KATHOLIEKE UNIVERSITEIT
LEUVEN

Arenberg Doctoral School of Science, Engineering & Technology
Faculty of Engineering
Department of Computer Science

Understanding Machine Learning Performance with Experiment Databases

ir. Joaquin Vanschoren

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor in Engineering

May 2010

Understanding Machine Learning Performance with Experiment Databases

Joaquin Vanschoren

Jury:
Prof. Dr. ir. J. Berlamont, president
Prof. Dr. ir. H. Blockeel, promotor
Prof. Dr. M. Bruynooghe
Prof. Dr. ir. J. Suykens
Prof. Dr. P. Brazdil
(University of Porto, Portugal)
Prof. Dr. G. Holmes
(Waikato University, New Zealand)
Prof. Dr. J. Kok
(Universiteit Leiden, The Netherlands)

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor in Engineering

UDC 681.3∗I26

May 2010

D/2010/7515/46
ISBN 978-94-6018-206-8

Beknopte samenvatting

Het onderzoek in automatische leermethoden en *data mining* kan aanzienlijk versneld worden wanneer we experimentele onderzoeksresultaten kunnen uploaden naar het internet en verzamelen in applicaties die de data filteren en organiseren. De massale stroom aan experimenten die uitgevoerd worden om nieuwe algoritmes te vergelijken, onderzoekshypothesen te testen en data te modelleren is wellicht erg bruikbaar in verder onderzoek, maar wordt momenteel weggegooid of vergeten. In deze thesis ontwikkelen we een infrastructuur om automatisch experimenten te exporteren naar *experiment databanken*, databanken die specifiek ontworpen zijn om de details van grote hoeveelheden experimenten, uitgevoerd door verschillende onderzoekers, te verzamelen en om databank queries op te stellen over nagenoeg elk aspect van het leergedrag van leeralgoritmes. Ze kunnen opgezet worden voor persoonlijk gebruik, om resultaten te delen binnen een onderzoekslabo, of om publieke, globale databanken op te zetten.

Gelijkaardige trends bestaan in vele andere onderzoeksgebieden, en wij volgen een gelijkaardige strategie door eerst een formeel domeinmodel, een ontologie, op te stellen. Daarna gebruiken we deze ontologie om een XML-gebaseerde taal af te leiden voor het uitwisselen van experimenten, alsook om databankmodellen af te leiden om alle gedeelde resultaten te organiseren.

Ten slotte tonen we aan hoe zulke databanken gebruikt kunnen worden om te *meta-leren*: om nieuwe inzichten te bekomen in het leergedrag van leeralgoritmes. Door vaak niet meer dan een enkele databank query te gebruiken hebben we vaak verrassende nieuwe resultaten bekomen. Zo hebben we gedetailleerde rangschikkingen van experimenten opgesteld, hebben we inzicht gekregen in het gedrag van *ensemble*-methoden, hebben we verbeteringen voorgesteld voor bepaalde algoritmes, hebben we leercurve-analyses uitgevoerd en hebben we inzicht gekregen in het *bias-variance* gedrag van leeralgoritmes. We hebben ook meta-modellen gebouwd om geschikte leeralgoritmes en parameterinstellingen te voorspellen en te verklaren.

Dit illustreert dat er veel geleerd kan worden door voorgaande leerexperimenten te verzamelen en te hergebruiken, en dat het bouwen van experiment databanken om deze experimenten te bevragen een zeer doeltreffende techniek is om deze data te exploiteren. Vaak leiden zulke analyses tot verrassende nieuwe inzichten, of interessante nieuwe onderzoeksvragen.

Abstract

Research in machine learning and data mining can be speeded up tremendously by moving empirical research results out of people's heads and labs, onto the network and into tools that help us structure and filter the information. The massive streams of experiments that are being executed to benchmark new algorithms, test hypotheses or model new datasets have many more uses beyond their original intent, but are often discarded or their details are lost over time. In this thesis, we developed a framework to automatically export experiments to *experiment databases*, databases specifically designed to collect all the details on large numbers of past experiments, performed by many different researchers, and to compose queries about almost any aspect of the behavior of learning algorithms. They can be set up for personal use, to share results within a lab, or to build community-wide repositories.

Following similar developments in several other sciences, we first define a formal domain model, an ontology, for experimentation in machine learning, after which we use this ontology to define an XML-based language to exchange experiments, as well as a database model to organize all submitted results.

