calls Algorithm 8.2.

```
input  : Datasets $D_s$
         Portfolio of algorithms $A_{in}$
output: Portfolio of competitive algorithms $A_c$
1: Initialize $A_c$ to empty list
2: for all $d_i \in D_s$ do
3:     $A_{c_i} \leftarrow$ Identify competitive algorithm ($A_{in}$,$d_i$)
4:     $A_c \leftarrow A_c + A_{c_i}$
5: end for
```

Algorithm 8.1: Identifying competitive algorithms for all datasets

Algorithm 8.1 requires as input the datasets D_s and a set of algorithms A_{in} and outputs a subset of algorithms A_c. The for-loop (lines 2–5) includes a call to Algorithm 8.2 (on line 3), which returns list A_{c_i} representing the most competitive algorithms of A_{in} for the dataset d_i. Any performance measure can be used to identify such algorithms. It can be, for instance, accuracy, or as Abdulrahman et al. (2019) have shown, a combined measure of accuracy and runtime. This list includes the topmost algorithm in the average ranking and all algorithms with equivalent performance. Finally, list A_{c_i} is added (using the operator $+$) to the A_c, representing a list of lists.

Algorithm 8.2 works as follows: First, A_{c_i} is initialized to an empty list. Then the ranking R_{d_i} is constructed on the basis of the test results of the algorithms of A_{in} on dataset d_i. There is no need to conduct tests, as the test results can be simply retrieved from the meta-database. The next goal is to initialize a_{best} to be the topmost algorithm in R_{d_i}.

The method proceeds by identifying all algorithms (a_{eq}) with an equivalent performance (e.g., a combined measure of accuracy and runtime) to a_{best}. This is done by processing all algorithms (configurations) $a_j \in A_{in}$, and conducting a statistical test

```
input : Algorithms $A_{in}$, Dataset $d_i$
output: Competitive algorithms $A_{c_i}$
1: Initialize $A_{c_i}$ to empty list
2: Construct a ranking $R_{d_i}$ of algorithm in $A_{in}$ on $d_i$
3: Identify the topmost algorithm $a_{best}$ in $R_{d_i}$
4: Use a statistical test to identify algorithms in $a_{eq}$ with equivalent performance
5: $A_{c_i} \leftarrow a_{best} + a_{eq}$
```

Algorithm 8.2: Identify competitive algorithms for a specific dataset

(e.g., Wilcoxon signed-rank test with a confidence level of 95% (Demšar, 2006)) to determine whether the performance of a_j is equivalent to a_{best}. This test is done on the basis of *fold* information of the cross-validation procedure that is available in the meta-database. All algorithms that have an equivalent performance to a_{best} are included in the candidate set A_{c_i}. When all pairs of algorithms have been processed, list A_{c_i} is returned.

Example

To explain how the non-competitive algorithms are removed from the ranking, as detailed in Algorithm 8.1, we show an example. For simplicity, our example includes only six algorithms (a_1,a_2,...,a_6) and six datasets (D_1,D_2,...,D_6) (see Table 8.1). The algorithms in dark-grey slots represent the topmost algorithms (a_{best}) identified for each dataset.

The topmost algorithm for each dataset is then compared with the other algorithms in the ranking. For example, when dealing with R_{D_1}, algorithm a_2 is identified as a_{best}. It is compared with a_6, a_4, a_3, a_5, and a_1 using the Wilcoxon signed-rank test. Any algorithm that has a similar performance to the topmost algorithm is maintained in the ranking, while all the others are dropped. In Table 8.1 the algorithms that have been maintained are shown in grey slots.

This process is repeated for all datasets used in the experiment, and the competitive algorithms for each dataset are identified. In our example the competitive set of

Table 8.1: Competitive algorithms (in gray) identified using statistical test

Rank	R_{D_1}	R_{D_2}	R_{D_3}	R_{D_4}	R_{D_5}	R_{D_6}
1	a_2	a_5	a_4	a_6	a_4	a_1
2	a_6	a_1	a_2	a_5	a_2	a_2
3	a_4	a_3	a_1	a_1	a_3	a_4
4	a_3	a_6	a_5	a_2	a_5	a_5
5	a_5	a_4	a_3	a_3	a_6	a_3
6	a_1	a_2	a_6	a_4	a_1	a_6

algorithms includes

$$A_c = \{(a_2)(a_5, a_1)(a_4, a_2, a_1)(a_6, a_5, a_1)(a_4, a_2)(a_1)\}. \quad (8.1)$$

If we eliminate the duplicates, we obtain

$$A_c = \{a_2, a_5, a_1, a_4, a_5\}. \tag{8.2}$$

A question arises as to whether all these algorithms are needed or whether we can still drop some. This issue is covered in the next section.

Using covering algorithm to select "non-redundant" algorithms

If a portfolio is constructed by adding various algorithms to it, it could include various versions of the same algorithm with very similar performance. Their inclusion in the algorithm portfolio may not really be desirable.

The issue of whether two algorithms (configurations) have similar performance can be determined on a macro- or micro-level. Comparisons on a macro-level involve measures (e.g., accuracy or area under the ROC curve) on the whole dataset. They represent aggregated measures across different examples. One approach that uses this notion of similarity is discussed in the next subsection.

Alternatively, it is possible to exploit the notion of similarity on a micro-level, i.e., by taking into account the performance on individual examples. This approach is discussed further on.

Both approaches seek to "cover" each dataset by at least one algorithm. The term "algorithm covers a dataset" is used here to indicate that the algorithm appears in the subset of the algorithms with the best performance. All algorithms identified as "similar" are dropped.

