If you try and take a cat apart to see how it works, the first thing you have on your hands is a non-working cat.

Douglas Adams

Chapter One

Meta-Learning

This chapter covers the practice and state-of-the-art in the field of meta-learning. We start by providing a general definition, and then we move on to discuss three increasingly hard meta-learning settings in Section 1.2. After this, we focus on the setting of algorithm selection and outline a general framework in Section 1.3, which we use to highlight the key questions and proposed solutions in each component of the general meta-learning process in the subsequent sections. Finally, we conclude with a summary of the state-of-the-art, open issues and suggestions for future research in Section 1.10.

1.1 Introduction

1.1.1 Definition

A quite broad definition of meta-learning is given by Brazdil et al. (2009), which we present here in a slightly adjusted form[1]:

> Metalearning is the study of principled methods that exploit *meta-knowledge* to improve or build self-improving learning systems through *adaptation* of machine learning and data mining *processes*.

[1] We replaced the words 'to obtain efficient models and solutions' with 'to improve or build self-improving learning systems' to emphasize that we want to improve upon learning systems which do not employ the said meta-knowledge.

Clarification of the definition's key concepts is a useful starting point:

Metaknowledge is *knowledge about learning processes* acquired through experience with past learning episodes. This knowledge can comprise complete patterns in learning behavior discovered either automatically or by machine learning experts. We will refer to this type of knowledge as *meta-models*, models of a learner's ability to correctly model the data. Mostly, however, it consists of *characterizations*, measurable properties of datasets and learning algorithms, such as the number of attributes in a dataset or the relative speed of an algorithm, combined with measurable properties of the models built by learning algorithms, such as their accuracy or size (e.g. the size of a decision tree). To avoid any confusion, we shall use the term *meta-data* for such characterizations of dataset and learning algorithms. Like any knowledge discovery problem, it is the usefulness and quality of this meta-data that will ultimately decide the quality of the learned meta-models.

Processes These machine learning and data mining processes can be any part of the knowledge discovery process. We may want to use metaknowledge to only propose useful data preprocessing and transformation techniques, to select a learning algorithm or even just a part of it, e.g. a parameter setting or an internal component, or even to suggest entire *workflows* of techniques (from collecting the data to postprocessing the produced model).

Adaptation If we are faced with a new learning problem, adapting these processes means selecting, adding or combining the most suitable components. For instance, many learning algorithms may have to be combined to model the data better. Alternatively, in case a (partial) workflow exists, it also means to replace, remove or change certain components based on the metaknowledge we have.

In short, meta-learning monitors the automatic learning process *itself*, in the context of the learning problems it encounters, and tries to adapt its behavior to perform better. Whereas a normal learning algorithm stops after modeling the data, a meta-learning system operates continuously over many different problems. Indeed, any learning system can only learn to adapt itself if it studies the stream of problems it encounters.

To distinguish between the 'basic' form of machine learning and meta-learning, we shall call the former *base-learning* (Brazdil et al. 2009). In base-learning, the learning process consists of increasingly refining an hypothesis with an increasing number of examples. Nevertheless, if we offer the learning system the exact same examples again, it will produce the exact same hypothesis, unless it contains a randomized component. In any case, it will never learn to improve its performance by dynamically adapting some internal components to fit the data better.

1.2 Flavors of meta-learning

There are many different types of metaknowledge, as well as ways to use it to improve learning systems. In this section, we provide a short outline of three meta-learning settings, amounting to increasingly hard meta-learning problems, while focussing on *why* these approaches improve the speed and/or accuracy of the underlying base-learners. First, in Section 1.2.1, we try to learn from models that have been built previously by other algorithms on the *same* data, but which we want to improve upon. Next, in Section 1.2.2, we try to learn from models built on *similar* data, usually pertaining to a similar task, and try to facilitate learning by reusing what we have learned before. And finally, in Section 1.2.3, we try to learn from the performance of other algorithms on many *other* kinds of data, hoping to generalize their learning behavior to be able to recommend useful learning algorithms for future learning problems.

This is by no means meant to be an exhaustive survey, a more complete discussion of the various settings in which meta-learning is applied can be found in Brazdil et al. (2009), Vilalta and Drissi (2002b) and Vilalta et al. (2005).

1.2.1 Ensemble learning

1.2.1.1 Definition

An ensemble of learning algorithms is a set of algorithms, trained on the same problem, whose individual hypotheses are combined in some way to predict the outcome of new examples (Dietterich 1997; Dietterich 2000). Ensemble learning works by trying to build models that are *sufficiently different* from each other, each supposed to be better at modeling some parts of the data than other parts, so that, by combining them, we get a better fit for the entire dataset. These techniques are also known as *model combination.*

1.2.1.2 Why ensembles work

There are two basic approaches to ensemble learning (Brazdil et al. 2009). The first approach causes the base-learner's models to be different by feeding different subsets of the data to the same base-learner. The meta-algorithm combining the ensuing predictions can be very simple, such as simply taking the majority prediction. The second approach exploits differences among base-learners, and thus feeds the same data to different learning algorithms, after which a meta-learner tries to learn how the individual predictions should be combined to get a better prediction. The first approach works especially well for *unstable* learning algorithms: those whose models undergo major changes in response to small changes in the training data, such as decision trees, neural networks and rule learners. The second approach is useful for *stable* algorithms, such as linear regression, nearest neighbor, and linear threshold algorithms.

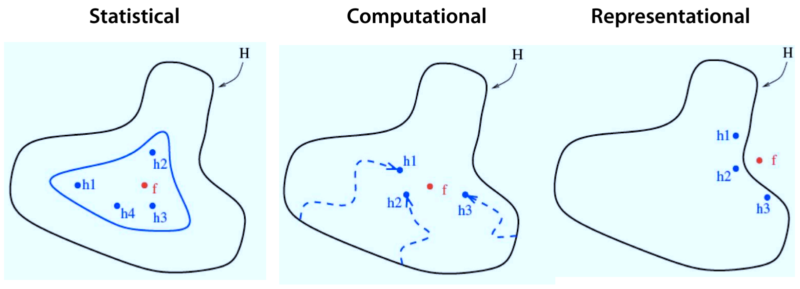

Figure 1.1: Depiction of three reasons why an ensemble learner could perform better than an individual base-learner. Adapted from Dietterich (2000).

For now, we only consider the first type of ensemble learning. To understand *why* such ensembles improve learning performance, take a look at Figure 1.1, in which the outer curve denotes the hypothesis space H of the individual base-learner, and the point labeled f is the true hypothesis we wish to find. There are three fundamental reasons why an ensemble would perform better than the individual base-learner (Dietterich 2000):

Statistical Depicted on the left, this effect is most pronounced when there are too little training examples to converge to f, or simply too much noise in the training data. Each of the base learners will stop short of finding the true hypothesis, but will find an hypothesis that is sufficiently close to it (denoted by the inner curve). By constructing an ensemble of the produced models, we can smooth out the individual predictions, which is presumably less likely to be wrong than any individual prediction.

Computational Since algorithms perform a heuristic local search of H, e.g. the greedy splitting approach in decision tree learners, they may get stuck in local optima. An ensemble constructed by running the local search from many different starting points may provide a better approximation to the true unknown function than any of the individual classifiers, as shown in the middle of Figure 1.1.

Representational The third reason is depicted on the right of Figure 1.1 and concerns the base-learner's representational bias. In most cases, f cannot be represented by any hypothesis in H, at least not with a finite number of training examples. However, by forming weighted sums of hypotheses drawn from H, it is possible to implicitly 'expand' the space of representable functions.

1.2.1.3 Ensemble learning techniques

The first type of ensemble learning can be exemplified by the following approaches, for a more complete overview, see Dietterich (2000):

Bagging Short for bootstrap aggregation, bagging (Breiman 1996) presents the base-learner with i training sets, each consisting of a sample of m training examples drawn randomly with replacement[2], called a *bootstrap*, from the original training set of n items. Usually, $m = n$. To provide the final prediction, a majority vote is held among the predictions of the i produced models. It thus samples from the space of all possible hypotheses with a bias toward hypotheses that give good accuracy on the training data. Consequently, their main effect will be to address the statistical problem and, to a lesser extent, the computational and representational. By combining many models one gets a virtual model that would probably not have been found otherwise (unless perhaps, more data was available). Especially in noisy datasets, being primarily a statistical problem, Bagging proves to be very effective.

Randomization Another way to produce sufficiently diverse models is to introduce randomness in the model construction process. *Random Forests* (Breiman 2001), for instance, combine bagging with a slight randomization of the splits made by decision tree learners, causing the trees to be more unstable, and making the ensemble more diverse at the risk of generating lower-quality trees. Randomization is especially useful on very large datasets because those tend to contain many similar examples, making the bootstrap replicates, and thus the resulting models, very similar.

Boosting Boosting (Freund and Schapire 1996) works by adding a weight to each of the training examples. In each iteration, the base-learner is invoked to minimize the weighted error on the training set, and it returns an hypothesis h. The weighted error of h is then used to update the weights on the training examples, placing more weight on training examples that were misclassified by h and less weight on examples that were correctly classified. In a sense, this creates progressively more difficult learning problems for the base-learner. In the end, each model is weighted according to its performance and a weighted vote is performed to obtain the final prediction. It directly attacks the representational problem, by forcing the learner to consider alternative hypotheses. In high-noise cases however, boosting actually pushes the learner to model the noise, causing overfitting.

See Dietterich (2000) for a more complete overview, including cross-validated committees (Parmanto et al. 1996) and error-correcting output coding (Dietterich and Bakiri 1995). It is worth noting that the base-learners used for these techniques are often of the same type, such as decision trees learners.

The second type of ensemble learning hopes to profit mostly from the representational aspect of Section 1.2.1.2. It amounts to modeling the data several times with different biases, arriving at a new bias that hopefully fits the data better. It can be exemplified by the following approaches:

Stacking Short for stacked generalization, stacking (Wolpert 1992) takes n base-learners $A_1, ..., A_n$ and runs them on the same dataset D to produce n

[2]This means that the same example can be selected several times.

hypotheses $h_1, ..., h_n$. Then, a new dataset T is constructed by replacing (or appending) the features of D with the predictions of each of the hypotheses, and offered to another algorithm A_{meta} which builds a meta-model mapping the predictions of the base-learners to the target of D. One assumes that several parts of the data are best modeled by different base-learners, and that this relation can be learned by algorithm A_{meta}.

Cascade generalization Instead of learning hypotheses $h_1, ..., h_n$ in parallel, we can also learn h_1 first, append its predictions to dataset D as a new feature, and then pass this new dataset on to the next base-learner, and so on, until the final base-learner A_n tries to learn from D together with the $n - 1$ prior hypotheses (Gama and Brazdil 2000).

Other approaches are cascading (Kaynak and Alpaydin 2000), delegating (Ferri et al. 2004), arbitrating (Ortega et al. 2001) and meta-decision trees (Todorovski and Džeroski 2003). See Brazdil et al. (2009) for an overview.

As a final note, we should mention that sometimes, these techniques are viewed as a *postprocessing* step of the knowledge discovery process, since each of the base-learners already produces a model of the given data. In this text, however, we will follow a more popular view and treat ensembles as learning algorithms in their own right, with their own bias (influenced by which base-learner(s) they use), and their own suitability for use on specific problems.

1.2.2 Transfer learning

1.2.2.1 Definition

In a typical learning scenario, the learner has to start from scratch even if a new task (e.g. recognizing types of galaxies) is very similar to a previous task (e.g. recognizing types of stars). In transfer learning, one wishes to reuse experience on one task to be better at learning a similar one (Thrun and Pratt 1998; Caruana 1997; Rendell et al. 1987). The metaknowledge thus consists of previously learned models, which the learner is expected to benefit from.

Transfer learning usually works by *forcing* the hypotheses generated by learning on different (but similar) tasks to be very similar or to share a common structure. The commonalities between those hypotheses thus represent, implicitly or explicitly, what all these tasks have in common, which we hope to use to improve our learning of other similar tasks.

Depending on the stage of the learning process in which this experience is used, we can differentiate between two types of transfer learning (Thrun and Pratt 1998). In *representational transfer* knowledge is first generated in one task before being exploited in the next task, and in *functional transfer*, also known as *multitask learning*, various tasks are learned in parallel (Caruana 1997).

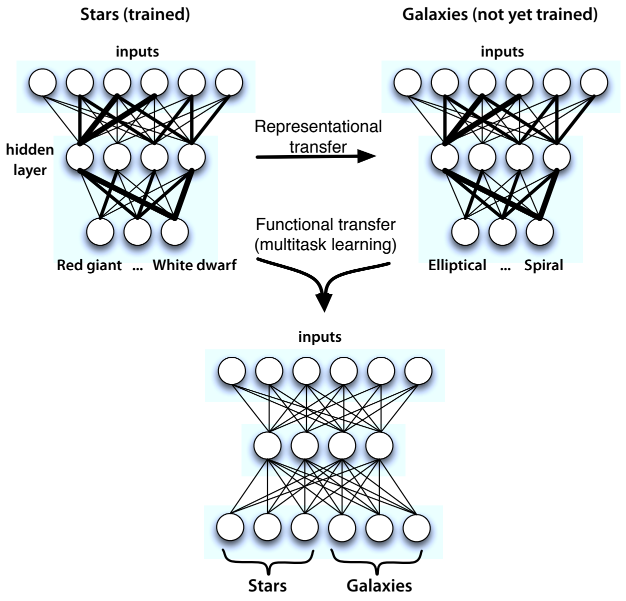
Stars (trained)
Galaxies (not yet trained)
inputs
hidden layer
Red giant ... White dwarf
Representational transfer
Functional transfer (multitask learning)
inputs
Elliptical ... Spiral
inputs
Stars
Galaxies

Figure 1.2: Two different types of transfer learning in neural networks. Adapted from Brazdil et al. (2009).