Finally, we demonstrate how such databases can be queried to *meta-learn*: to gain new insight into learning algorithm behavior. Using often no more than a single database query, we obtained surprising new results. This includes detailed rankings of learning algorithms, insight into the behavior of ensemble methods, suggestions for improvement of certain algorithms, learning curve analyses and insight into the bias-variance behavior of algorithms. We also built meta-models for predicting and explaining the suitability of learning algorithms and parameter settings. This illustrates that much can be learned by collecting and reusing past machine learning experiments, and that building experiment databases to query for them provides an effective way of tapping into this information, often yielding surprising new insight or generating interesting new research questions.

Acknowledgements

Although I didn't realize it at the time, this story started off on a chilly evening in March, as I hurried into an artist's bar in Leuven one evening to meet the *A.I. Reading Club* for the first time. I was writing my Master's thesis at the time, and my daily supervisor, Robby Goetschalckx, convinced me to check it out. It was there that I met the first batch of whimsy and bright colleagues that travelled with me along the road, all of which, I presume, are still trying to get through the reading club's first book: Gödel, Escher, Bach. It was also there that I first met my supervisor, Hendrik Blockeel, in person.

Skipping ahead six months, around the same late hour, I was working in my office across the hall from Hendrik, when he came in holding my PhD proposal, saying he had the perfect solution for all my (scientific) problems: *experiment databases*. Thanks, Hendrik, for giving me a great idea and letting me run with it - even though many thought the idea to be overly ambitious, we showed them :). He has been a great source of inspiration, motivation and insight during these years, teaching me to challenge what is known, to pry open black boxes, and to keep an eye on the bigger picture. He always allowed me to drag him out of his office to show him a surprising new result, and travelled with me by plane, car, rickshaw, junk and rowing boat to far-away conferences. Finally, his eternal good humor and patience helped me soothe over the rough patches of a PhD, making it almost seem easy :).

I also wish to thank Luc De Raedt for encouraging me to spread the word, and Maurice Bruynooghe for his trust and leadership.

Along me in the trenches was a small army of colleagues that was always there when I needed them. While I cannot possibly comment on all of them here, they are all valiant heroes in their own respect, and created a bustling, creative atmosphere that proved invaluable for nurturing many of the ideas in this work.

Learning me the tricks of the trade whilst creating a happy ambiance, there were my first office-mates: Anneleen Van Assche, Celine Vens, Werner Uwents and Stefan Raeymaekers. Anneleen and Celine (a.k.a. Hendrik's girls) happily allowed me to distract them with tricky questions about ensemble algorithms, databases or other meta-topics. Later, in the 'girl's office', there was Elisa Fromont who, aside from being contagiously enthusiastic, taught me how to play poker and many other games, and gladly cared for the cat when I was

abroad. With Parisa Kordjamshidi I had many interesting discussions, quite a few about getting to grips with parenthood, and finally, Wannes Meert and Nima Taghipour helped me through the last grueling months of thesis writing.

Every once in a while there was the familiar clatter of collapsing stacks of soda cans from Siegfried Nijssen's office, which I frequented regularly to fire off questions no one else could answer (or simply to borrow the couch). Albrecht Zimmermann and Björn Bringmann inspired me with their almost German efficiency and dazzling presentation skills, and Anton Dries manned the Genius bar whenever I ran into computer trouble. Tias Guns recharged me with youthful enthusiasm whenever mine started to wear off, and Fabrizio Costa made sure my whiteboard was always full of new problems to think about, walking in with a "Hey, man!" and launching new ideas. Finally, Maarten Mariën, Joost Vennekens, Leander Schietgat and Eduardo Costa helped me to stay fit through regular running and tennis sessions during the lunch break. I promise to show up more often now the thesis is finished!

I was also lucky to meet many accomplished, inspiring researchers who were very supportive of my dreams, and invited me to come work with them. First, there is the Waikato gang: Geoffrey Holmes, Bernhard Pfahringer and Eibe Frank. They made my stay in New Zealand so fruitful and enlightening that three months abroad seemed to fly by in a flash. Geoff regularly got my day off to a great start by sending me small emails entitled "Inspiration", "Thoughs?" or "Further pursuits", in which he challenged me to answer open questions or to verify surprising studies. He taught me to "boldly go where no machine learning researcher has gone before" and, more importantly, he also taught me how to make a mean cappuccino. Bernhard regularly amazed me by coming up with the most perfect explanations for the weirdest of results. Not a week went by without interesting new findings. Peter Reutemann and Dale Fletcher were great office mates and inspired me with their practical solutions to many problems. Furthermore, making sure I got the best out of my time in New Zealand, Jesse, Robert, Michelle, Edmund and Albert regularly invited me to climb a mountain or to go on short trips. I fondly remember the day we drove 300 kilometers through a breathtaking landscape to go skiing on an active volcano, and stopping on the way back to buy swimsuits and do some impromptu hot pooling. Finally, a very special thank you goes out to Kim, Tim, Rebecca and Isobel, who warmly welcomed me into their family, even though I was only supposed to feed their cat :). I had a great time, guys.