In addition, both approaches give preference to algorithms that cover preferably many datasets. This is based on the assumption, that by assembling a set of algorithms that work well for many past datasets, it is likely that one of these will be a good choice for the new datasets.

Using covering algorithm with macro-level similarity

This approach of Abdulrahman et al. (2019) followed a covering approach and a hill-climbing strategy. In the first step, the algorithm that covers the largest number of datasets is selected. This approach assumes that it is sufficient to cover each dataset using just one algorithm. All other algorithms with a similar macro-level performance are not considered and hence effectively dropped from further consideration. All datasets covered are eliminated from further consideration. The process is repeated in an iterative manner until all datasets have been covered. The authors have shown that a particular metalearning system, AR* (see Chapter 2), can identify well-performing algorithms sooner with a reduced portfolio than similar metalearning systems that use a full (unreduced) portfolio.

We note that this approach may eliminate algorithms that appear similar on a macro-level but are rather different on a micro-level. This shortcoming can be avoided by adopting micro-level in the detection of similarity.

Using a covering algorithm with micro-level similarity

One good measure to detect micro-level similarity is *classifier output difference* (COD) (Peterson and Martinez, 2005), which determines the proportion of cases on which two classifiers differ in their predictions. It is defined as follows:

$$COD(i,j,d) = \frac{|x \in d \text{ s.t. } \hat{f}_i(x) \neq \hat{f}_j(x)|}{|d|}. \tag{8.3}$$

Here, d is the dataset at hand and x represents an instance (example) from d. The two classifiers that are being compared are denoted by i and j, and their predictions on a given instance x are denoted by $\hat{f}_i(x)$ and $\hat{f}_j(x)$, respectively. So if this measure is 0 (or very near to this value) it indicates that the two classifiers generate virtually the same predictions on the given dataset. Lee and Giraud-Carrier (2011) use this measure to determine a set of different classifiers, which would complement each other in a portfolio.

8.5.2 A reduction method oriented towards a combination of measures

The reduction method described above works with the combined measure of accuracy and time. However, we note that the relative importance of accuracy and runtime is fixed. This is done by setting the value of the parameter Q (see Eq. 2.3 in Chapter 2). So this represents a certain limitation, as only a certain region of points on the so-called *Pareto front* is considered important.

The concept of *Pareto front* (or *frontier*) comes from the area of multiobjective optimization (Miettinen, 1999). A solution is called *nondominated, Pareto optimal*, if none of the objective functions can be improved without degrading some of the other objective values. A set of Pareto-optimal solutions constitutes the Pareto front.

So a question arises as to how to extend the method discussed in the previous subsection to cover the algorithms on the Pareto front (or in its vicinity). We describe two approaches in the following subsections.

Approach based on an envelopment curve

This approach tries to identify points (algorithms) on the envelopment curve or in its vicinity. Each point (algorithm) is characterized by two coordinates: runtime and accuracy. Various methods exist that can identify the points on the envelopment curve. Brazdil and Cachada (2018) have used a relatively simple method that led to quite satisfactory results. This method first orders all points (algorithms) by their runtime. Then all points are examined one by one, and if the accuracy of the successor is higher than that of all its predecessor, it is assumed to represent a competitive algorithm. Only the competitive algorithms are returned.

Figure 8.1 shows an example of both an unreduced set of algorithms (blue points) and the corresponding reduced set (red triangles) for one dataset.

This approach can be regarded as a simple solution for a rather complex problem. Although it can identify the points on the Pareto front, it does not use any notion of an uncertainty band below this front.

8.6 Configuration Spaces in Symbolic Learning

Version spaces

T. Mitchell has defined so-called *version spaces* in the context of concept learning (Mitchell, 1980, 1982, 1990, 1997). The version space contains all possible hypotheses consistent with the given positive and negative examples and shows that all these

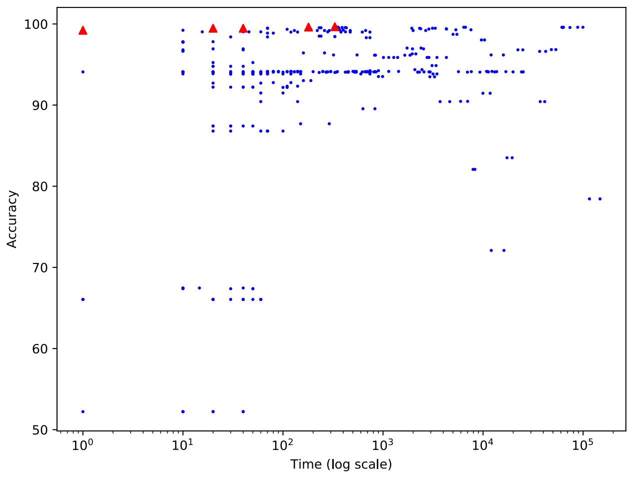

Fig. 8.1: Example of an unreduced set (blue points) and the corresponding reduced set (red triangles) of algorithms for one dataset

hypotheses can be organized in a lattice.[1] Thus, it is possible to follow the links from more specific to more general (or vice versa). Besides, as was shown, it is possible to define the general boundary G and the specific boundary S. Concept learning that exploits the version space proceeds by reducing the version space as each new example is analyzed.

Various researchers have considered the conditions needed by a given concept learning system so that it can generate the target hypothesis. In this context, the term *bias* was often used (Mitchell, 1990; Russell and Grosof, 1990b,a; Gordon and desJardins, 1995). Mitchell (1990) distinguishes between *weak* and *strong* bias, depending on whether weak or strong assumptions need to be made to be able to generate a model capable of classifying well all examples.