1.2.2.2 Example

As an example, we illustrate how transfer learning can be achieved with artificial neural networks. An artificial neural network is a network of interconnected nodes, which mimics certain aspects of the structure of our brain. The nodes are organized in different layers, in which the first layer has a node for each input feature, and the final layer has a node for each required output. In between are a number of *hidden layers* in which each neuron is connected to a number of nodes of the previous and subsequent layer, shown in the top left of Figure 1.2. Each connection has a weight (symbolized by the thickness of the lines), which defines how much of the activation of the previous node is passed on to the next. Without going into too much detail, the network is typically trained through *backpropagation*, in which the feature values of examples are presented to the corresponding input nodes, and errors in the output are corrected by adjusting the weights of the connections to previous nodes.

The structure and weights of a trained neural network can be seen as an (uninterpretable) model of the data it was trained on and can be used to predict the outcome of future observations. Neural networks are exceptionally useful for representational transfer learning since the final set of weights obtained by training the network on a specific problem (called the source network) can be used as a good *initialization* of the network for a similar problem (the target network) (Thrun and Mitchell 1995; Baxter 1996). This is shown by the first arrow in Figure 1.2: the exact same weights of the source neural network trained for recognizing types of stars are used to initialize the target neural network meant for the recognition of types of galaxies. The next step would be to train the second network with real examples of galaxies, hoping that a better initialization of a neural network leads to much faster convergence to the target concept, so we get a better fit with much fewer training examples. Depending on the similarity of the tasks, one might modify the the source network first before transferring it, an approach called *non-literal transfer* (Sharkey and Sharkey 1993).

However, neural networks can also be used for multitask learning (Silver and Mercer 1996). In this scenario, nodes are shared by different tasks during training, as shown on the bottom of Figure 1.2. We construct a single neural network supposed to classify any type of stellar object as the right type of star or galaxy. If the two tasks have common properties, the same internal nodes can serve to represent the common sub-concepts simultaneously for both tasks. Besides from having to learn only once instead of twice, we assume that training many similar tasks in parallel on a single neural network induces information that would not have been induced when learning the tasks separately.

1.2.2.3 Other techniques

Other techniques have been proposed for transfer learning with kernel methods (Evgeniou et al. 2006; Evgeniou and Pontil 2004), parametric Bayesian models (Rosenstein et al. 2005; Raina et al. 2005; Bakker and Heskes 2003), Bayesian networks (Niculescu-Mizil and Caruana 2005), clustering (Thrun and Pratt 1998) and reinforcement learning (Hengst 2002; Dietterich et al. 2002).

1.2.3 Modeling learning behavior

In this setting, the metaknowledge consists of a large collection of meta-data about different learning problems $P_1, ..., P_n$, different learning algorithms $A_1, ...,$ A_n, and the performance, under different parameter settings, of an algorithm A_i on a problem P_i. From this information, we wish to learn the relationship between the properties of the data and the performance of the learning algorithms. If we can model this relationship, we can predict which algorithms are worth considering when faced with a certain problem.

This setting will be the focus of this thesis and the remainder of this chapter, and when we speak of meta-learning from now on, we will always mean this particular setting. We will still discuss ensembles in great length, but we will handle them as complex types of algorithms, studying their learning behavior on different kinds of data, and comparing them with other learning algorithms, simple or complex.

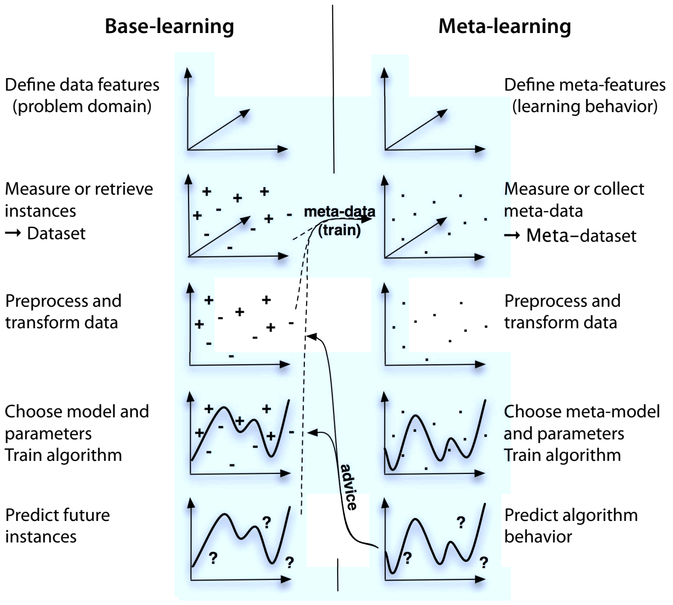
Base-learning
Meta-learning
Define data features (problem domain)
Define meta-features (learning behavior)
Measure or retrieve instances → Dataset
meta-data (train)
Measure or collect meta-data → Meta-dataset
Preprocess and transform data
Preprocess and transform data
Choose model and parameters Train algorithm
advice
Choose meta-model and parameters Train algorithm
Predict future instances
Predict algorithm behavior

Figure 1.3: A comparison of base-learning and meta-learning.

1.2.3.1 Base-learning versus meta-learning

The interplay between base-learning and meta-learning can be described as shown in Figure 1.3. They both consist of the typical knowledge discovery steps. On the base-level, we need to select the data features we think are useful for discovering a pattern, then we collect data points (or retrieve them from a database), we preprocess and transform the data, e.g. to remove redundant features, we choose and apply a learning algorithm to model the data, and finally, we use the produced model (directly or indirectly) to make predictions about the outcome of future observations. On the meta-level, we need to define a set of meta-features: characterizations of datasets, learning algorithms and the produced models we think are useful to learn how to handle future problems better. We then collect the meta-data (see the dashed lines) from a large number of base-learning examples: data-characteristics from the original and preprocessed datasets, as well as characteristics of the produced models such as the algorithm that built them and its parameter settings, their accuracy, size and the time it took to build them. Next, we need to preprocess the meta-data and choose and train a meta-model to discover patterns in this data. Finally, we use these meta-models to provide advice: we measure the same data-characteristics on new problems, and use them to predict which preprocessing steps might be useful, which learning algorithms will perform well on it, and even to propose entire workflows for handling the new problem.

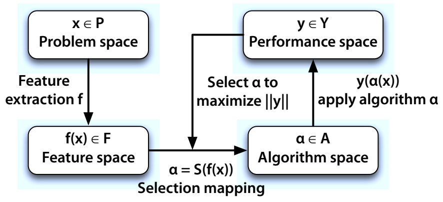

Figure 1.4: Rice's framework for algorithm selection. Adapted from Smith-Miles (2008a).

1.3 An algorithm selection framework

In the remainder of this chapter, we shall focus on the latter type of meta-learning. We will first outline a framework for algorithm selection, in which we can cast prior work in this area.

1.3.1 Rice's framework

The algorithm selection problem is older than meta-learning itself. It was first described in Rice (1976), which presented a formal abstract model that can be used to explore the question: "With so many available algorithms, which one is likely to perform best on my problem and specific instance?"

The model is shown in Figure 1.4, and contains four main components:

- The problem space P, in our case the space of all learning problems
- The feature space F of all measurable characteristics of each of the problems in P, calculated by a feature extraction process f
- The algorithms space A: the set of all base-level learning algorithms
- The performance space Y representing the mapping of each algorithm A to a set of performance metrics

The algorithm selection problem can be then be formally stated as follows:

> For a given problem instance $x \in P$, with features $f(x) \in F$, find a mapping $S(f(x))$ into algorithm space A, such that the selected algorithm $\alpha \in A$ maximizes the performance mapping $y(\alpha(x)) \in Y$.

It also states that features must be chosen so that the varying complexities of the problem instances are exposed, any known structural properties of the problems are captured, and any known advantages and limitations of the different algorithms are related to features. In meta-learning, the selection mapping S, responsible for selecting the right algorithm A given features $f(x)$ is, of course, produced by a learning algorithm.

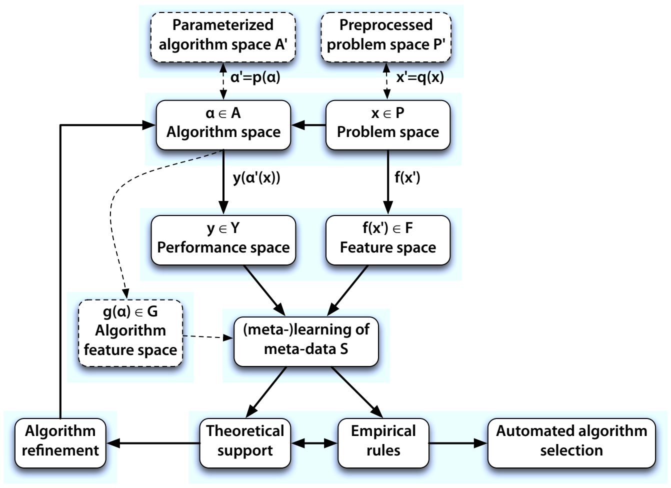

Figure 1.5: Proposed framework for meta-learning in Smith-Miles (2008a) (full lines), and our extensions (dashed lines) of this framework.

1.3.2 Smith-Miles's framework

Smith-Miles (2008a) provides a very insightful survey unifying several past meta-learning studies in Rice's framework, and also proposes an updated framework for meta-learning applications, shown in Figure 1.5 (full lines). Like in Rice's framework, problems are characterized with features and algorithms are evaluated on those problems. Instead of predicting a single algorithm, though, this meta-data will be used in two different ways. First, we can learn from this meta-data to gain insights into the behavior of learning algorithms, which can in turn be used to improve existing learning algorithms (shown on the bottom left of Figure 1.5). Second, we can build models of learning behavior, and use these to automatically recommend interesting learning algorithms for a particular problem (shown on the bottom right).

1.3.3 An updated framework

However, this framework still does not capture some important aspects of meta-learning. That is why we propose some extensions, shown in dashed lines in Figure 1.5. First of all, nearly all algorithms have a range of parameter settings which have a profound impact on the algorithm's bias, and thus on their performance on a specific problem P. We want to be able to recommend or study the effect of these parameter settings as well, and therefore introduce the space A', consisting of algorithms α' with a specific set of parameter settings. While one could argue that algorithms with specific learning parameters are simply different learning algorithms (Soares and Brazdil 2006; Ali and Smith-Miles

2006), we wish to maintain a distinction between how well an algorithm can perform in general, and what the effects of each of its parameters are. In some meta-learning settings, the focus could also be entirely on A'. Also, we want to be able to predict useful preprocessing steps for a given algorithm, which is why we introduce the space of all preprocessed problems P', in which x' is a dataset that has been preprocessed in a certain way. Finally, we also want to be able to generalize over learning algorithms to find patterns involving *properties* of algorithms, so we introduce the space G of measurable algorithm features.

We will now use this framework to highlight the most important meta-learning issues and their proposed solutions in the remainder of this chapter.

1.4 The data meta-feature space F

As lllustrated in Rice's framework and in Figure 1.3, the first step in meta-learning is to extract a number of properties in the data that are good predictors of the relative performance of algorithms. They should have *discriminative power*, meaning that they should be able to distinguish between base-learners in terms of their performance, and have a *low computational complexity*, preferably lower than $O(n \log n)$ (Pfahringer et al. 2000) otherwise predicting useful base-learners would not be much faster than actually running all of them.

1.4.1 Simple, statistical and information-theoretic properties

A range of commonly used data properties measure statistical or information-theoretical aspects of their distribution. They include the number of attributes, the skewness of the class attribute, the correlation between two numerical attributes, the entropy of attribute value distributions and many more. A complete list together with explanations of how they relate to the properties of certain learning algorithms is provided in Appendix A.

1.4.2 Concept-based properties

Another approach is to try and *characterize the hidden concept* in the dataset. Vilalta (1999) and Vilalta and Drissi (2002a) proposed two very interesting measures to do this for classification and prove these measures have great impact on algorithm performance:

- *concept variation*, the (non-) uniformity of the class-label distribution throughout the feature space (measured through the distance between two examples of a different class)
- *example cohesiveness*, the density of the example distribution in the training set

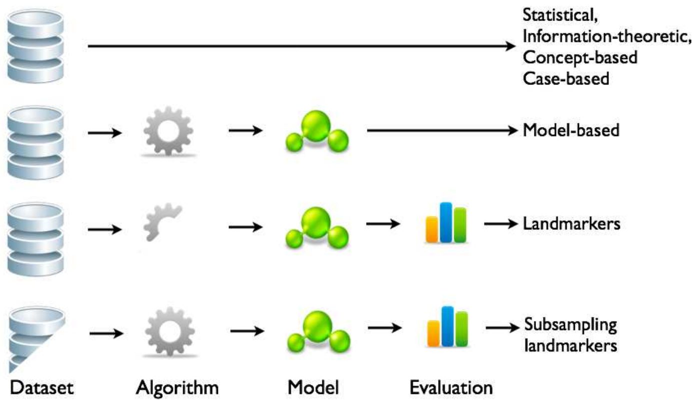

Figure 1.6: Different data characterization approaches. Adapted from Brazdil et al. (2009).