If any scientific project inspired me in particular, it was the Robot Scientist project in Aberystwyth. I was very lucky to work with Ross King, Larisa Soldatova and Amanda Clare, whose views on the next generation of scientific discovery had a profound impact on this thesis. Larisa taught me everything I know about ontology design, and was remarkably easy to work with. I also

thank Qi, Wayne, Andrew, Ken and Jem for the many interesting discussions I had with them.

My last visit, to Ljubljana, was shorter but very efficient thanks to Sašo Džeroski and Panče Panov, and Dragi, Ivica, Daniela and Violeta filled my evenings with laughter.

Besides Hendrik, Maurice and Geoff, I am very honored to have three more outstanding researchers serving on my Ph.D. committee: Johan Suykens, Pavel Brazdil and Joost Kok. Their detailed feedback contributed greatly to the quality of the final text. I also thank Prof. Berlamont for chairing my defence.

I gratefully acknowledge the financial support received for the work performed during this thesis from the Flemish government in GOA grant 2003/08 Inductive Knowledge Bases and from the Flemish Fund for Scientific Research (F.W.O.-Vlaanderen) grant G.0108.06 Foundations of Inductive Databases for Data Mining. F.W.O.-Vlaanderen also supported me with two travel grants. Although also often taken for granted, I would like to thank the sysadmins at the Computer Science department and the LUDIT High Performance Computing Cluster, without whose support I could not possibly have performed the massive amounts of experiments in this thesis. Incidentally, I also thank my laptop for literally breaking in half on it's own accord, two weeks before I had to send this text in, greatly adding to the thrill of research.

Last but definitely not least, I wish to thank my family and friends. I'd especially like to thank my parents, for misunderstanding all my ideas but nodding and smiling all the same, and for not minding me working on Christmas morning too much. *Bedankt voor jullie onvoorwaardelijke steun en nooit aflatende geloof in mij!* A special thank you goes to Sarah and Monique as well.

Next, I'd like to thank just a few of my friends, several of which are currently writing up their thesis as well: Adriaan, Wim, Jan & Lore, Joris & Iris, Wouter & Tine, Liesbeth & Thomas, Eveline & Edwin, Yves & Paula, Lies & Jozef, Nele & Robin, Lukas & Joke, Joris & Marijke, An & Markske, Isabelle & Joeri, Karen & Bart, Dominic & Audrey, Lieven & Sofie and Katrien & Sven. Thanks for all the great and funny moments!

Beyond the power of words, I'd like to thank Veerle, my love. Even when I had to burn the midnight oil, or basically any oil available, she was eternally supportive and when things looked grim, she always brought back the sunshine. She's the bee's knees.

She also gave me a beautiful son, Kobe, now eight months old. Every morning, Kobe's excited smile magically makes me realize the beauty of the world again.

No matter how tired or weary, hearing him say ‘papwa’ always brings me back to life. He taught me that, in a world with seemingly unsurmountable challenges, everything begins with baby steps. Kobe, I hope you can read this soon - there are some pretty pictures in the back ;).

Joaquin Vanschoren
Leuven, May 2010

Contents

List of Figures

List of Tables

A learning experience is one of those things that say:
"You know that thing you just did? Don't do that."

Douglas Adams

Chapter One

Introduction

1.1 Machine Learning

The burgeoning field of Machine Learning tries to answer the following question:

> How can we build computer systems that automatically improve with experience, and what are the fundamental laws that govern all learning processes? (Mitchell 2006)

Probably the most quintessential type of learning is the discovery of patterns in series of observations, called *knowledge discovery*. It is the cornerstone of empirical science: in the 16th century, the detailed astronomical observations of Tycho Brahe allowed Johannes Kepler to discover the empirical laws of planetary motion, and nowadays, machine learning techniques are being used to automatically discover unusual astronomical phenomena in continuous streams of images generated by earth-bound and space telescopes. Using computer algorithms, any hidden regularities can be automatically detected and employed to generate increasingly accurate predictions as more data (experience) is made available. Indeed, many data-intensive empirical sciences now depend on machine learning techniques to accelerate the discovery of patterns in experimental data, for instance to pinpoint the functions of individual genes in living cells, to discover which molecules are active against diseases, and to build highly accurate profiles of internet search engine users and online shoppers.