Controlling the domain-specific language bias

Various researchers have proposed to control the language bias in various ways. For instance, these included *determinations* (Davies and Russell, 1987), *relational clichés* (Silverstein and Pazzani, 1991), *clause schemata* (Kramer and Widmer, 2001), *metapredicates* that define a translation between a *metafact* and a domain-level rule (Morik et al., 1993), and *topologies* (Morik et al., 1993), representing abstracted graphs of rules. Other

[1]A lattice consists of a partially ordered set of points in which every two elements have a unique supremum (also called a least upper bound) and a unique infimum (also called a greatest lower bound).

researchers have proposed various approaches based on grammars, including, for instance, the proposal of Cohen (1994). Some of these grammar-based approaches restrict the concepts that can be introduced (e.g., Jorge and Brazdil (1996)); Others impose restrictions on the variables also, such as the *D*LAB formalism (De Raedt and Dehaspe, 1997). Grammar-based formalisms were discussed in Chapter 7.

Although many proposals have been made in the past, to the best of our knowledge, no large-scale comparative studies have been carried out to determine which representation would be the best one. The areas of AutoML and metalearning open new possibilities, so it is conceivable that new comparative studies will be carried out in the future.

Extending the domain-specific language bias

In concept learning, when the version space has shrunk and the resulting version space is empty, it can be taken as an indication that something must be altered. Mitchell (1990) suggests that, for instance, the representation language should be changed to also allow, for instance, disjunctive concepts apart from disjunctive ones. As was pointed out, this analysis is applicable only to noise-free data. However, it can be extended to noisy settings by assuming that the data may contain a certain given proportion of noisy examples, as, for instance, Mitchell (1977) and Hirsh (1994) have shown.

Here, we are concerned with enriching the descriptive language. Too many errors can be taken as a sign that the descriptors of cases should be extended and, consequently, it is possible to arrive at the required target concept. This was already suggested by Russell and Grosof (1990b):

"It is important to note that the deductive process needs to be under some higher-level control in order to handle the case of the collapse of the version space when the observations are inconsistent with the initial concept language bias. In such contexts it becomes necessary to weaken the concept language bias, by relaxing the constraints on the form of concept definitions, or by extending the allowed predicate vocabulary".

8.7 Which Datasets Are Needed?

In Section 8.3 we discussed the relationship between tasks T_C that can be solved by a given system that can exploit the configuration space C and tasks T_R that we expect to encounter in the future. The main issue there was whether the given set of algorithms is adequate to solve T_R. Basically, we wish to have $T_R \subset T_C$.

We note that metalearning approaches require not only a set of algorithms, but also metaknowledge containing information regarding the performance on different datasets associated with particular task. As was shown in some of the previous chapters (e.g., Chapters 2 and 5), the metaknowledge is used to provide recommendations on the new dataset. This approach is successful if the new dataset and some dataset encountered in the past deal with the same *task*, such as classification that involves an unbalanced class distribution. In addition, the two datasets should be sufficiently similar. So, we need a sufficient number of datasets for each type of task we may encounter in the future.

Researchers and practitioners working in this area have used various strategies to come up with an answer to this problem. Some strategies that have been considered in the past are:

- Relying on existing repositories of datasets

- Generating synthetic datasets
- Generating variants of existing datasets
- Segmenting a large dataset or data stream
- Searching for datasets with discriminatory power

All the alternatives above are discussed in more detail in the following subsections.

8.7.1 Relying on existing dataset repositories

Currently, various dataset repositories exist from which different datasets may be retrieved. Some well-known repositories are:

- University of California at Irvine (UCI) Machine Learning Repository (Asuncion and Newman, 2007)
- OpenML (Vanschoren et al., 2014)
- UCI Knowledge Discovery in Databases Archive (Hettich and Bay, 1999)
- University of California at Riverside (UCR) Time Series Data Mining Archive
- UCR Time Series Data Mining Archive (Keogh and Folias, 2002)

among others. In the UCI repository, there are several hundred datasets. Although this is sufficient for many purposes, it is not much for metalearning. We cannot expect to obtain a general model for such a complex problem as algorithm recommendation using a limited number of datasets. Also, there is no guarantee that the datasets constitute a representative sample for each possible task we may encounter in future.

Dataset repositories are discussed in more detail in Chapter 16.

8.7.2 Generating synthetic datasets

The generation of synthetic datasets could be regarded as a natural way to extend the number of datasets needed in metalearning. In the proposal by Vanschoren and Blockeel (2006), new datasets are generated by varying a set of characteristics that describe the concepts to be represented in the data. The characteristics include the *concept model* and the *size* of the model. The datasets generated should have similar properties to natural (i.e., real-world) data.

Vanschoren and Blockeel (2006) propose the use of existing techniques for experimental design as an inspiration to guide dataset generation for metalearning studies. However, they recognize that building such a generator is a challenging task. Partial solutions have been proposed, in which the correlation between features and concepts is obtained by recursive partitioning on the space of features (Scott and Wilkins, 1999).

Given that it is difficult to make sure that the datasets generated are similar to natural ones, this approach is more suitable for understanding algorithm behavior than for the purpose of algorithm recommendation.

8.7.3 Generating variants of existing datasets

An alternative method to obtain more metadata is to generate new datasets by manipulating existing ones. This may be done either by changing the values of a particular feature, which may affect its distribution, or by changing the structure of the data (e.g., adding irrelevant or noisy features) (Aha, 1992; Hilario and Kalousis, 2000). Usually the changes are done separately on independent features and on a dependent one (i.e., the target feature) by, for instance, adding noise. Usually the changes are focused on a certain aspect of the behavior of the given algorithms. For instance, the addition of redundant features can be used to investigate the resilience of some algorithms to redundancy.