1.4.3 Case base properties

Another approach is to compare entire *observations* with each other (Köpf and Iglezakis 2002). A dataset may contain two observations with similar or equal attribute values, but with different labels which might 'confuse' a classifier. Analogously, there might be two or more observations which are identical, which gives them more weight in some algorithms. Here, properties derived from *case-based learning*, originally intended to assess the *quality* of a given case-base (Iglezakis and Reinartz 2002; Reinartz et al. 2001), are used. They perform well in combination with other data characteristics (Köpf and Iglezakis 2002):

- *consistency*: A single example is consistent if there does not exist any other example that is identical, but has a different target value.
- *uniqueness*: An example is unique if there exists no identical example.
- *minimality*: An example is *subsumed* by another example if its attributes form a true subset of another example with the same label. An example that is not subsumed by another example is minimal.
- *incoherence*: Incoherence is a measure for how dissimilar the examples are in their attribute space. An example is called incoherents if it does not overlap with any other example in a predefined number δ of attributes.
- *similarity*: The overall similarity of examples in a dataset is defined as the normalized weighted sum of four different local similarity measures (Köpf and Iglezakis 2002).

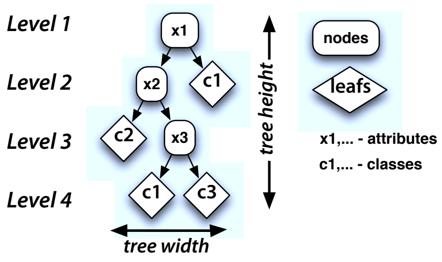

Figure 1.7: Structure of a decision tree. Adapted from Peng et al. (2002).

1.4.4 Model-based properties

Until now, all data characteristics have been computed directly on the dataset itself by measuring properties of the data distribution. As shown in Figure 1.6, this is only one possible approach. The following three are based on running actual learning algorithms on the data to get a first idea about the structures hidden in the data. They are, in a sense, *reconnaissance tactics* aimed at probing the hidden concept without doing complete evaluations of the induced models. The first, *model-based characterization*, builds a model that is typically very fast to induce, and characterizes the data based on properties of that model without doing a full-fledged evaluation of a wide range of learning algorithms. The second, *landmarking*, runs very simplified, and very fast, learning algorithms to get an idea of how well that learning approach might perform in general on that dataset. Finally, *subsampling landmarking* evaluates (complete) learning algorithms, but only on a small portion of the data, which, for most algorithms, will run much faster than running them on the entire dataset.

One model that is quite fast to train, and which has shown to provide useful meta-features for algorithm selection is the decision tree (Bensusan 1998; Peng et al. 2002; Peng et al. 2002):

- *tree width, treedepth, number of nodes, number of leaves* are illustrated in Figure 1.7. A complex tree generally means that the hidden concept is also quite complex, but sometimes even simple concepts are modelled with complex decision trees (e.g. if the decision boundary is a line that is not parallel to the axes). They actually tell us how easily the concept can be represented by a decision tree.
- $max_{level}, \mu_{level}, \sigma_{level}$ are the maximum, mean and standard deviation of the number of nodes in each level, reflecting the 'shape' of the tree.
- $max_{branch}, min_{branch}, \mu_{branch}, \sigma_{branch}$ are the maximum, minimum, mean and standard deviation of all branch lengths, reflecting whether some instances can be classified with fewer attributes than others.
- $max_{att}, min_{att}, \mu_{att}, \sigma_{att}$ are the maximum, minimum, mean and stan-

dard deviation of the number of times each attribute occurs in the nodes of the tree. It reflects whether some attributes are more important than other attributes, and how many different attributes are needed to generate a reasonable model of the data.

1.4.5 Landmarking

1.4.5.1 Landmarking algorithms

A radically different way of probing the structure of the concepts hidden in the data is running some algorithms on it with very different biases, and see how well they perform: the better they do, the closer their bias fits the data. This is what is done naturally when we *manually* seek an appropriate algorithm: we first select a wide range of very different algorithms, do some preliminary evaluations, and then we remove all algorithms that don't seem to perform well, leaving us with a small set of candidates to evaluate in detail. The caveat is that evaluating complete algorithms just to characterize datasets is quite wasteful, and the goal of meta-learning has always been to find some meta-features which can be computed quickly.

The first approach to solve this is to find some very 'inexpensive' versions of the algorithms, hoping they tell us something about the performance of their more expensive siblings. *Landmarking* (Pfahringer et al. 2000; Bensusan and Giraud-Carrier 2000b; Bensusan and Giraud-Carrier 2000a) mimics this approach by proposing some very simplified, bare-bones algorithms which only do a few refinement steps[3]. Indeed, some learning algorithms work by first building a very crude model of the data and then increasingly refining it. As the most important decisions are made in those first few steps, these models already hint at how well the algorithm's bias matches the data. As such, building several of such crude models with very different learning algorithms and evaluating their performance can give us a good indication of the nature of the structures hidden in the data, and allows us to situate datasets in the space of all learning problems. Indeed, one can imagine this space consisting of *areas of expertise* (Bensusan and Giraud-Carrier 2000b), in which a particular learning algorithm performs very well, and the goal of landmarking is locating new problems in this space: "tell me what can learn you and I will tell you what you are".

The following landmarking algorithms have been proposed:

- *Decision stumps*: Using a decision tree learner, C5.0 to be precise, a *single decision node* is constructed (representing a single split of the data) which is then to be used for classifying test observations. The goal of this landmark learner is to establish closeness to linear separability.

[3]The idea was first proposed in the StatLog project (Michie et al. 1994) under the term *yard stick methods*, but was initially not followed up on.

- *Random tree*: Also using decision trees, an *attribute is chosen randomly* at each node until the entire tree is built. The goal of this landmark learner is to inform about irrelevant attributes.
- *Worst node*: By using the decision tree's information gain ratio again, the *least informative attribute* is used to make the single split. Together with the first landmark learner, this landmarker is supposed to inform about linear, axis-parallel separability (Pfahringer et al. 2000).
- *1-Nearest Neighbor*: Define a distance on the instance space, e.g. the euclidean distance, and label a new observation with the observation of the closest training example. In classification problems, the goal of this landmark learner is to determine how close instances belonging to the same class are.
- *Elite 1-Nearest Neighbor*: A 1-Nearest Neighbor is used again, but only on a subset of attributes, i.e. the most informative attributes according to the information gain ratio. It intends to establish whether a task involves parity-like relationships between its attributes (Clark and Thornton 1997), which means that no single attribute is considerably more informative than another.
- *Naïve Bayes*: A simple learning algorithm using Bayes' theorem to calculate the possibility that an observation belongs to a certain class. Since it assumes that the attributes are conditionally independent from each other, this landmarker is used to measure the extent to which the attributes actually are independent given the class.
- *Linear Discriminant*: A single linear target function is computed splitting the instance space in two. Like decision stumps, it also establish closeness to linear separability, but not axis-parallelism.

1-Nearest Neighbor, Naïve Bayes and Linear Discriminant are actual learning algorithms with their parameters set to the simplest possible setting, which makes them fast enough to be used for landmarking.

Some variants exist. Bensusan and Kalousis (2001) showed that while landmarking is of great use for predicting the performance of algorithms, it did not fare so well on *ranking* learning approaches (see Section 1.9.1). *Relative landmarkers* (Soares et al. 2001) remedy this by using performance measures that are predictors of relative, rather than individual performance. While the former studies have focussed on classification, landmarkers for regression have also been proposed (Ler et al. 2005).

1.4.5.2 Sub-sampling landmarkers

A second solution are *subsampling landmarkers* (Fürnkranz and Petrak 2001): they run the learning algorithms on a small sample of the data, which for most algorithms will result in much faster training times. Indeed, learning algorithms typically learn the most from the first few examples. More examples will further fine-tune the model, but the performance gains will be small in comparison with the first few examples. When plotting the performance of learning algorithms on increasingly larger samples of a dataset, the resulting curve is called a *learning curve*. It will typically shoot up after a small percentage of the data and will, depending on the learning algorithm, start to level off shortly after that. The assumption of subsampling landmarking is that the performance of one point in the beginning of the learning curve will help us to predict the performance on the entire dataset. Unfortunately, there are great variations in the learning curves of several algorithms: some will level off completely, while others tend to keep on improving slowly given more data. Probably because of this, subsampling landmarking is not as efficient as was hoped (Fürnkranz and Petrak 2001; Fürnkranz et al. 2002)[4]. A relative version, predicting rankings instead of performances, was also proposed: these so-called *sampling-based relative landmarks* (Soares et al. 2001) seem to be better for ranking, but overall the results are rather mixed.

However, instead of taking one subsample, we could invest a bit more and take small subsamples of different sizes, effectively measuring the first part, and thus the *shape* of the learning curve, which hopefully gives us a better prediction of the performance on the entire dataset than just a single point. *Meta-learning on data samples (MDS)* (Leite and Brazdil 2005) works by first determining the complete learning curves of a number of learning algorithms on several different datasets. Next, for a new dataset, progressive subsampling was done up to a certain point, experimentally determined to be $m = 7$ steps, each of which containing $2^{6+0.5m}$ data points. Further work defined an algorithm to automatically decide when to stop sampling (Leite and Brazdil 2007). Finally, the resulting partial learning curves were matched to the nearest complete learning curve for each algorithm in order to predict their final performances on the entire new dataset. To tackle situations where the *shape* of the learning curve may be correct, but the *starting point* is not, a case-based reasoning process called *adaption* is used which moves the curve up or down to fit the partial curve better. The meta-knowledge thus simply consists of all learning curves on different problems, which are quite expensive to compute at training-time, but at prediction-time, drawing the partial learning curves is fairly efficient, and they seem to be able to predict learning performance quite accurately, depending on how many progressive samples are taken. The same approach has also been used to predict the *stopping point* in progressive sampling: the point where the learning curve does not increase sufficiently to warrant further training, leading to time savings when many algorithms need to be evaluated (Leite and Brazdil 2004).

In closing, it may be clear that landmarking offers an interesting trade-off: if available, more time could be invested in getting more accurate meta-features and thus more accurate predictions. In the limit, it approximates running the

[4]Fürnkranz et al. (2002) also introduced 5 variations that combine the results of subsampling landmarkers in different ways, e.g. pairing 2 algorithms and storing which one performed best on the subsample, but this does not seem to improve performance

complete algorithms on the complete datasets, but of course the goal is to stop long before that.

1.4.6 Task-specific meta-features

In some tasks, the data cannot be represented as a single table, and many meta-features may not be useful anymore. One example is *time series analysis*, in which the data consists of a *sequence* of values, or a *signal* over time. In this case, meta-features such as the trend, or slope, of the time series regression model can be used (Arinze 1994; Venkatachalam and Sohl 1999). Also, while the normal correlation ρ_{XY} cannot be used, the sample *autocorrelation* coefficients (given by the correlation between points which are a lag time τ apart in the series) can be used instead (Arinze 1994; Prudêncio and Ludermir 2004; dos Santos et al. 2004):

- *MEAN-COR*, the mean of the absolute values of the 5 first autocorrelations: high values of this feature suggests that the value of the series at a time point is very dependent of the values in recent past points.
- *TAC-X*, the statistical significance of autocorrelations: indicates the presence of at least one significant autocorrelation in the first X autocorrelations.
- *COEF-VAR*, coefficient of variation: measures the degree of instability in the series.

Note that, since some of these meta-features are time-dependent, they will be updated several times during the meta-learning process to include the most recent time series information.

Also for unsupervised learning, or *clustering*, in which no labels are available, some variants of statistical and information-theoretic meta-features have been proposed using, among others, the $VarCoef$ of each attribute separated in three values corresponding to the first three quartiles of their value distributions (Soares et al. 2009). Finally, Wang et al. (2006, 2009) define meta-features for clustering time series, although they may have to be revisited since it has been shown that the clusters extracted by most current algorithms are essentially random (Keogh and Lin 2005).

Another meta-learning setting is to not select the best algorithm, but rather to select the best parameter settings for a predefined algorithm. In this case, *algorithm-specific meta-features* may be defined aimed at discriminating the performance of the learning algorithm under different parameter settings. For instance, to predict the right parameters for a specific kernel of a Support Vector Machine, it has been shown that kernel matrix-based meta-features lead to better results than general ones (Soares and Brazdil 2006). In a somewhat different approach, properties of the kernel matrices where combined with other meta-features describing the data in terms of their relation to the margin (Tsuda et al. 2001).

Finally, meta-learning has also been promoted as a way to solve the algorithm selection problem in other sciences, a very nice overview of which is provided by Smith-Miles (2008a). Smith-Miles (2008b) proposed a set of meta-features specifically aimed at selecting algorithms for the Quadratic Assignment Problem in the field of optimization.

1.4.7 Selecting meta-features

While several studies have compared different sets of meta-features, also combining several meta-features of different types (Bensusan and Kalousis 2001; Pfahringer et al. 2000; Köpf et al. 2000; Todorovski et al. 2002; Lee and Giraud-Carrier 2008), there are no conclusive results indicating that one set of meta-features is definitely better or worse than others. It depends on the nature of the meta-learning problem under study each time. Interestingly though, a comparison between landmarkers and statistical and information-theoretic meta-features showed that using landmarkers alone yields the best results, and that adding other meta-features often tends to impair their performance (Bensusan and Kalousis 2001; Pfahringer et al. 2000). The exception to this rule seems to occur when the chosen meta-learner is a linear discriminant or naïve Bayes algorithm, in which case it is best to combine the two types of meta-features.

Case-based meta-features only seem to perform adequately in combination with other meta-features (Köpf and Iglezakis 2002). More studies are needed to provide further insight.

In any case, making a good selection of meta-features is paramount. It may be clear by now that the meta-feature space is very high-dimensional, while each instance in this space requires a completely new dataset to characterize. Since the collection of publicly available datasets is limited, this amounts to very few samples in a very large space, providing very sparse evidence for any recommendation we deduce from it. It may help to reduce dimensionality by selecting the most relevant meta-features, or otherwise, to find a way to obtain more datasets (see Section 1.6).