The second part of the question goes beyond building learning systems, and tries to discover regularities in the behavior of learning processes themselves. The aim here is to express how learning systems are affected by specific properties of the data they encounter. This can be done *theoretically*, leading to an interesting body of work called *computational learning theory*, or *empirically*, by discovering patterns in the performance of existing learning systems. The latter is called *meta-learning*, and is the focus of this text.

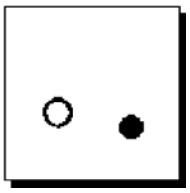
First, Section 1.2 will introduce some basic concepts, after which Section 1.3 explains the importance of meta-learning, leading up to the goal of this thesis stated in Section 1.4. Finally, an outline of the thesis with an overview of its most important contributions is provided in Section 1.6.

1.2 Knowledge Discovery: An illustration

In this section, we use a running example to introduce some key concepts of the knowledge discovery process. It may be skipped by those already well-versed in this area.

Figure 1.1 shows 3 different examples of *Bongard problems*[1], named after the Russian computer scientist M. M. Bongard. A typical problem consists of 6 figures conforming to a specific, hidden rule (shown on the left), as well as 6 counterexamples, which do *not* conform to that rule (but sometimes conform to the negation of it). The ordering of the boxes has no meaning, i.e. all the boxes on the left side of each problem can be scrambled at will, and there is also nothing magical about the number 6: some variants exist that have more examples or that have more positive than negative examples. Can you discover the hidden patterns?

This is an example of a *supervised classification problem*, a problem in which each observation (each of the boxes) is *labeled* with a classification of that observation. In this case there are two classes: positive (conforming to the hidden rule) or negative. The goal is to learn the *target concept* (the hidden rule) based on the given observations, and to use it to predict the class of future, unlabeled observations, such as this one:



[1] For an in-depth discussion of Bongard problems, and a very fundamental approach to solving them, see Foundalis (2006)