Soares (2009) proposed a method to obtain a larger number of datasets using a simple transformation of the existing datasets. The new datasets generated are referred to as *datasetoids*. The author tested the approach on the problem of using metalearning to predict when to prune decision trees. The results show a significant improvement when using datasetoids.

However, the increase in the number of datasets raises a problem, as it is necessary to estimate the performance of the algorithms on these datasets. Running all candidate algorithms on all new datasets can be computationally very expensive. Prudêncio et al. (2011) have proposed to use active learning, which in this context represents *active metalearning*. The authors have shown that it is possible to significantly reduce the computational cost not only without loss of metalearning accuracy but with potential gains.

8.7.4 Segmenting a large dataset or data stream

New datasets may be generated by segmenting a large dataset or data stream. In the area referred to as *extreme data mining* (Fogelman-Soulié, 2006), a large database is segmented into a number of subsets (e.g., by customer or product) and different models are generated for each subset.

In massive data streams, large volumes of data are continuously available. Some data streams include a concept shift, where some aspect of the data changes. Such a data stream can be used to generate different datasets corresponding to somewhat diverse portions of the data.

More details about algorithm recommendation for data streams can be found in Chapter 11.

8.7.5 Searching for datasets with discriminatory power

In this subsection we discuss two different approaches. The first one uses dataset characteristics and algorithm performance to obtain so-called 2D footprints. The authors show that many algorithms have overlapping footprints. The second approach uses pairs of rankings to characterize the diversity of both datasets and algorithms.

Using datasets characteristics and 2D footprints

In order to guarantee good performance of a metalearning system relying on a given portfolio, we need sufficiently diverse datasets that enable to discriminate well between different alternatives so as to provide the best possible recommendation for each case. This problem was addressed by Muñoz et al. (2018).

Their study included 235 datasets in total, most of which were from the UCI.[2] The methodology proposed relies on a good characterization of all datasets, so the authors have considered quite a large number of features (509) for this task. Their aim was to select a small subset that would characterize well the hardness of the classification task.

[2]The authors refer to the datasets as *instances*.

The details regarding the selection process are described in their paper. This way the authors identified just ten features is discussed in Section 4.7.

In the next step, the ten-dimensional space was projected onto a two-dimensional space, thus enabling to visualize classification datasets as points in a two-dimensional space. This reveals pockets of hard and easy datasets in an area in the 2D space, referred to as a *footprint*, where a particular algorithm is expected to do well. Quantitative metrics capturing, for instance, the area of the footprint provide objective measures of the relative advantage of an algorithm across the given set of datasets.

The results presented in this paper demonstrate the lack of diversity of the given test datasets, as most algorithms have similar footprints. This may be due to three possible reasons: (1) the algorithms are all essentially rather similar, (2) the datasets are not revealing the strengths and weaknesses of each algorithm as much as is desired, or (3) the features used may not be sufficiently discriminative.

The authors have proposed a method to generate new test datasets, aiming to enrich the diversity of datasets. The proposed method uses a Gaussian mixture model (GMM), which can be tuned. The sample from GMM is characterized by f_S. The method requires defining the target vector of features, which drives the tuning process. As the authors have shown, this process can lead to datasets covering well the 2D space. Future work should show that the richer metadata is useful and indeed facilitates the process of selecting algorithms for new datasets.

Using correlation of rankings to characterize diversity

Abdulrahman et al. (2018) have proposed to characterize a given metalearning problem in the following way. They have observed that, if two datasets are very similar, the algorithm rankings will also be similar and, consequently, the pairwise correlation coefficient will be near 1. On the other hand, if the two datasets are quite different, the correlation will be low. In the extreme case, it will be –1 if one ranking is the inverse of the other. So the distribution of pairwise correlation coefficients provides an estimate of how difficult the metalearning task is for a given portfolio of algorithms.

Figure 8.2 shows a histogram of correlation values, reproduced from Abdulrahman et al. (2018). The histogram can be characterized by its median value and appropriate percentiles (e.g., 25% and 75%). A metalearning task can be considered easy if the median value is high and if the inter-percentile range is small. Let us consider why this is so. If we select any dataset and consider it similar to the target dataset, many other datasets are similar to it. So we can reuse the metaknowledge acquired on these datasets.

However, we note that this method can be used only if the new target datasets are of the same kind as the datasets analyzed so far (i.e., those used to construct the distribution).

8.8 Complete versus Incomplete Metadata

In Chapters 1, 2, and 5 we have presented a basic scheme of metalearning, which involves the following steps: generation of metadata, generation of meta-model, and application of the meta-model to the target dataset. The first step involves running experiments of some algorithms (or workflows) on some of the available datasets. If we were to run all algorithms (or workflows) on all available datasets, we would obtain *complete results* and hence complete metadata.

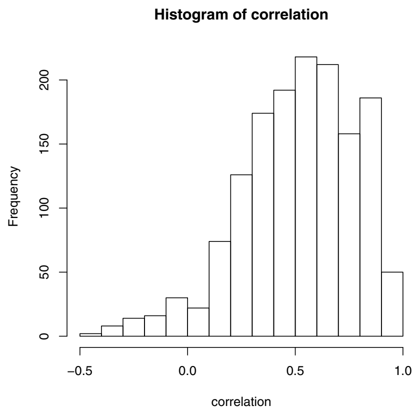

Fig. 8.2: Spearman's rank correlation coefficient between rankings for pairs of datasets

Incomplete results would arise if, for some pair (or pairs) of datasets and algorithms (or workflows), the performance result would not be available. The aim of this section is to address the following questions:

- Is it possible to obtain complete metadata?
- Is it necessary to have complete metadata?
- Does the order of the tests matter?
- How can we exploit the ideas from multi-armed bandits to schedule tests?
- Should we delegate the responsibility of gathering test results to the community?