One way of selecting meta-features is to only choose those measures assumed to be relevant for the problem at hand. For instance, if we have a constrained set of base-learners, we could select meta-features useful for discriminating between them (Aha 1992; Kalousis and Hilario 2001b; Kalousis and Hilario 2001a). For instance, a set of seven meta-features were selected especially to discriminate between very diverse algorithms: decision trees, neural networks, nearest neighbors and naïve Bayes (Brazdil et al. 2003).

Lee and Giraud-Carrier (2008) used visual analysis and computational complexity considerations to find 4 meta-features whose values are directly relevant to certain ranges of predictive accuracy for 7 learning algorithms. They also defined discretization thresholds for these 4 features and showed they can be used to boost the accuracy of the meta-level classification task.

A more fundamental approach is to start from the complete set of meta-features, and use automatic feature selection techniques to eliminate those features which seem to contribute the least, depending on the specific meta-learning problem. In Todorovski et al. (2000) a wrapper feature selection approach is used to select the most useful meta-features, showing that it improves meta-learning performance. A slightly different approach is used in Kalousis and Hilario (2001b). First, the meta-learning problem is transformed into several pairwise meta-learning problems, each aimed at predicting which of two algorithms is superior, after which feature selection techniques are used to select different sets of meta-features for each single one of them.

1.5 The algorithm meta-feature space G

Next to describing properties of the datasets, it might be useful to describe properties of algorithms as well. Indeed, it would be very useful to induce meta-rules that tell us when a *kind* of algorithm is useful, rather than a specific implementation: otherwise we are unable to tell whether a somewhat modified version of an algorithm would work equally well. Also, these properties may tell us *why* an algorithm performs well or badly, or whether a certain modification may correct the problem or not (Van Someren 2001), which is useful in algorithm design.

To address these issues, we need to expand our range of meta-features with properties of the algorithms under consideration.

1.5.1 Qualitative properties

A first set of algorithm characteristics have been identified by Michie et al. (1994) and Hilario and Kalousis (2001):

- *Type of data it can handle*, e.g. numerical, nominal or relational
- *Whether it can handle costs.* This feature indicates whether the algorithm can take a *cost function* into account which adds weight to certain types of errors. Sometimes, e.g. when fighting credit card fraud or in medical predictions, a false negative can be much worse than a false positive.
- *Strategy*, a categorization of learning algorithms by the learning strategy they follow, or by the family of learning algorithms they belong to. One example is to look at the type of model they build.
- *Variable handling*, i.e. whether the algorithm operates sequentially, examining one attribute at a time (e.g. decision tree learners), in parallel, examining all attributes simultaneously (e.g. neural networks), or a hybrid of the two.
- *Cumulativity.* Algorithms which do not allow attributes to interact, such as naïve Bayes have high cumulativity, while decision trees have low cu-

mulativity, allowing maximal interaction between variables (Blockeel and Dehaspe 2000).

- *Incrementality.* An algorithm is incremental if it can build and refine a model gradually as new training instances come in, without reexamining all instances seen in the past. However, they are generally very sensitive to the order in which examples are presented.

A second set of properties is more subjective and expresses how *practical* the algorithm is to use. Each is assigned a 5-level score (Hilario and Kalousis 2001).

- *Comprehensibility* of the method: is it easy for users to understand how the algorithm works and what its parameters do?
- *Interpretability* of the learned model: whether the model can be interpreted to extract knowledge. For instance, a decision tree is easy to interpret by humans, a neural network is not.
- *Parameter handling*, the degree to which model and search parameters are handled automatically, or how much work goes into fine-tuning all parameter settings. Naïve Bayes is very simple to tune, Support Vector Machines are much harder to optimize.

1.5.2 Quantitative properties

The main problem with these qualitative properties is that they depend on experts to correctly classify each algorithm. One expert's opinion may differ from the next, and some algorithms may be hard or impossible to classify. Another approach is to test properties of algorithms through controlled experiments. As such, we can build *quantitative profiles* of algorithms, a number of which are proposed in Hilario and Kalousis (2000a). We mention the proposed algorithm properties here with related work.

1.5.2.1 Bias-variance profile

Bias-variance analysis (Kohavi and Wolpert 1996) is a statistical technique that allows us to characterize the errors made by learning algorithms by splitting up the total expected misclassification error in a sum of three components, each covering a portion of the data points that were classified incorrectly.

- The (squared) *bias error* is the systematic error the learning algorithm is expected to make because its bias doesn't match the data: the algorithm cannot approximate the concept close enough. Given different samples of the data, the algorithm will always classify these data points wrongly.
- The *variance error* is a measure for how strongly generated hypotheses vary on different samples of the same dataset. Given different samples of the data, the algorithm will sometimes classify these data points correctly and sometimes incorrectly.

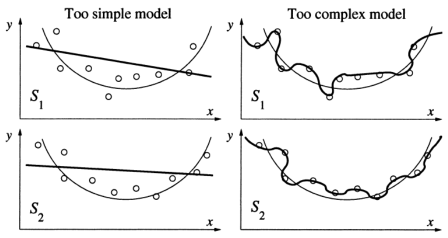

Figure 1.8: Bias versus variance error (Geurts et al. 2005).

- The (squared) *intrinsic error* is associated with the inherent uncertainty in the data.

Dietterich and Kong (1995) investigate the relationship between learning bias and the statistical bias- and variance error. If an algorithm's bias is 'appropriate', meaning that it can approximate the hidden concept close enough, there exists a trade-off between the bias error and variance error: using complex (or 'flexible') models results in low bias error, but high variance error due to overfitting: the algorithm is more likely to model noise. On the other hand, using simple, highly constrained models leads to low variance error, but high bias error since the model is too rigid to approximate the target concept more closely, also known as underfitting.

This is illustrated in Figure 1.8. Each row uses a certain data sample and each column a certain algorithm. The target concept is the low degree polynomial. The algorithm on the left builds a linear model which is too rigid to approximate the target concept, and which only varies slightly on different samples. It has high bias error: a misclassified point will likely be misclassified no matter which training sample is used. The algorithm on the right builds models that are too complex, and which will vary greatly as the training sample changes. This means it has high variance: many points will be correctly classified on some training samples, but incorrectly on others. Vice versa, the left algorithm exhibits low variance error, while the right one exhibits low bias error.

Bias and variance can be measured only with regard to the expectation of a learned models actual output, averaged over the ensemble of possible training sets D for a fixed sample size N. We thus need to evaluate algorithms on many different equally-sized samples of the same dataset, and repeat this process over many datasets in order to assess the average percentage of bias error versus variance error typically observed in the algorithm. If an algorithm has parameters that influence its bias, then we must take these into account as well.

1.5.2.2 Runtime properties

Training and execution time and (memory) space can be measured either by recording the relative cpu and memory usage, or by expressing the algorithm's time and space complexity as a function of dataset dimensions.

1.5.2.3 Resilience properties

These properties measure how well the algorithm handles certain types of data that are typically harder to learn from.

Scalability A learning algorithm is *scalable* if its performance doesn't suffer too much when operating on larger datasets.

Resilience to missing values Increasingly remove values from a dataset and track how badly each algorithm is affected by this (Hilario and Kalousis 2000a).

Resilience to noise Increasingly add noise to datasets and study how algorithms are affected (Aha et al. 1991).

Resilience to irrelevant attributes Irrelevant attributes are attributes that are not correlated with the target attribute. Increasingly add irrelevant attributes and study the effect Hilario and Kalousis (2000b). For instance, naïve Bayes is utterly unaffected by irrelevant attributes.

Resilience to redundant attributes Redundant attributes are attributes that are interdependent, i.e. they offer no information that hasn't been given by other attributes. Domingos and Pazzani (1997) shows that naïve Bayes is surprisingly resistant to redundant attributes, while Wettschereck et al. (1997) shows that nearest neighbors algorithms are affected strongly.

1.6 The problem spaces P and P'

Having now described an extensive array of useful meta-features, the next thing we need is a large collection of datasets and algorithms to characterize and to gather the necessary meta-data to learn from.

There are several large repositories of datasets we can use, such as the UCI Machine Learning Repository (Asuncion and Newman 2007). Unfortunately, while these contain plenty of datasets for benchmarking algorithms (currently around 150), each dataset provides one meta-example of the performance of a learning algorithm, giving us not much to learn from. Of course, there are many, many more datasets out there, but few of them are public domain. In some settings, such as massive data streams which continuously generate new examples, or truly giant datasets which can be segmented into a large number of subsets, these problems are not so acute.

1.6.1 Expanding P

There are several ways to get more datasets. The first is to generate new datasets by manipulating existing ones. This may be done by changing the distribution of the data, e.g. by adding noise or changing the class distribution, or by changing the structure of the problem, e.g. by adding irrelevant features (Aha 1992; Hilario and Kalousis 2000b). Soares (2009) proposed to generate new datasets by simply switching various attributes with the target attributes yielding new datasets referred to as datasetoids, and showed that the extra meta-training points improve performance.

A second approach is to generate new datasets from scratch (Vanschoren and Blockeel 2006). This works by first defining a concept, and then randomly generating data points, labeling them with the predefined concept. This has the great benefit that the hidden concept is known, so we can find out exactly which properties of these concepts are beneficial or detrimental to a learning algorithm's performance. As such, these artificial datasets are a great asset to really understand learning behavior.

However, when trying to generate meta-data for algorithm selection, some caution is needed. First, we must avoid, as much as possible, to introduce unnecessary bias in the set of artificial datasets by including only certain kinds of concepts. And second, we must try to generate datasets that approximate natural ones. This means building a dataset generator that introduces typical characteristics of natural data such as noise, irrelevant attributes, missing values, complex relationships between attributes, different correlations, and very different distributions of attribute values. Unfortunately, there is no way of telling how 'natural' an artificial dataset is, and building a dataset generator able to include all these aspects remains a challenge (Scott and Wilkins 1999; Dries 2006).

1.6.2 The preprocessed problem space P'

The knowledge discovery process is much more involved than simply preparing a dataset and selecting an appropriate learning algorithm. Often, the applicability of a learning algorithm depends heavily of the data preprocessing and transformation processes that preceded it. A simple data transformation, e.g. discretizing the numerical attributes, can enable the use of new learning algorithms, and a preprocessing step such as feature selection aimed at removing irrelevant features can have a profound impact on the relative performance of learning algorithms which are less resilient against such complications than other algorithms, as discussed in Section 1.5.2.

We should therefore not simply predict the correct algorithm: we should offer complete *workflows* of processes.

Note that all preprocessed datasets making up the preprocessor space P' are also good candidates to extend P. Datasets in various stages of preprocessing and transformation can be characterized and have algorithms run on them to get meta-data about how various learning algorithms are affected by such data alterations. Not only does this provide a more complete picture of the performance of learning algorithms under various conditions, it is also useful meta-data to generate workflows for future problems. For instance, algorithms that are more resilient against missing values are less likely to require a preprocessing step aimed at reducing the number of missing values.

The proposed bias-variance profile of learning algorithms can be of great help here. If a learning algorithm exhibits high bias error, *feature construction* techniques may be of use. These replace the attributes by new attributes typically constructed by combining several other ones, radically changing the structure of the instance space. *Feature selection*, on the other hand, is generally useful for variance reduction: fewer irrelevant attributes reduce the chance of overfitting, whereas little relevant information is lost. Analysis on real-world data mining problems has revealed that especially variance is a very important source of error before preprocessing (Van Der Putten and Van Someren 2004).

1.7 The algorithm spaces A and A'

When users needs to find an algorithm implementation, there is quite a large variety of data mining suites they can choose from, such as WEKA[5], KNIME[6] and R[7], to name a few. Including all these algorithms in a meta-learning scenario would require running all of them on a large number of datasets. Moreover, many of these algorithms have a number of parameters which have a profound effect on their performance, and many of these parameters are continuous, so the number of alternative values is infinite. The computational cost of generating meta-data for all these alternatives is sufficient to warrant a careful selection of learning algorithms and parameter settings under consideration.

Since we want a limited selection of base-learners that still covers the algorithm space A, we need to select algorithms that are sufficiently different from each other. Likewise, we want to cover the space of parameterized algorithms A', so we also need to select a number of parameter values that cover the range of possible parameter values in a sufficiently dense manner.

There are few objective guidelines about how to do this. Usually, an informed decision must be made using knowledge of how the various algorithms and implementations differ from each other, what are the most sensible values for each of the parameters, and how all this relates to the datasets under consideration.

However, there are some experimental approaches to determine the adequacy of a given set of base-algorithms. Soares et al. (2004) proposes the following

[5]http://www.cs.waikato.ac.nz/ml/weka/
[6]http://www.knime.org/
[7]http://www.r-project.org/

four conditions to evaluate a given set on m algorithms $A = a_1, ..., a_m$, or alternatively, m parameter settings for a given learning algorithm:

- *Overall relevance*: For *most* datasets, there must be a a_i that performs better than a suitable baseline (e.g. a very simple learning algorithm).
- *Overall competitiveness*: The results of all algorithms in A cannot be *significantly* improved by adding an additional algorithm to A.
- *Individual competitiveness*: For every a_i there should be at least one dataset on which a_i is superior to any other algorithm in A
- *Individual relevance*: For every a_i and every other a_j, there should be at least one dataset on which a_i is *significantly* better than a_j.

Except for the second condition, it is possible to check these conditions experimentally by generating the meta-data and checking afterwards whether these conditions hold. Though it is practically difficult to guarantee the second condition, increasingly large alternative sets can be tried to see if the overall competitiveness can be improved.