The questions above are addressed in subsequent sections. The last one is discussed in a separate chapter (Chapter 16).

8.8.1 Is it possible to obtain complete metadata?

The answer to the above question is, in general, negative. A complete set of results could only be obtained in situations that involve a limited number of algorithms and datasets. Our answer is negative for various reasons that are explained in the following subsections.

There may be too many experiments to carry out

The number of experiments depends on the size of the search space. When dealing with continuous configuration spaces, there is an infinite number of configurations. As such, there is no way to enumerate all possible configurations or store them, unless the configuration space is discretized in some way. Even in the domain of discrete search spaces, if we were to deal with a modest number of 100 algorithms, each with 100 different hyperparameter settings, and conducted tests on 100 datasets, we would have to conduct 100^3 (i.e., 10^6) tests.

This number would grow further if we were to consider different preprocessing operations. If we decided, for practical reasons, to conduct a subset of these tests, the metadata can be incomplete.

Some experiments may result in failure

Some experiments result in failure or do not terminate in a given time budget. Failures do sometimes occur in the execution of base-level algorithms. In some cases, it is possible to recover from such failures (e.g., insufficient memory), but there are cases where the performance of an algorithm on a dataset cannot be obtained (e.g., software bug). If an algorithm fails to run on a dataset, its performance is not quantifiable, although it is not missing. One approach to deal with this issue is to penalize such algorithms in some way. The simplest strategy could be to use the appropriate default strategy, which is normally based on simple statistics of the data. In classification, this would be predicting the most frequent class, and in regression, it would be predicting the mean target value. The estimated performance of this default strategy would be used to replace the performance of the algorithms that fail.

New dataset(s) have been introduced

It can be expected that extending our set up with a new dataset and the corresponding metadata could somewhat improve the ability of the system to provide good recommendations for the new problems. Therefore, it is important to carry out such extensions whenever new datasets become available. However, whenever new datasets have been introduced into our set up, the metadata becomes incomplete. This requires that all the available algorithms are run on the new dataset. This may be computationally rather expensive, particularly if the dataset is large and the number of alternatives to test is large too (the number of algorithms or workflows and its variants).

When a new base-level algorithm becomes available, it is necessary to update the metaknowledge so that the system could consider it in the recommendations provided. For that purpose, the metadata describing the performance of algorithms on known datasets (i.e., meta-examples) must be extended with information concerning the new algorithm. It is therefore necessary to run it on those datasets, which may require significant computational effort. One approach is to run all experiments off-line and update the metadata only after the results become available.

Using estimates instead of real values

The process of obtaining test results is computationally expensive, and so different strategies exist to try to alleviate the problem. This can be done in two ways. Firstly, it can be used to generate *estimates of the performance* and use them as substitutes for the true performance until the true performance becomes available. Curiously enough, this idea was mentioned in the first edition of this book. A similar idea was followed in the area of hyperparameter optimization, where the estimates were provided by *surrogate models* (see Chapter 6 for details).

8.8.2 Is it necessary to have complete metadata?

To answer this question satisfactorily, it is necessary to consider different systems and determine what this system can (cannot) do when the metadata is incomplete. Not many systems are accompanied by such study, so it is difficult to provide an all-encompassing answer to this question here.

We will limit this exposition to one study carried out by Abdulrahman et al. (2018) which involves the average ranking method AR*, discussed in Chapter 2. The authors have shown that the performance of this method was not affected even by 50% of omissions in the metadata. However, as was shown, it was necessary to modify the aggregation method so that it would cater for incomplete rankings.

8.8.3 Does the order of tests matter?

In Chapter 6, we discussed various approaches including random search on the one hand and various "more informed" methods, such as sequential model-based optimization (SMBO), on the other hand. Various authors have shown that the more informed methods achieve, in general, better results than uninformed ones (e.g., random search). This is done by reordering the tests on the basis of information that has been gathered in previous tests. Further details on this topic are given in the following section.

8.9 Exploiting Strategies from Multi-armed Bandits to Schedule Experiments

This section addresses some ideas for how to obtain performance results of algorithms on earlier datasets. As argued, there is a large number of possibilities, and smart planning might improve the quality of the metadata. The process of gathering test results can be compared to the process of gathering knowledge about different "arms" in multi-armed bandit (MAB) problems. The MAB problem includes a gambler whose aim is to decide which arm of a k-slot machine to pull to maximize his total reward in a series of trials (Katehakis and Veinott, 1987; Vermorel and Mohri, 2005). The aim is to find a good compromise between *exploration* (i.e., examining different arms) and *exploitation* (using the best arm(s) for the target problem).

Many real-world learning and optimization problems can be modeled using this paradigm, and algorithm selection/configuration is one of them. Different workflows (pipelines) configurations can be compared to different arms. In consequence, the solution to the MAB problem can inspire new effective solutions to the problem of algorithm selection/configuration.

8.9.1 Some concepts and strategies of MAB

One important notion in this area is the concept of *reward*, which is received after an arm has been pulled. The difference from the optimal is often referred to as the *regret* or *loss*. Typically, the aim is to maximize the accumulated reward, which is equivalent to minimizing the accumulated loss, as different arms have been pulled. Table 8.2 shows the correspondence of some terms used in the area of MAB and the terms used in the area of algorithm selection/configuration and metalearning. Some common MAB strategies are discussed in the following subsections.

Table 8.2: Correspondence of terms in MAB and this book

N levers	N algorithms
Pulling a lever	Evaluating an algorithm (configuration, workflow)
Reward	Performance (e.g., accuracy)
Regret	Loss
Accumulated regret	Area under the loss curve
Horizon	Time budget
Contextual MAB	Metalearning problem that exploits features

Strategy *ϵ-greedy*

This strategy chooses a random lever with frequency ϵ, where $\epsilon \in (0, 1)$ is set by the user. For the remaining $(1 - \epsilon)$ proportion of cases, the best arm is chosen.

Strategy *ϵ-first*

This strategy can be seen as a variant of *ϵ-greedy*. It carries out all the exploration at the beginning. For a given number $\epsilon T \in N$ of rounds, the levers are randomly pulled during the T first rounds. This pure exploration phase is followed by an exploitation phase. That is, during the remaining $(1 - \epsilon)T$ rounds, the lever with the highest estimated mean is selected.

Strategy *ϵ-decreasing*

This variant is similar to *ϵ-greedy* except that the value of ϵ decreases as the experiment progresses, resulting in high exploration in the initial rounds, while exploitation is preferred later. This can be captured by $\epsilon_t = min(1, \epsilon_t/t)$, where t is the index of the current round.

Probability matching method (SoftMax)

The SoftMax strategy was discussed already by Luce (1959), but many variants of this method were described later. This strategy exploits a random choice from Gibbs distribution. The lever k is chosen with probability

$$p_k = e^{\hat{\mu}_k/\tau} / \sum_{i=1}^{n} e^{\hat{\mu}_i/\tau} \tag{8.4}$$

where $\hat{\mu}_k$ is the estimated mean of the rewards brought by pulling lever k and τ is a parameter called the *temperature*, which is set by the user.

This method belongs to the so-called *probability matching methods*, as the choice is made according to a probability distribution, reflecting how likely the choice is optimal. The SoftMax strategy is sometimes also called *Boltzmann exploration*. One variant of SoftMax is referred to as *decreasing SoftMax*. In this variant, the temperature decreases with the number of rounds played.

Interval estimation and upper confidence bound (UCB) methods

The interval estimation method attributes to each lever an *optimistic reward estimate* within a certain confidence interval and then chooses the lever with the *highest optimistic mean* (Kaelbling, 1993). Unobserved or infrequently observed levers have an over-valued reward mean, which stimulates further exploration. The more frequently a lever is pulled, the closer its optimistic reward estimate will be to the true reward mean. The original approach of Kaelbling (1993) was applied to Boolean awards. Vermorel and Mohri (2005) have applied it to real values and assumed that rewards are normally distributed. The upper bound estimate is based on that assumption. Assuming that a lever is observed n times with $\hat{\mu}$ as the empirical mean and $\hat{\sigma}$ as the empirical standard deviation, the upper bound is defined by

$$u_\alpha = \hat{\mu} + \frac{\hat{\sigma}}{\sqrt{n}} c^{-1}(1-\alpha), \tag{8.5}$$

where c is the cumulative normal distribution function.

The upper confidence bound algorithms (Agrawal, 1995; Auer et al., 2002; Lai and Robbins, 1985) work in a similar way. Specifically, in each trial t, these algorithms estimate both the mean payoff $|\hat{\mu}_{t,a}|$ of each arm a as well as a corresponding confidence interval $c_{t,a}$, so that $|\hat{\mu}_{t,a} - \mu_a| < c_{t,a}$ holds with high probability. They then select the arm that achieves the highest upper confidence bound (UCB), namely $a_t = argmax_a(|\hat{\mu}_{t,a} + c_{t,a})$.

Pricing strategies (POKER)

Pricing strategies establish a price for each lever. The approach of Vermorel and Mohri (2005), called *Price of Knowledge and Estimated Reward* (Poker), relies on three main ideas: pricing uncertainty capturing value of information, the lever distribution and horizon.

The idea behind pricing uncertainty is to assign a value to the knowledge gained while pulling a particular lever. The notion of *value of information* or *exploration bonuses* has been studied in various domains and in the bandit literature (Meuleau and Bourgine, 1999). The objective is to quantify the uncertainty in the same units as the rewards.

The second idea is that the properties of unobserved levers could potentially be estimated, to a certain extent, from the levers already observed. This is particularly useful when there are many more levers than rounds.

The third observation is that the strategy should explicitly take into account the horizon H, that is, the number of rounds that remain to be played. The amount of exploration clearly depends on H. If, for instance, the play is reaching the horizon (H=1), it does not make sense to explore more, and the strategy should be to rely on pure exploitation, i.e., choosing the lever with the highest estimated reward. So the horizon value affects the estimate of the value of the knowledge that could be acquired.

Contextual-bandit problem

Some researchers have introduced the so-called *contextual-bandit problem*, where different arms are characterized by features. These are represented in the form of a feature vector (context vector).

The agent uses these context vectors together with the rewards of the arms played in the past to make the choice of the arm to play in the current iteration. Over time, the learner's aim is to collect enough information about how the context vectors and rewards relate to each other, so that it can predict the next best arm to play by looking at the feature vectors (Langford and Zhang, 2007).

This approach has been applied, for instance, to personalized recommendation of news articles (Li et al., 2010). The learning algorithm sequentially selects articles for users based on contextual information about the users and articles, while simultaneously adapting its article selection strategy based on user-click feedback.

Contextual approaches can be compared to metalearning approaches that exploit dataset features and tests.

8.10 Discussion

In this chapter, we addressed two components from the framework of Rice (1976), i.e., the problem space and the algorithm space. The algorithm space is commonly known as the configuration space. In traditional metalearning systems, the configuration space is often a discrete set of algorithms or workflows, whereas in AutoML systems, often a continuous set is considered. While these representations are similar on some level, they also require different approaches and invoke different biases. Both of these representations have been studied extensively.