Another approach is to determine empirically how similar two algorithms are in terms of their performance on a collection of datasets. Kalousis et al. (2004) achieves this by considering all possible pairings of algorithms, running them on a set of datasets, and calculating the *error correlation*: the correlation between the errors made by the two algorithms. This value is one if the two algorithms make exactly the same errors, and zero if they never make the same error. For each algorithm pair, the error correlations over all datasets are collected and plotted in a histogram. Finally, the *affinity coefficient* (Bock and Diday 2000) is used to calculate the distance between the histograms, and this information is in turn used to construct a hierarchical clustering, grouping algorithms together that perform similarly over all datasets. Knowing which algorithms perform similarly and which datasets are similar provides useful insights into whether a given learning algorithm should be considered or not.

1.8 The performance space Y

With a well-defined set of algorithms and datasets, we now need to evaluate the algorithms on the datasets to see how well they perform. This will, together with the dataset and algorithm characterizations, constitute the meta-data from which we want to learn about the the suitability of learning algorithms on different kinds of data.

1.8.1 Evaluation metrics

The performance of the base-learning algorithms can be quantified in many different ways. The vast majority of studies in meta-learning only use the *predictive accuracy* in classification, i.e., the percentage of examples classified correctly, and the *normalized mean squared error* in regression, i.e., the normalized squared sum of all errors. However, there are many more useful metrics that can be considered instead, such as the *area under the Receiver Operating Characteristic-curve*, *precision* and *recall* to name a few, each of which are specifically useful in particular data mining settings. Definitions of these can be found in any textbook on machine learning or data mining.

1.8.2 Combining metrics

Furthermore, the end users of a meta-learning system may be interested in an *aggregate* of evaluation metrics: they may want a learning algorithm that is both fast *and* accurate. Since different users have different needs, it is no use to define these aggregates beforehand. Rather, we evaluate the base-learners with a range of different metrics, and we aggregate the performance evaluations afterwards, during the meta-learning phase, according to the wishes of the user.

Brazdil et al. (2003) propose an aggregate metric, the *adjusted ratio of ratios*, which allows the user to add emphasis either on the predictive accuracy or the training time. An alternative is to use Data Envelopment Analysis (DEA) (Charnes et al. 1978; Nakhaeizadeh and Schnabl 1997) which automatically adjusts the weights of the different evaluation metrics. Variants of DEA also exist that allow the user to define the relative importance of the different metrics (Nakhaeizadeh and Schnabl 1998).

1.9 The meta-learner S

Having collected all the necessary meta-data, the final step in the meta-learning process, illustrated in Figure 1.3, is to choose a meta-learner, train it on the meta-data and predict which algorithms are best suited for a given problem.

1.9.1 Meta-learning targets

There are several types of outputs that can be generated by meta-learning algorithms (Brazdil et al. 2009), their utility depends on the specific meta-learning setting:

Recommendations We can simply return the estimated best algorithm out of the entire set of base-learners (Pfahringer et al. 2000; Kalousis 2002), or in case we want to predict parameter settings, the estimated optimal value for that parameter (Kuba et al. 2002; Soares et al. 2004). However, to give the user some more options, we can also return all algorithms that perform sufficiently well: within a predefined margin around the performance of the best algorithm (Todorovski and Dzeroski 1999), or not significantly worse than the best one (Jorge and Brazdil 1996; Kalousis and Theoharis 1999).

Rankings Alternatively, we could provide a full ranking of all the base-learners (Brazdil et al. 1994; Brazdil et al. 2003; Soares and Brazdil 2000), so that users can select the highest-ranking algorithm they consider useful, or if more time is available, to do further evaluations with the top-N algorithms. Some variants of this approach are *weak rankings*, in which algorithms that perform similarly (not significantly different) get the same rank (Brazdil et al. 2001), and *incomplete ranking*, in which some base-learners are omitted, e.g. because not enough meta-data was available to warrant a reliable prediction.

Performance estimates In this case, regression models are built predicting the performance of each of the learners on the new dataset (Bensusan and Kalousis 2001; Gama and Brazdil 1995; Köpf et al. 2000; Sohn 1999). Since absolute values can be misleading, Gama and Brazdil (1995) proposes three ways of rescaling them: by the distance to the performance of the best algorithm, the distance to some performance baseline, or normalizing them by calculating the difference with the mean performance on that dataset divided by the standard deviation. Tsuda et al. (2001) proposes an alternative approach in which the meta-learner predicts, for each of the examples in the new dataset, whether a specific base-learner can correctly predict its class.

Descriptive meta-learning Next to *predictive meta-learning*, in which the aim is to predict the utility of algorithms, we can also pursue *descriptive meta-learning*, aimed at using the collected meta-data to build models that teach us something about the learning algorithms involved: to find *patterns* in the learning behavior of the base-learners. Not only can these patterns can be exploited to make informed decisions about which algorithms to select, they also provide insights useful for *modifying* algorithms to better fit the problem at hand, or for developing better or more generally applicable algorithms.

1.9.2 Algorithms for meta-learning

Any base-level algorithm could be considered to be used at the meta-level as well. Selecting the most useful one constitutes a *meta-meta-learning problem* (Rendell et al. 1987), depending on the precise set of meta-features and performance-based meta-data used. Also, more practical considerations may come into play when selecting a meta-learner, such as the desired type of output or incrementality, the ability to easily extend the model when new meta-examples become available. However, there do exist some experimental comparisons of several meta-level learners. We discuss them based on the type of output under consideration.

1.9.2.1 Recommendations

In the setting of recommending a single algorithm, Bensusan and Giraud-Carrier (2000b) and Pfahringer et al. (2000) examine which of ten different meta-learners are most useful. In both cases, the meta-learners were trained on a large collection of artificial datasets and tested on a smaller sample of real-world datasets. Results indicate that there is no single best meta-learner, but that decision tree-like algorithms (C5.0trees, C5.0rules and C5.0boost) seem to have an edge, especially when used in combination with landmarkers. Further experiments performed purely on real-world data corroborate these results, although they also show that most meta-learners are very sensitive to the exact combination of statistical and information-based meta-features used (Köpf and Iglezakis 2002).

In the setting of recommending a subset of algorithms, Kalousis (2002) and Kalousis and Hilario (2001b) showed that on statistical and information-theoretical meta-features, boosted decision trees obtained better results than nearest neighbor, normal decision trees and decision tree-based rule learning methods. Lindner and Studer (1999) and Hilario and Kalousis (2001) employed *relational case-based reasoning*, able to use relational representations of meta-examples, including algorithm properties independent of the dataset (see Section 1.5.2) and histogram representations of dataset attribute properties (see Section A.7).

1.9.2.2 Ranking

The most commonly used meta-learning algorithm for ranking is based on k-nearest neighbors (kNN): the k datasets nearest to the new dataset are selected and the rankings of algorithms on those datasets are combined into the final ranking of algorithms for the new dataset (Soares and Brazdil 2000; Brazdil et al. 2003; dos Santos et al. 2004). The rankings can be combined by simply computing the average rank of each algorithm (Soares and Brazdil 2000), or by using *success rate ratio's* and *significant wins* (Brazdil and Soares 2000). (Brazdil et al. 2001) also uses significant wins to produce a more concise ranking in which redundant algorithms, those that never seem to perform significantly better than another algorithm, are removed. There seem to be no significant differences in the performances of these methods (Soares 2004).

Another way to produce rankings is to first estimate the performances of all algorithms, and then transform these results into a ranking (Bensusan and Kalousis 2001; Köpf et al. 2000), yielding better results than the nearest neighbor-based approach.

Finally, rankings can also be produced by using predictive clustering trees (Todorovski et al. 2002; Blockeel et al. 2000) (see below). These, in turn, yield more accurate rankings than the former two methods.

1.9.2.3 Performance estimation

Predicting algorithm performance generates as many regression problems as there are base-learners, each predicting the performance of a specific base-learner on the new dataset. Gama and Brazdil (1995) compares four different regression techniques: logistic regression, C4.5rules, M5.1 (a model tree learner[8]) and kNN, and finds that they exhibit similar performance. Bensusan and Kalousis (2001) compares a regression rule learner (Cubist) with a kernel method and finds that the latter produces slightly more accurate predictions.

In a different setting, Janssen and Fürnkranz (2007) use a regression algorithm to predict the accuracy of different rule learning heuristics. Both linear regression and neural networks have been tested with the latter achieving slightly better results, but only in its simplest setting (1 hidden node), which corresponds to a linear model.

Instead of predicting the performance of each algorithm separately, we can also use a clustering tree[9] (Blockeel et al. 2000) and predict the performance of all algorithms at once (Todorovski et al. 2002). The decision nodes represent tests on the values of meta-features and the leaf nodes contain sets of performance estimates: one for each base-learner. An extra benefit is that the resulting tree can be easily interpreted (see the next section). Results obtained with this method are comparable with those combining separate regression models.

1.9.2.4 Descriptive meta-learning

In order to learn about the behavior of learning algorithms, the meta-learner of choice clearly needs to be one whose model can be easily interpreted. One of the earliest studies aimed at discovering guidelines for the selection of algorithms was the StatLog project (Michie et al. 1994), in which rules were constructed that dictated whether an algorithm was deemed appropriate or inappropriate for use on a given dataset. More recently, Ali and Smith (2006) used a decision-tree based rule learner to discover update rules defining, for each algorithm, when it should be used. The same approach was used to discover rules about the selection of kernels in SVMs (Ali and Smith-Miles 2006). Aha (1992) and Kalousis and Theoharis (1999) constructed decision trees for pairs of algorithms indicating whether, on a given dataset, one of them was significantly better or not. While these are actually predictive approaches, the underlying models were interpretable. A very different approach was followed by Smith et al. (2002), in which Self-Organising Maps (SOMs) (Kohonen 1982) were employed, an unsupervised neural network approach to clustering able to draw

[8]Model trees (Quinlan 1992) are decision trees capable of regression by storing linear regression models in their leaves.

[9]A clustering tree, such as the CLUS algorithm, tries to minimize the variance of the examples in each leaf and maximize the variance across different leaves, an approach also used in clustering.

a two-dimensional map of the clusters. In this case, they were used to cluster similar datasets based on statistical and information-theoretic meta-features and identify how algorithms perform in those different clusters.

Some approaches have also tried to exploit relational descriptions of meta-features. For instance, Todorovski and Dzeroski (1999) has used FOIL (Quinlan and Cameron-Jones 1993), an ILP algorithm, to induce rules about the utility of learning algorithms.

Gaining insights into the behavior of learning algorithms is a central theme in this thesis, and we will come back to this issue soon.

1.10 Summary and conclusions

In this chapter, we have given a perspective overview of the field of meta-learning, focussing on the task of understanding and predicting learning behavior, and we have used an extension of Rice's framework for algorithm selection to highlight the key questions and proposed solutions in each component of the general meta-learning process. Similar to Smith-Miles (2008a), we can recast the literature in Rice's framework, or in this case, the extended framework used in this chapter. The result is shown in Table 1.1, containing a representational sample of the various meta-learning studies in machine learning. We've also ordered the studies by date of publication to get a sense of how the field has evolved over time.

The table shows the task handled in each study: classification, regression, time series analysis (forecasting), clustering or parameter selection. It also covers each of the meta-learning aspects discussed in this chapter: the number of natural and artificial datasets P and preprocessed datasets P' used, the number of algorithms A and their parameter settings A' covered, the meta-features F and G exploited, the evaluation metric Y applied and the meta-learners S considered for each of the targets: recommendation, ranking, performance prediction and declarative model generation. The word 'dyn' in the landmarking column means that the number of subsampling landmarkers is chosen dynamically.

Algorithms and datasets Reading the table from left to right, a first trend to observe is that while meta-learning used to focus on classification, more recently some inroads have been made into other tasks, showing that meta-learning can be of great use there as well. When it comes to the datasets used, it shows that most studies, with the exception of time series analysis, employ relatively few datasets. Some studies try to tackle the lack of publicly available datasets by generating artificial datasets or, more interestingly, by applying preprocessing techniques to generate alternative datasets based on natural data. Still, only few studies go that far and then only try a few preprocessing techniques, such as adding or removing irrelevant attributes or missing values. This approach

Table 1.1: Overview of the literature in our extended meta-learning framework.

Study reference	Task	P		P'	A	A'	F[a]					G	Y	S			
		Nat.	Artif				S&I	Co	CB	MB	LM			Recommend.	Rank.	Pred.	Descript.
Aha (1992)	classificat.	1	2430	8	3		7						acc				C4.5rules
Brazdil and Henery (1994)*	classificat.	22*			23*		16*						acc				C4.5rules
Gama and Brazdil (1995)	classificat.	22*			23*		16*						acc	4 regressors			
Kalousis and Theoharis (1999)	classificat.	77			3		45						acc	pairwise kNN			
Lindner and Studer (1999)	classificat.	80			21*		21					4	acc	CBR			
Todorovski and Dzeroski (1999)	classificat.	20			3		22						acc	6 classifiers			FOIL
Vilalta (1999)	classificat.	19			2			2					acc				Visual
Venkatachalam and Sohl (1999)	time series	180			9		6						acc	ANN			
Köpf et al. (2000)	regression		5450		3		16*						MSE,MAD	C5.0		M6	
Pfahringer et al. (2000)	classificat.	18	222		6		5				4		acc	11 classifiers			Ripper
Bensusan and Kalousis (2001)	regression	20			9		50				7		MAD			2 regr	
Hilario and Kalousis (2001)	classificat.	98	1250	2	9		16*					14	acc	CBR			
Kalousis and Hilario (2001b)	classificat.	77	1082	2	10		57						acc	4 classifiers			
Smith et al. (2001, 2002)	classificat.	57			6		21						acc	ANN			SOM
Todorovski et al. (2002)	classificat.	65			8		33				7		acc		CLUS		
Köpf and Iglezakis (2002)	classificat.	78			10		16*		5		7		acc	10 classifiers			
Peng et al. (2002)	classificat.	47			10		21			15	7		acc+time		kNN		
Brazdil et al. (2003)	classificat.	53			10		7						acc+time		kNN		
Soares et al. (2004)	param.sel.	42			1	11	14						NMSE		kNN		
Prudêncio and Ludermir (2004)	time series	645			3		5						acc	pairwise ANN			
Ler et al. (2005)	classificat.	80			6						10		acc	7 classifiers			
Ali and Smith-Miles (2006)	param.sel.	112			1	5	29						acc+time				C5.0rules
Leite and Brazdil (2007)	classificat.	40			5		7				dyn		acc	curve match			
Soares et al. (2009)	clustering		160		9		9						acc	SVM,ANN			
Wang et al. (2009)	time series	46	315		4		9						RAE,SPB				C4.5,SOM

[a]Statistical & Information-theoretic (S&I), Concept-based (Co), Case-based (CB), Model-based (MB) or Landmarking (LM)

should definitely be taken further: including the behavior of learning algorithms under these alternate conditions will provide much richer meta-data.