For discrete spaces, Abdulrahman et al. (2019) aimed to reduce the configuration space, so that the search for the best algorithm or workflow would converge faster. For continuous spaces, various authors have tried to determine which hyperparameters are generally important.

Recently, the neural architecture search community started to address the problem of search space construction (Yu et al., 2020; Yang et al., 2020). In order to keep this chapter focused towards traditional metalearning approaches, we did not go into details here, but the interested reader might find the references a good starting point.

Finally, this chapter looked at various aspects regarding the problem space, in particular, which datasets should be included in the metadata, how many experiments should be carried out, and in which order, and terminated with the discussion of the multi-armed bandit methods, which suggest some answers to the last issue.

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.

Agrawal, R. (1995). Sample mean based index policies with O(log n) regret for the multi-armed bandit problem. *Advances in Applied Probability*, 27(4):1054–1078.

Aha, D. W. (1992). Generalizing from case studies: A case study. In Sleeman, D. and Edwards, P., editors, *Proceedings of the Ninth International Workshop on Machine Learning (ML92)*, pages 1–10. Morgan Kaufmann.

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

Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3):235–256.

Biedenkapp, A., Lindauer, M., Eggensperger, K., Fawcett, C., Hoos, H., and Hutter, F. (2017). Efficient parameter importance analysis via ablation with surrogates. In *Thirty-First AAAI Conference on Artificial Intelligence*, pages 773–779.

Bischl, B., Casalicchio, G., Feurer, M., Gijsbers, P., Hutter, F., Lang, M., Mantovani, R. G., van Rijn, J. N., and Vanschoren, J. (2021). OpenML benchmarking suites. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, NIPS'21.

Brazdil, P. and Cachada, M. (2018). Simplifying the algorithm portfolios with a method based on envelopment curves (working notes).

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.

Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

Cohen, W. W. (1994). Grammatically biased learning: Learning logic programs using an explicit antecedent description language. *Artificial Intelligence*, 68(2):303–366.

Davies, T. R. and Russell, S. J. (1987). A logical approach to reasoning by analogy. In McDermott, J. P., editor, *Proceedings of the 10th International Joint Conference on Artificial Intelligence, IJCAI 1987*, pages 264–270, Freiburg, Germany. Morgan Kaufmann.

De Raedt, L. and Dehaspe, L. (1997). Clausal discovery. *Machine Learning*, 26:99–146.

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

Došilovič, F., Brčič, M., and Hlupič, N. (2018). Explainable artificial intelligence: A survey. In *Proc. of the 41st Int. Convention on Information and Communication Technology, Electronics and Microelectronics MIPRO*.

Fawcett, C. and Hoos, H. (2016). Analysing differences between algorithm configurations through ablation. *Journal of Heuristics*, 22(4):431–458.

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., and Hutter, F. (2015). Efficient and robust automated machine learning. In Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems 28*, NIPS'15, pages 2962–2970. Curran Associates, Inc.

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J. T., Blum, M., and Hutter, F. (2019). Auto-sklearn: Efficient and robust automated machine learning. In Hutter, F., Kotthoff, L., and Vanschoren, J., editors, *Automated Machine Learning: Methods, Systems, Challenges*, pages 113–134. Springer.

Fogelman-Soulié, F. (2006). Data mining in the real world: What do we need and what do we have? In Ghani, R. and Soares, C., editors, *Proceedings of the Workshop on Data Mining for Business Applications*, pages 44–48.

Fréchette, A., Kotthoff, L., Rahwan, T., Hoos, H., Leyton-Brown, K., and Michalak, T. (2016). Using the Shapley value to analyze algorithm portfolios. In *30th AAAI Conference on Artificial Intelligence*.

Gordon, D. and desJardins, M. (1995). Evaluation and selection of biases in machine learning. *Machine Learning*, 20(1/2):5–22.

Hettich, S. and Bay, S. (1999). The UCI KDD archive. `http://kdd.ics.uci.edu`.

Hilario, M. and Kalousis, A. (2000). Quantifying the resilience of inductive classification algorithms. In Zighed, D. A., Komorowski, J., and Zytkow, J., editors, *Proceedings of the Fourth European Conference on Principles of Data Mining and Knowledge Discovery*, pages 106–115. Springer-Verlag.

Hirsh, H. (1994). Generalizing version spaces. *Machine Learning*, 17(1):5–46.

Hutter, F., Hoos, H., and Leyton-Brown, K. (2013). Identifying key algorithm parameters and instance features using forward selection. In *Proc. of International Conference on Learning and Intelligent Optimization*, pages 364–381.

Hutter, F., Hoos, H., and Leyton-Brown, K. (2014). An efficient approach for assessing hyperparameter importance. In *Proceedings of the 31st International Conference on Machine Learning*, ICML'14, pages 754–762.

Jorge, A. M. and Brazdil, P. (1996). Architecture for iterative learning of recursive definitions. In De Raedt, L., editor, *Advances in Inductive Logic Programming*, volume 32 of *Frontiers in Artificial Intelligence and applications*. IOS Press.

Kaelbling, L. P. (1993). *Learning in Embedded Systems*. MIT Press.

Katehakis, M. N. and Veinott, A. F. (1987). The multi-armed bandit problem: Decomposition and computation. *Mathematics of Operations Research*, 12(2):262–268.

Keogh, E. and Folias, T. (2002). The UCR time series data mining archive. http://www.cs.ucs.edu/˜eamonn/TSDMA/index.html. Riverside CA. University of California – Computer Science & Engineering Department.

Kramer, S. and Widmer, G. (2001). Inducing classification and regression trees in first order logic. In Džeroski, S. and Lavrač, N., editors, *Relational Data Mining*, pages 140–159. Springer.

Lai, T. L. and Robbins, H. (1985). Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics*, 6(1):4–22.

Langford, J. and Zhang, T. (2007). The epoch-greedy algorithm for contextual multi-armed bandits. In *Advances in Neural Information Processing Systems 20*, NIPS'07, page 817–824. Curran Associates, Inc.

Lee, J. W. and Giraud-Carrier, C. (2011). A metric for unsupervised metalearning. *Intelligent Data Analysis*, 15(6):827–841.

Li, L., Chu, W., and Schapire, R. E. (2010). A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the International Conference on World Wide Web (WWW)*.

Luce, D. (1959). *Individual Choice Behavior*. Wiley.

Meuleau, N. and Bourgine, P. (1999). Exploration of multi-state environments: Local measures and back-propagation of uncertainty. *Machine Learning*, 35(2):117–154.

Miettinen, K. (1999). *Nonlinear Multiobjective Optimization*. Springer.

Mitchell, T. (1977). *Version spaces: A candidate elimination approach to rule learning*. PhD thesis, Electrical Engineering Department, Stanford University.

Mitchell, T. (1980). The need for biases in learning generalizations. Technical Report CBM-TR-117, Rutgers Computer Science Department.

Mitchell, T. (1982). Generalization as Search. *Artificial Intelligence*, 18(2):203–226.

Mitchell, T. (1990). The need for biases in learning generalizations. In Shavlik, J. and Dietterich, T., editors, *Readings in Machine Learning*. Morgan Kaufmann.

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

Morik, K., Wrobel, S., Kietz, J., and Emde, W. (1993). *Knowledge Acquisition and Machine Learning: Theory, Methods and Applications*. Academic Press.

Muñoz, M., Villanova, L., Baatar, D., and Smith-Miles, K. (2018). Instance Spaces for Machine Learning Classification. *Machine Learning*, 107(1).

Peterson, A. H. and Martinez, T. (2005). Estimating the potential for combining learning models. In *Proc. of the ICML Workshop on Meta-Learning*, pages 68–75.

Probst, P., Boulesteix, A.-L., and Bischl, B. (2019). Tunability: Importance of hyperparameters of machine learning algorithms. *Journal of Machine Learning Research*, 20(53):1–32.

Prudêncio, R. B. C., Soares, C., and Ludermir, T. B. (2011). Combining meta-learning and active selection of datasetoids for algorithm selection. In Corchado, E., Kurzyński, M., and Woźniak, M., editors, *Hybrid Artificial Intelligent Systems. HAIS 2011.*, volume 6678 of *LNCS*, pages 164–171. Springer.

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

Russell, S. and Grosof, B. (1990a). Declarative bias: An overview. In Benjamin, P., editor, *Change of Representation and Inductive Bias*. Kluwer Academic Publishers.

Russell, S. and Grosof, B. (1990b). A sketch of autonomous learning using declarative bias. In Brazdil, P. and Konolige, K., editors, *Machine Learning, Meta-Reasoning and Logics*. Kluwer Academic Publishers.

Scott, P. D. and Wilkins, E. (1999). Evaluating data mining procedures: techniques for generating artificial data sets. *Information & Software Technology*, 41(9):579–587.

Sharma, A., van Rijn, J. N., Hutter, F., and Müller, A. (2019). Hyperparameter importance for image classification by residual neural networks. In Kralj Novak, P., Šmuc, T., and Džeroski, S., editors, *Discovery Science*, pages 112–126. Springer International Publishing.

Silverstein, G. and Pazzani, M. J. (1991). Relational clichés: Constraining induction during relational learning. In Birnbaum, L. and Collins, G., editors, *Proceedings of the Eighth International Workshop on Machine Learning (ML'91)*, pages 203–207, San Francisco, CA, USA. Morgan Kaufmann.

Snoek, J., Swersky, K., Zemel, R., and Adams, R. (2014). Input warping for Bayesian optimization of non-stationary functions. In Xing, E. P. and Jebara, T., editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *ICML'14*, pages 1674–1682, Bejing, China. JMLR.org.

Soares, C. (2009). UCI++: Improved support for algorithm selection using datasetoids. In *Proceedings of Pacific-Asia Conference on Knowledge Discovery and Data Mining*.

Sobol, I. M. (1993). Sensitivity estimates for nonlinear mathematical models. *Mathematical Modelling and Computational Experiments*, 1(4):407–414.

van Rijn, J. N. and Hutter, F. (2018). Hyperparameter importance across datasets. In *KDD '18: The 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM.

Vanschoren, J. and Blockeel, H. (2006). Towards understanding learning behavior. In *Proceedings of the Fifteenth Annual Machine Learning Conference of Belgium and the Netherlands*.

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

Vermorel, J. and Mohri, M. (2005). Multi-armed bandit algorithms and empirical evaluation. In *Machine Learning: ECML-94, European Conference on Machine Learning, LNAI 3720)*. Springer.

Xu, L., Hutter, F., Hoos, H., and Leyton-Brown, K. (2012). Evaluating component solver contributions to portfolio-based algorithm selectors. In Cimatti, A. and Sebastiani, R., editors, *Theory and Applications of Satisfiability Testing – SAT 2012*, pages 228–241. Springer Berlin Heidelberg.

Yang, A., Esperança, P. M., and Carlucci, F. M. (2020). NAS evaluation is frustratingly hard. In *International Conference on Learning Representation*, ICLR 2020.

Yu, K., Sciuto, C., Jaggi, M., Musat, C., and Salzmann, M. (2020). Evaluating the search phase of neural architecture search. In *International Conference on Learning Representation*, ICLR 2020.