Furthermore, most studies pick only a few different algorithms, and their parameters settings are never varied, thus only generating meta-learning advice valid for their default performance, which is likely to be suboptimal. The only exception are studies where the meta-learning goal is to predict the value of a parameter, in which case each parameter setting is viewed as a different algorithm, and no other algorithms are involved. Moreover, this parameter is always a component of an SVM algorithm. It would be very useful to explore the parameter space of many more algorithms, especially those where performance changes dramatically with different settings, such as ensemble learners.

Most likely, the main reason why so little dataset and algorithm variants are tried is that collecting the meta-data is expensive: every single combination of these factors must be tried, cross-validated, and repeated several times to account for random effects, easily resulting in millions of experiments. Also keep in mind that most of the studies between 1994 and 2003 were heavily facilitated by two large projects, StatLog (Michie et al. 1994) and METAL (Giraud-Carrier 2005), in which many researchers pooled their resources and established new ways to generate and share meta-data. It is no coincidence that many studies share the same datasets, algorithms and meta-features: those indicated with an asterix (*) all stem from the StatLog project, and most studies between 2001 and 2003 use the same 9-10 base-learners included in the METAL project. If we want to take meta-learning to a level where many more datasets, preprocessors, algorithms and parameters are included we need to find new ways to efficiently share and exploit meta-data.

This is an issue we wish to resolve in this thesis: in Part II of this thesis, we define an XML-based language in which machine learning experiments can be freely exchanged and introduce experiment databases that automatically gather and organize them. As will be discussed in Chapter **??** they bring many benefits for various machine learning researchers, and are an ideal platform for meta-learning studies.

Meta-features Still, meta-learning is not only about empirical data. The more theoretical aspect of finding useful meta-features is equally important. Our summary table, and our discussion in Section 1.4 show that the range of data meta-features, especially statistical and information-theoretic ones, is investigated quite extensively. One might still wonder whether these meta-features get us any closer towards understanding *why* an algorithm performs well on a dataset or not. Many studies are based on the idea that if we simply use enough meta-features, there will be enough information for a meta-learner to discover the complex patterns that link combinations of these features to the performance of the algorithm. The meta-rules generated in this manner generally use complex combinations of terms like mutual information and attribute skewness, and do not always provide much insight.

The only fundamental reason why a learning algorithm performs well on a dataset is because its bias matches the data. Even the simplest of algorithms will perform brilliantly if the data is represented as the algorithm expects it to be. Therefore, we need better ways of describing the bias of learning algorithms, and define meta-features able to link *properties* of the data to *properties* of the learning algorithms.

Some developments do seem more related to a learner's bias. Landmarking provides an estimate of how well an algorithm's *representational bias* fits the data. Subsampling landmarkers also help by characterizing the shape of the learning curve of various learning algorithms. Indeed, learning curves are a very useful expression of how the learner's bias reacts to the given data (both representationally and procedurally) that may deserve more attention. Algorithm properties work the other way around: we take a known, easy to measure property of the data, and see how well each algorithm responds to it. Again, this is a very useful link between data properties and learning bias that deserves to be investigated more deeply.

Evaluation and advice When it comes to the applied performance metrics, we see that almost all studies use predictive accuracy. One could wonder why other metrics are ignored, especially since applying performance metrics is very fast. If other metrics had been recorded as well, these could be used to offer more targeted recommendations.

Finally, we can see that most studies have focussed on recommendation and ranking. This reflects the fact that they are performed with *practitioners* in mind. Very few studies are aimed at *designers* of learning algorithms: researchers who wish to understand *why* an algorithm performs well on a given dataset or not, and what can be done to improve algorithms.

This is also an issue we wish to resolve through the creation of experiment databases that allow a very wide range of queries on the behavior of learning algorithms.

Appendix A

Simple, Statistical and Information-Theoretic Data Properties

A.1 Some notes on notation

Let us introduce some notation that will be used to describe the characterizations we are going to use. For a continuous variable X, we denote its mean, standard deviation and variance by μ_X,σ_X and σ_X^2 respectively. The covariance of variables X and Y, indicating how much these two change together is denoted by σ_{XY}. For nominal attributes X, with I distinct values, and Y, with J distinct values, we can display their joint distribution in a contingency matrix with I rows and J columns, and denote each value as π_{ij}. We denote the marginal distributions of X and Y as $\pi_{i+} = \sum_j \pi_{ij}$ and $\pi_{+j} = \sum_i \pi_{ij}$. The conditional probability distribution of Y given X is denoted by $\pi_{j|i} = P(Y = y_j|X = x_i)$.

A.2 Simple features

The first set of meta-features exploit the fact that many algorithms are very sensitive to the number of instances in the data, the number of attributes, the number or missing values, the number of classes in classification, or the type of attributes (nominal, numerical or binary). Learning tends to become increasingly harder as the number of attributes increases, a principle known as the *curse of dimensionality*. Every added attribute adds a new dimension to the feature space, which means that the distance between observations in this space will become ever larger, and any unseen case we wish to predict will lie very far from the examples in the instance space. To attain the same 'density' of examples, we need an exponentially increasing amount of extra training points. Some algorithms, like nearest neighbor, are very sensitive to this problem, while decision tree learners, which only consider the most 'interesting' features, in terms of information gain, are much less so.

Furthermore, missing values can be *guessed*, e.g. using a costly low rank approximation, but this introduces uncertainty in the data. Many algorithms simply remove all instances with missing values, which wastes precious training points.

Also, some algorithms simply cannot handle some types of attributes and internally transform the data. For algorithms that are *numerically-challenged* (to be politically correct) numerical attributes can be replaced by nominal ones through *discretization*: separating the attribute's values in several intervals, thus losing some information. Vice versa, for *nominally-challenged* algorithms, each value of each nominal attribute can be replaced by a binary *indicator attribute*, indicating whether the example has that attribute's value or not, which often introduces many new attributes. The latter technique, although necessary for some algorithms, heavily increases the dimensionality of the data and may thus be disadvantageous for algorithms sensitive to the curse of dimensionality.

Most of these meta-features, indicated with an [s], were first identified in the StatLog project (Michie et al. 1994), an overview of which can also be found in Castiello et al. (2005). Meta-features indicated with a [k] were, to the best of our knowledge, identified by Kalousis (2002), and those indicated with a [r] are variants for regression which are useful when a meta-feature can only be used for classification, identified in Soares et al. (2004).

- n [s], the total number of instances (both training and test)
- $attr$ [s], the number of attributes (including target attribute)
- num [k], the number of numerical attributes
- nom [s], the number of nominal attributes
- bin [s], the number of binary attributes (including nominal attributes coded as indicator variables)
- $\%nom$ [k], the percentage of nominal attributes

$$\%nom = \frac{nom}{attr} \tag{A.1}$$

- $\%num$ [k], the percentage of numerical attributes

$$\%num = \frac{num}{attr} \tag{A.2}$$

- cl [s], the number of classes
- dim [k], the dimensionality of the dataset (defined by

$$dim = \frac{attr}{n} \tag{A.3}$$

- ex/cl, the number of examples for class (Aha 1992)
- $mvals$ [k], the number of missing values
- $\%mvals$ [k], the percentage of missing values

$$\%mvals = \frac{mvals}{attr \cdot n} \tag{A.4}$$

- *%outliers* [r], the proportion of attributes with outliers

Many different versions of these measures abound. For instance, Kalousis et al. (2004) uses $logn$, $log\frac{n}{attr}$ and $log\frac{n}{cl}$ instead.

A.3 Normality-related features

Some classification algorithms, like Naive Bayes and linear discriminants, assume that the values of the attributes are normally distributed within each class, so meta-features have been proposed to measure how non-normal the value distributions actually are:

- γ [s], the *skewness* or lack of symmetry in the distribution, or the 3rd standardized moment
$$\gamma = \frac{E(X - \mu_X)^3}{\sigma_X^3} \tag{A.5}$$
- β [s], the *kurtosis* or fatness of the distribution's tail, or the 4th standardized moment
$$\beta = \frac{E(X - \mu_X)^4}{\sigma_X^4} \tag{A.6}$$

A.4 Redundancy-related features

Furthermore, algorithms are also affected by the degree of redundancy in a dataset: if two or more attributes are dependent, they don't add much information and only increase the dimensionality of the dataset. This is measured by estimating the strength of the relationship between attributes:

- ρ_{XY} [s], the *correlation coefficient* measuring the association between two numerical attributes
$$\rho = \frac{\sigma_{XY}}{\sqrt{\sigma_X^2 \sigma_Y^2}} \tag{A.7}$$
- R_i [s], the *multiple correlation coefficient* measuring the maximal correlation coefficient between a numerical attribute X_i and some linear combination of all other numerical attributes $Z_i\alpha$, with $Z_i = (X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_{num})$ and α a non-zero vector.
$$R_i = argmax_{\alpha \neq 0} \frac{\sigma_{X_i Z_i \alpha}}{\sqrt{\sigma_{X_i}^2 \sigma_{Z_i \alpha}^2}} \tag{A.8}$$
- τ_{XY} [k], the *concentration coefficient* measuring the association between two nominal attributes, or the proportional reduction in the probability

of an incorrect guess predicting Y, with J distinct values, using X, with I distinct values

$$\tau_{XY} = \frac{\sum_i \sum_j \frac{\pi_{ij}^2}{\pi_{i+}} - \sum_j \pi_{+j}^2}{1 - \sum_j \pi_{+j}^2} \tag{A.9}$$

- $p_{val_{XY}}$ [k], the *p-value of the F-distribution* for a nominal attribute X with I values and a numeric attribute Y. The Analysis of Variance (ANOVA) examines how a numerical variable affects a nominal one by examining whether the means of the I groups defined on Y by X are different. The ratio of the *between group variance* and the *within group variance* $MS(B)/MS(W)$ follows the F-distribution and the p-value of that distribution gives the probability of observing that ratio under the assumption that the group means are equal (Kalousis 2002).

A.5 Attribute-target associations

Probably the most important property of an attribute is its association with the target attribute: the more the attribute tells us about the target attribute, the more useful it will be to model the data. As discussed above, irrelevant attributes add to the curse of dimensionality, and different learning algorithms exhibit different degrees of resilience against them (Hilario and Kalousis 2000b). If a dataset has many irrelevant attributes, this may indicate that it is better to use a feature selection or feature construction step first, or to choose an algorithm that, implicitly or explicitly, performs such operations internally. Most of these measures can be used only in classification.

- $H(X)$ [s], the *entropy* of a nominal[1] attribute X is a measure of the uncertainty (or randomness) associated with it. It measures the average information content one is missing when one does not know the exact value of X. If entropy is zero (if all values are the same), the attribute contains no information. The class entropy $H(C)$ is the amount of information required to specify the class of an instance, a measure for how 'informative' the attributes need to be. A low $H(C)$ means that the distribution of examples among classes is very skewed (containing some very infrequent classes) which some algorithms cannot handle well.

$$H(X) = -\sum_i \pi_{i+} log_2(\pi_{i+}) \tag{A.10}$$

[1]Although a definition exists for numerical distributions (using an integral instead of a summation), it is of no use for empirical data, and the entropy of numerical attributes (or targets) is calculated by discretizing the values in equal-length intervals (Michie et al. 1994).

- $H(X)_{norm}$ [s], the *normalized entropy* of a nominal attribute X rescales entropy to the [0..1] interval (Castiello et al. 2005)

$$H(X)_{norm} = \frac{H(X)}{log_2 n} \tag{A.11}$$

- $MI(Y,X)$ [s], the *mutual information* between nominal attributes X and Y describes the reduction in uncertainty of Y due to the knowledge of X, and leans on the conditional entropy $H(Y|X)$. It is also the underlying measure of the information gain metric used in decision tree learners.

$$\begin{aligned} MI(Y,X) &= H(Y) - H(Y|X) && \text{(A.12)} \\ H(Y|X) &= \sum_i p(X=x_i)H(Y|X=x_i) && \text{(A.13)} \\ &= -\sum_i \pi_{i+} \sum_j \pi_{j|i} log_2(\pi_{j|i}) && \text{(A.14)} \end{aligned}$$

- $UC(X,Y)$, the *uncertainty coefficient* is the mutual information between an attribute X and target attribute Y divided by the entropy of Y. It measures the proportional reduction in the statistical variance of Y when X is known (Agresti 2002). It is strongly related to the *information gain ratio* used in decision trees, which is defined as $UC(Y,X)$, or the proportional reduction in in the variance of X when target Y is known.

$$UC(X,Y) = \frac{MI(Y,X)}{H(X)} \tag{A.15}$$

- μ_{UC}, the *median of the uncertainty coefficients*, indicates the amount of information each attribute contains about the target Y (Kalousis et al. 2004).
- $\overline{MI(C,X)}$ [s], the *average mutual information of each attribute X with the class attribute C* is a simplified alternative for μ_{UC}

$$\overline{MI(C,X)} = \frac{\sum_{i=1}^{attr} MI(C,X_i)}{attr} \tag{A.16}$$

- EN-*attr* [s], the *equivalent number of attributes* is a quick estimate of the number of attributes required, on average, to describe the class (assuming independence).

$$EN\text{-}atrr = \frac{H(C)}{\overline{MI(C,X)}} \tag{A.17}$$

- NS-*ratio* [s], the *noise to signal ratio* is an estimate of the amount of non-useful information in the attributes regarding the class. $\overline{H(X)}$ is the average information (useful or not) of the attributes.

$$NS\text{-}ratio = \frac{\overline{H(X)} - \overline{MI(C,X)}}{\overline{MI(C,X)}} \tag{A.18}$$

The following are useful for numerical targets:

- $VarCoef_{target}$ [r], the coefficient of variation of the target is defined as the ratio of the standard deviation to the mean of the target attribute and can be used instead of entropy on numerical targets. It is a normalization of the standard deviation of the target useful for numerical targets X_{target}. A related measure, *sparsity of the target*, is $VarCoef_{target}$ discretized into 3 values.
$$VarCoef_{target} = \frac{\sigma_X}{\mu_X} \tag{A.19}$$
- $outliers_{target}$ [r], indicates the presence of outliers in X_{target}.
- $stationarity_{target}$ [r], indicates whether $\sigma_{X_{target}} > \mu_{X_{target}}$.
- $\rho_{XY_{target}}$, see Formula A.7, measures the correlation between a numerical attribute X and a numerical target Y_{target}.
- $p_{val_{XY_{target}}}$, see Section A.4, measures the correlation between a nominal attribute X and a numerical target Y_{target}.
- ρ_{max} [s], the *first canonical correlation coefficient* measures the association between all numerical attributes and a nominal (class) attribute. In principal component analysis (PCA), datasets are transformed into a new dataset with fewer dimensions (attributes). The first dimension, called the first principal component is a new axis in the direction of maximum variance. The variance of this principal axis is given by the largest eigenvalue λ_1. It thus measures how well the classes can be separated by the numerical attributes.
$$\rho_{max} = \sqrt{\frac{\lambda_1}{1+\lambda_1}} \tag{A.20}$$
- $frac1$ [s], the *fraction of the total variance retained in the 1-dimensional space defined by the first principal component* can be computed as the ratio between the largest eigenvalue λ_1 of the covariance matrix S and the sum of all its eigenvalues:
$$frac1 = \frac{\lambda_1}{\sum_i \lambda_i} \tag{A.21}$$

Even more statistical meta-features, based on the distances between examples and the probability density function (pdf) and cumulative distribution function (cdf) of the entire dataset are discussed in Ali and Smith-Miles (2006).

A.6 Algorithm-specific properties

One could also define some properties tailored to the bias of certain learning algorithms.

- *M* [s], *Box's M-statistic* measures the equality of the covariance matrices S_i of the different classes. If they are equal, then linear discriminants could be used, otherwise, quadratic discriminant functions should be used instead. As such, M predicts whether a linear discriminant algorithm should be used or not. In the following, $S_i = \frac{S_{c_i}}{n_i - 1}$ is the i class covariance matrix with S_{c_i} the i class scatter matrix and n_i the number of examples pertaining to class i, and $S = \frac{1}{n-cl}\sum_i S_{c_i}$ the pooled covariance matrix. It is zero when all individual covariance matrices are equal to the pooled covariance matrix.

$$M = \gamma \sum_i (n_i - 1) log \frac{|S|}{|S_i|} \tag{A.22}$$

$$\gamma = 1 - \frac{2num^2 + 3num - 1}{6(num+1)(cl-1)} (\sum_i \frac{1}{n_i - 1} - \frac{1}{n - cl}) \tag{A.23}$$

- *SD-ratio* [s], the *standard deviation ratio*, is a reexpression of M which is one if M is zero and strictly greater than one if the covariances differ.

$$SD\text{-}ratio = exp(\frac{M}{num \sum_i (n_i - 1)}) \tag{A.24}$$

A.7 Propositional versus relational features

Many of these features produce a value for each attribute, and some of them, like ρ_{XY}, compute a value for all pairs of continuous attributes, resulting in $O(2^{num})$ coefficients. If we are to use a propositional learner (one that expects a single table as its dataset) this results in a variable number of feature columns for each dataset. The solution of the STATLOG project, as well as subsequent studies (Brazdil et al. 1994; Lindner and Studer 1999; Soares and Brazdil 2000; Sohn 1999) was to simply take the average over all attributes (or all combinations). This means that some very different datasets may end up with exactly the same description. For instance, one dataset may have attributes which are either highly correlated (close to 1) or highly anti-correlated (close to -1), while another one may show correlations all close to zero: in both cases, the average correlation will be close to zero, while most algorithms will perform very differently on them (Kalousis 2002).

There are several ways of countering this problem. Todorovski et al. (2000) included the minimum and maximum value as well, and used feature selection techniques to lower the number of meta-features afterwards. Kalousis and Theoharis (1999) composed a *histogram* of the values in each set: for every meta-feature consisting of more than one value, its theoretical range of values is determined, e.g. [-1..1] in case of attribute correlation, and divided in ten equal length bins. As such, the correlation coefficients between all numerical attributes can be represented as $[\rho_1 \ldots \rho_{10}]$ on each dataset. One extra bin, in this case ρ_{NaN}, is added to signal the amount of outcomes that were impossible to compute. The same technique can also be used to signal the amount of missing values in each attribute, as this distribution has been shown to critically affect the performance of learning algorithms Kalousis and Hilario (2001b).

A more fundamental approach is to use a *relational learning algorithm* as a meta-learner, one that can read in relational data descriptions. Todorovski and Dzeroski (1999) used inductive logic programming (ILP) (De Raedt et al. 2008), allowing the meta-features to be stored in a relational representation (e.g. a relational database), and generating first order rules about the behavior of learning algorithms. A somewhat similar approach is proposed in Hilario and Kalousis (2001) and (Kalousis and Hilario 2003), where a case-based reasoning system is used instead.

References

Agresti, A. (2002). *Categorical Data Analysis.* Wiley Interscience.

Aha, D. (1992). Generalizing from case studies: A case study. *Proceedings of the Ninth International Conference on Machine Learning*, 1–10.

Aha, D., D. Kibler, and M. Albert (1991). Instance-based learning algorithms. *Machine Learning 6*(1), 37–66.

Ali, S. and K. Smith (2006). On learning algorithm selection for classification. *Applied Soft Computing Journal 6*(2), 119–138.

Ali, S. and K. Smith-Miles (2006). A meta-learning approach to automatic kernel selection for support vector machines. *Neurocomputing 70*(1-3), 173–186.

Arinze, B. (1994). Selecting appropriate forecasting models using rule induction. *Omega 22*(6), 647–658.

Asuncion, A. and D. Newman (2007). Uci machine learning repository. *University of California, School of Information and Computer Science.*

Bakker, B. and T. Heskes (2003). Task clustering and gating for bayesian multitask learning. *The Journal of Machine Learning Research 4*, 83–99.

Baxter, J. (1996). Learning internal representations. *Advances in Neural Information Processing Systems (NIPS).*

Bensusan, H. (1998). God doesn't always shave with occam's razor - learning when and how to prune. *Lecture Notes in Computer Science 1398*, 119–124.

Bensusan, H. and C. Giraud-Carrier (2000a). Casa batló is in passeig de gràcia or landmarking the expertise space. *Proceedings of the ECML-00 Workshop on Meta-Learning: Building Automatic Advice Strategies for Model Selection and Method Combination*, 29–46.

Bensusan, H. and C. Giraud-Carrier (2000b). Discovering task neighbourhoods through landmark learning performances. *Lecture Notes in Computer Science 1910*, 136–137.

Bensusan, H. and A. Kalousis (2001). Estimating the predictive accuracy of a classifier. *Lecture Notes in Computer Science 2167*, 25–36.

Blockeel, H. and L. Dehaspe (2000). Cumulativity as inductive bias. *Data Mining, Decision Support, Meta-Learning and ILP: Forum for Practical Problem Presentation and Prospective Solutions*, 61–70.

Blockeel, H., L. D. Raedt, and J. Ramon (2000). Top-down induction of clustering trees. *Proceedings of the 15th International Conference on Machine Learning*, 55–63.

Bock, H. and E. Diday (2000). Analysis of symbolic data: exploratory methods for extracting statistical information from complex data. *Springer*.

Brazdil, P., J. Gama, and B. Henery (1994). Characterizing the applicability of classification algorithms using meta-level learning. *Lecture Notes in Computer Science 784*, 83–102.

Brazdil, P., C. Giraud-Carrier, C. Soares, and R. Vilalta (2009). Metalearning: Applications to data mining. *Springer*.

Brazdil, P. and R. Henery (1994). Analysis of results. *Machine Learning, Neural and Statistical Classification, Chapter 10*.

Brazdil, P. and C. Soares (2000). A comparison of ranking methods for classification algorithm selection. *Lecture Notes in Computer Science 1810*, 63–75.

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

Brazdil, P., C. Soares, and R. Pereira (2001). Reducing rankings of classifiers by eliminating redundant classifiers. *Lecture Notes in Computer Science 2258*, 84–99.

Breiman, L. (1996). Bagging predictors. *Machine Learning 24*(2), 123–140.

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

Caruana, R. (1997). Multitask learning. *Machine Learning (Second Special Issue on Inductive Transfer) 28*(1), 41–75.

Castiello, C., G. Castellano, and A. Fanelli (2005). Meta-data: Characterization of input features for meta-learning. *Lecture Notes in Computer Science 3558*, 457–468.

Charnes, A., W. Cooper, and E. Rhodes (1978). Measuring the efficiency of decision making units. *European journal of operational research 2*, 429–444.

Clark, A. and C. Thornton (1997). Trading spaces: Computation, representation, and the limits of uninformed learning. *Behavioral and Brain Sciences 20*, 57–66.

De Raedt, L., P. Frasconi, K. Kersting, and S. Muggleton (2008). Probabilistic inductive logic programming. *Lecture Notes in Artificial Intelligence 4911*.

Dietterich, T. (1997). Machine-learning research - four current directions. *AI magazine*.

Dietterich, T. (2000). Ensemble methods in machine learning. *Lecture Notes in Computer Science.*

Dietterich, T. and G. Bakiri (1995). Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research 2*, 263–286.

Dietterich, T., D. Busquets, R. D. Màntaras, and C. Sierra (2002). Action refinement in reinforcement learning by probability smoothing. *Proceedings of the 19th International Conference on Machine Learning*, 107–114.

Dietterich, T. and E. Kong (1995). Machine learning bias, statistical bias, and statistical variance of decision tree algorithms. *Technical Report. Department of Computer Science, Oregon State University.*

Domingos, P. and M. Pazzani (1997). On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning 29*(2-3), 103–130.

dos Santos, P., T. Ludermir, and R. Prudêncio (2004). Selection of time series forecasting models based on performance information. *Proceedings of the 4th International Conference on Hybrid Intelligent Systems*, 366–371.

Dries, A. (2006). Dm square: Analyse van data-miningresultaten door middel van data mining. *Master's Thesis. Katholieke Universiteit Leuven.*

Evgeniou, T., C. Micchelli, and M. Pontil (2006). Learning multiple tasks with kernel methods. *Journal of Machine Learning Research 6*, 615–637.

Evgeniou, T. and M. Pontil (2004). Regularized multi-task learning. *Proceedings of the tenth ACM SIGKDD international conference on Knowledge Discovery and Data Mining.*

Ferri, C., P. Flach, and J. Hernández-Orallo (2004). Delegating classifiers. *Proceedings of the 21st international conference on machine learning*, 289–296.

Freund, Y. and R. Schapire (1996). Experiments with a new boosting algorithm. *Proceedings of the Thirteenth International Conference on Machine Learning*, 148–156.

Fürnkranz, J. and J. Petrak (2001). An evaluation of landmarking variants. *Working Notes of the ECML/PKDD 2001 Workshop on Integrating Aspects of Data Mining, Decision Support and Meta-Learning*, 57–68.

Fürnkranz, J., J. Petrak, P. Brazdil, and C. Soares (2002). On the use of fast subsampling estimates for algorithm recommendation. *Technical Report. Österreichisches Forschungsinstitut für Artificial Intelligence.*

Gama, J. and P. Brazdil (1995). Characterization of classification algorithms. *Lecture Notes in Computer Science 990*, 189–200.

Gama, J. and P. Brazdil (2000). Cascade generalization. *Machine Learning 41*(3), 315–343.

Geurts, P., O. Maimon, and L. Rokach (2005). Bias vs. variance decomposition for regression and classification. *Data mining and knowledge discovery handbook. Springer.*, 749–763.

Giraud-Carrier, C. (2005). The data mining advisor: meta-learning at the service of practitioners. *Proceedings of the 4th International Conference on Machine Learning and Applications*, 113–119.

Hengst, B. (2002). Discovering hierarchy in reinforcement learning with hexq. *Proceedings of the 19th International Conference on Machine Learning*, 243–250.

Hilario, M. and A. Kalousis (2000a). Building algorithm profiles for prior model selection in knowledge discovery systems. *Engineering Intelligent Systems 8*(2).

Hilario, M. and A. Kalousis (2000b). Quantifying the resilience of inductive classification algorithms. *Lecture Notes in Artificial Intelligence 1910*, 106–115.

Hilario, M. and A. Kalousis (2001). Fusion of meta-knowledge and meta-data for case-based model selection. *Lecture Notes in Computer Science 2168*, 180–191.

Iglezakis, I. and T. Reinartz (2002). Relations between customer requirements, performance measures, and general case properties for case base maintenance. *Lecture Notes in Computer Science 2416*, 247–257.

Janssen, F. and J. Fürnkranz (2007). On meta-learning rule learning heuristics. *Proceedings of the 7th IEEE Interational Conference on Data Mining*, 529–534.

Jorge, A. and P. Brazdil (1996). Architecture for iterative learning of recursive definitions. *Advances in Inductive Logic Programming. IOS Press.*

Kalousis, A. (2002). Algorithm selection via meta-learning. *PhD Thesis. University of Geneve.*

Kalousis, A., J. Gama, and M. Hilario (2004). On data and algorithms: Understanding inductive performance. *Machine Learning 54*, 275–312.

Kalousis, A. and M. Hilario (2001a). Feature selection for meta-learning. *Lecture Notes in Computer Science 2035*, 222–233.

Kalousis, A. and M. Hilario (2001b). Model selection via meta-learning: a comparative study. *International Journal on Artificial Intelligence Tools 10*(4), 525–554.

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

Kalousis, A. and T. Theoharis (1999). Noemon: Design, implementation and performance results of an intelligent assistant for classifier selection. *Intelligent Data Analysis 3*(4), 319–337.

Kaynak, C. and E. Alpaydin (2000). Multistage cascading of multiple classifiers: One man's noise is another man's data. *Proceedings of the 17th International Conference on Machine Learning*, 455–462.

Keogh, E. and J. Lin (2005). Clustering of time-series subsequences is meaningless: implications for previous and future research. *Knowledge and Information Systems 8*(2), 154–177.

Kohavi, R. and D. Wolpert (1996). Bias plus variance decomposition for zero-one loss functions. *Proceedings of the 1996 International Conference on Machine Learning (ICML'96).*

Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. *Biological cybernetics 43*(1), 59–69.

Köpf, C. and I. Iglezakis (2002). Combination of task description strategies and case base properties for meta-learning. *Proceedings of the 2nd international workshop on Integration and Collaboration Aspects of Data Mining, Decision Support and Meta-Learning (IDDM-2002)*, 65–76.

Köpf, C., C. Taylor, and J. Keller (2000). Meta-analysis: From data characterisation for meta-learning to meta-regression. *Proceedings of the PKDD2000 Workshop on Data Mining, Decision Support, Meta-Learning an ILP: Forum for Practical Problem Representtaion and Prospective Solutions.*, 15–26.

Kuba, P., P. Brazdil, C. Soares, and A. Woznica (2002). Exploiting sampling and meta-learning for parameter setting for support vector machines. *Proceeding of the Workshop Learning and Data Mining associated with Iberamia 2002, the 8th Iberoamerican Conference on Artificial Intelligence*, 209–216.

Lee, J.-W. and C. Giraud-Carrier (2008). Predicting algorithm accuracy with a small set of effective meta-features. *Seventh International Conference on Machine Learning and Applications (ICMLA08)*, 808–812.

Leite, R. and P. Brazdil (2004). Improving progressive sampling via meta-learning on learning curves. *Lecture Notes in Computer Science 3201*, 250–261.

Leite, R. and P. Brazdil (2005). Predicting relative performance of classifiers from samples. *Proceedings of the 22nd international conference on machine learning*, 497–504.

Leite, R. and P. Brazdil (2007). An iterative process for building learning curves and predicting relative performance of classifiers. *Lecture Notes in Computer Science 4874*, 87–98.

Ler, D., I. Koprinska, and S. Chawla (2005). Utilizing regression-based landmarkers within a meta-learning framework for algorithm selection. *Technical Report Number 569 School of Information Technologies University of Sydney*, 44–51.

Lindner, G. and R. Studer (1999). Ast: Support for algorithm selection with a cbr approach. *Lecture Notes in Computer Science 1704*, 418–423.

Michie, D., D. Spiegelhalter, and C. Taylor (1994). Machine learning, neural and statistical classification. *Ellis Horwood.*

Nakhaeizadeh, G. and A. Schnabl (1997). Development of multi-criteria metrics for evaluation of data mining algorithms. *Proceedings of the International Conference on Knowledge Discovery in Databases and Data Mining (KDD-97)*, 37–42.

Nakhaeizadeh, G. and A. Schnabl (1998). Towards the personalization of algorithms evaluation in data mining. *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining (KDD-98)*, 289–293.

Niculescu-Mizil, A. and R. Caruana (2005). Learning the structure of related tasks. *Proceedings of NIPS-2005 Workshop on Inductive Transfer: 10 Years Later.*

Ortega, J., M. Koppel, and S. Argamon (2001). Arbitrating among competing classifiers using learned referees. *Knowledge and Information Systems 3*(4), 470–490.

Parmanto, B., P. Munro, and H. Doyle (1996). Improving committee diagnosis with resampling techniques. *Advances in Neural Information Processing Systems 8*, 882–888.

Peng, Y., P. Flach, P. Brazdil, and C. Soares (2002). Decision tree-based data characterization for meta-learning. *ECML/PKDD'02 workshop on Integration and Collaboration Aspects of Data Mining, Decision Support and Meta-Learning*, 111–122.

Peng, Y., P. Flach, C. Soares, and P. Brazdil (2002). Improved dataset characterisation for meta-learning. *Lecture Notes in Computer Science 2534*, 141–152.

Pfahringer, B., H. Bensusan, and C. Giraud-Carrier (2000). Meta-learning by landmarking various learning algorithms. *Proceedings of the Seventeenth International Conference on Machine Learning*, 743–750.

Prudêncio, R. and T. Ludermir (2004). Meta-learning approaches to selecting time series models. *Neurocomputing 61*, 121–137.

Quinlan, J. (1992). Learning with continuous classes. *Proceedings of the 5th Australian Joint Conference on Artificial Intelligence*, 343–348.

Quinlan, J. and R. Cameron-Jones (1993). Foil: A midterm report. *Lecture Notes in Artificial Intelligence 667*, 3–20.

Raina, R., A. Ng, and D. Koller (2005). Transfer learning by constructing informative priors. *Neural Information Processing Systems (NIPS) Workshop on Inductive Transfer.*

Reinartz, T., I. Iglezakis, and T. Roth-Berghofer (2001). Review and restore for case-base maintenance. *Computational Intelligence 17*(2), 214–234.

Rendell, L., R. Seshu, and D. Tcheng (1987). Layered concept learning and dynamically-variable bias management. *Proceedings of the 10th International Joint Conference on Artificial Intelligence*, 308–314.

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

Rosenstein, M., Z. Marx, L. Kaelbling, and T. Dietterich (2005). To transfer or not to transfer. *Workshop on Inductive Transfer: 10 Years Later at Neural Information Processing Systems (NIPS).*

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

Sharkey, N. and A. Sharkey (1993). Adaptive generalization. *Artificial Intelligence Review 7*, 313–328.

Silver, D. and R. Mercer (1996). The parallel transfer of task knowledge using dynamic learning rates based on a measure of relatedness. *Connection Science 8*(2), 277–294.

Smith, K., F. Woo, V. Ciesielski, and R. Ibrahim (2001). Modelling the relationship between problem characteristics and data mining algorithm performance using neural networks. *Smart Engineering System Design: Neural Networks, Fuzzy Logic, Evolutionary Programming, Data Mining, and Complex Systems 11*, 356–362.

Smith, K., F. Woo, V. Ciesielski, and R. Ibrahim (2002). Matching data mining algorithm suitability to data characteristics using a self-organising map. *Hybrid Information Systems. Physica*, 169–180.

Smith-Miles, K. (2008a). Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys (CSUR) 41*(1), Article 6.

Smith-Miles, K. (2008b). Towards insightful algorithm selection for optimisation using meta-learning concepts. *Proceedings of the IEEE International Joint Conference on Neural Networks*, 4118–4124.

Soares, C. (2004). Learning rankings of learning algorithms. *PhD Thesis. Department of Computer Science. University of Porto..*

Soares, C. (2009). Uci++: Improved support for algorithm selection using datasetoids. *Lecture Notes in Computer Science 5476*, 499–506.

Soares, C. and P. Brazdil (2000). Zoomed ranking: Selection of classification algorithms based on relevant performance information. *Lecture Notes in Computer Science 1910*, 126–135.

Soares, C. and P. Brazdil (2006). Selecting parameters of svm using meta-learning and kernel matrix-based meta-features. *Proceedings of the 2006 ACM symposium on Applied computing*, 564–568.

Soares, C., P. Brazdil, and P. Kuba (2004). A meta-learning method to select the kernel width in support vector regression. *Machine Learning 54*, 195–209.

Soares, C., J. Petrak, and P. Brazdil (2001). Sampling based relative landmarks: Systematically testdriving algorithms before choosing. *Lecture Notes in Computer Science 3201*, 250–261.

Soares, R., T. Ludermir, and F. D. Carvalho (2009). An analysis of meta-learning techniques for ranking clustering algorithms applied to artificial data. *Lecture Notes in Computer Science 5768*, 131–140.

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

Thrun, S. and T. Mitchell (1995). Learning one more thing. *Proceedings of the International Joint Conference on Artificial Intelligence*, 1217–1223.

Thrun, S. and L. Pratt (1998). Learning to learn. *Kluwer Academic Publishers*.

Todorovski, L., H. Blockeel, and S. Dzeroski (2002). Ranking with predictive clustering trees. *Lecture Notes in Computer Science 2430*, 444–455.

Todorovski, L., P. Brazdil, and C. Soares (2000). Report on the experiments with feature selection in meta-level learning. *Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD-2000) Workshop on Data mining, Decision support, Meta-learning and ILP*, 27–39.

Todorovski, L. and S. Dzeroski (1999). Experiments in meta-level learning with ilp. *Lecture Notes in Computer Science 1704*, 98–106.

Todorovski, L. and S. Džeroski (2003). Combining classifiers with meta decision trees. *Machine Learning 50*(3), 223–250.

Tsuda, K., G. Ratsch, S. Mika, and K. Muller (2001). Learning to predict the leave-one-out error of kernel based classifiers. *Lecture Notes in Computer Science 2130*, 331–338.

Van Der Putten, P. and M. Van Someren (2004). A bias-variance analysis of a real world learning problem: The coil challenge 2000. *Machine Learning 57*, 177–195.

Van Someren, M. (2001). Model class selection and construction: Beyond the procrustean approach to machine learning applications. *Lecture Notes in Computer Science 2049*, 196–217.

Vanschoren, J. (2007). Meta-learning on experiment databases: learning to understand automatic learning mechanisms. *High Performance Computing @ K.U.Leuven Symposium.*

Vanschoren, J. and H. Blockeel (2006). Towards understanding learning behavior. *Proceedings of the Fifteenth Annual Machine Learning Conference of Belgium and the Netherlands (Benelearn06)*, 89–96.

Vanschoren, J. and H. Blockeel (2008). Investigating classifier learning behavior with experiment databases. *Data Analysis, Machine Learning and Applications: 31st Annual Conference of the Gesellschaft für Klassifikation*, 421–428.

Vanschoren, J. and H. Blockeel (2009a). A community-based platform for machine learning experimentation. *Lecture Notes in Artificial Intelligence 5782*, 750–754.

Vanschoren, J. and H. Blockeel (2009b). Stand on the shoulders of giants. towards a portal for collaborative experimentation in data mining. *Proceedings of the SoKD-09 International Workshop on Third Generation Data Mining at ECML PKDD 2009*, 88–99.

Vanschoren, J., H. Blockeel, and B. Pfahringer (2008). Experiment databases: Creating a new platform for meta-learning research. *Proceedings of the ICML/UAI/COLT Joint Planning to Learn Workshop (PlanLearn08)*, 10–15.

Vanschoren, J., H. Blockeel, B. Pfahringer, and G. Holmes (2008). Organizing the world's machine learning information. *Communications in Computer and Information Science 17*, 693–708.

Vanschoren, J., B. Pfahringer, and G. Holmes (2008). Learning from the past with experiment databases. *Lecture Notes in Artificial Intelligence 5351*, 485–492.

Vanschoren, J., A. Van Assche, C. Vens, and H. Blockeel (2007). Meta-learning from experiment databases: An illustration. *Proceedings of the 16th Annual Machine Learning Conference of Belgium and The Netherlands (Benelearn07)*, 120–127.

Venkatachalam, A. and J. Sohl (1999). An intelligent model selection and forecasting system. *Journal of Forecasting 18*(3), 167–180.

Vilalta, R. (1999). Understanding accuracy performance through concept characterization and algorithm analysis. *Proceedings of the 1999 International Conference on Machine Learning (ICML1999). Workshop on Recent Advances in Meta-Learning and Future Work.*

Vilalta, R. and Y. Drissi (2002a). A characterization of difficult problems in classification. *Proceedings of the International Conference on Machine Learning and Applications*.

Vilalta, R. and Y. Drissi (2002b). A perspective view and survey of meta-learning. *Artificial Intelligence Review*.

Vilalta, R., C. Giraud-Carrier, and P. Brazdil (2005). Meta-learning: Concepts and techniques. *The Data Mining and Knowledge Discovery Handbook, Chapter 33*.

Wang, X., K. Smith, and R. Hyndman (2006). Characteristic-based clustering for time series data. *Data Mining and Knowledge Discovery 13*(3), 335–364.

Wang, X., K. Smith-Miles, and R. Hyndman (2009). Rule induction for forecasting method selection: Meta-learning the characteristics of univariate time series. *Neurocomputing 72*(10-12), 2581–2594.

Wettschereck, D., D. Aha, and T. Mohri (1997). A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms. *Artificial Intelligence Review 11*(1-5), 273–314.

Wolpert, D. (1992). Stacked generalization. *Neural networks 5*(2), 241–259.