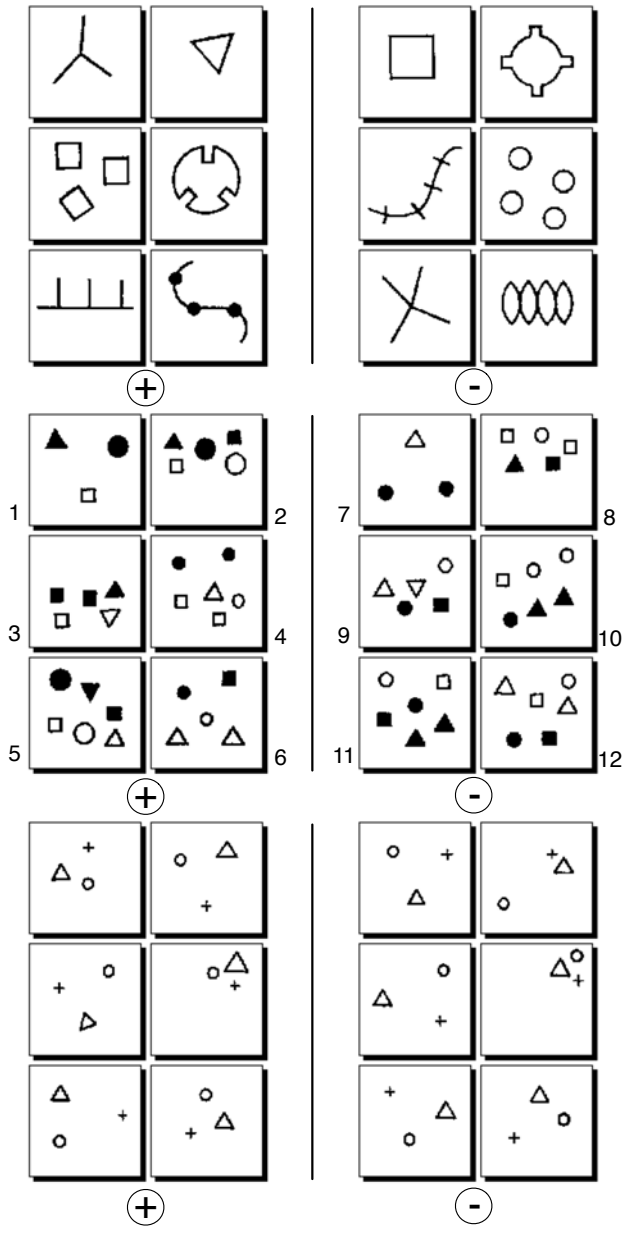


Figure 1.1: Three Bongard problems (#91,#48 and #54). Adapted from Foundalis (2006)

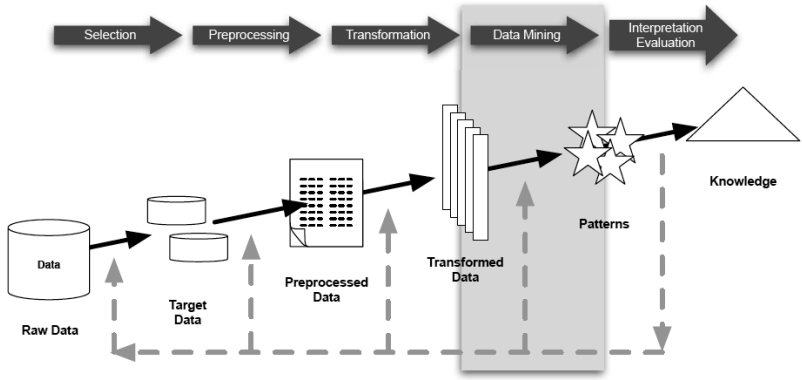


Figure 1.2: An overview of the steps composing the KDD process. Adapted from Fayyad et al. (1996)

1.2.1 The knowledge discovery process

To allow a learning algorithm to learn from the given observations, we first need to convert the images into a representation the algorithm understands and that (hopefully) still contains the information needed to discover the hidden pattern. Since the source data is typically stored in databases, this combination of database techniques and machine learning is called *knowledge discovery in databases (KDD)*, described by Fayyad et al. (1996) as follows:

> KDD is the nontrivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data.

As illustrated in Figure 1.2, it is a multi-step process:

Data selection First, we need to decide which aspects of the problem should be taken into account. Here, we need the boxed images and the labels.

Data preprocessing Next, we need to extract useful features from the raw data. In this case, we need to use image processing techniques, e.g. to detect edges, corners, curvature and entire shapes (such as triangles and circles).

Data transformation Next, we need to select which of the many generated features are most useful depending on the goal of the task. Possible approaches here are *feature selection* techniques which check which features seem to correlate with the target attribute, and *dimensionality reduction* techniques which employ statistical transformations of the data to yield new, but fewer, features composed by combining several other ones.

Data mining The actual learning step in the knowledge discovery process is called *data mining*, in which we employ a learning algorithm to build a model of the prepared data. As we shall discuss shortly, different algorithms will perform very differently on different data configurations, and selecting and modifying learning algorithms is a very involved process.

Table 1.1: A dataset representing the second Bongard problem.

#	# △	$\frac{\#black\triangle}{\#\triangle}$	$\frac{\#\triangle}{\#shapes}$	...	$\frac{\sum_i y(\blacklozenge)}{\#\blacklozenge}$	$\frac{\sum_i y(\diamondsuit)}{\#\diamondsuit}$	class
1	1	1	0.33	...	0.75	0.25	+
2	1	1	0.2	...	0.8	0.6	+
3	2	0.5	0.4	...	0.5	0.25	+
4	1	0	0.17	...	0.8	0.4	+
5	2	0.5	0.33	...	0.75	0.35	+
6	2	0	0.4	...	0.7	0.25	+
7	1	0	0.33	...	0.3	0.8	-
8	1	1	0.2	...	0.65	0.8	-
9	2	0	0.4	...	0.4	0.55	-
10	2	1	0.33	...	0.3	0.6	-
11	2	1	0.33	...	0.45	0.85	-
12	2	0	0.33	...	0.25	0.7	-
13	0	0	0	...	0.3	0.4	?

Interpretation Finally, the model produced by the learning algorithm needs to be evaluated for correctness and interpreted in order to speak of 'knowledge'. Typically, the first models will not be satisfactory, and the whole process will typically be reiterated and adjusted repeatedly.

1.2.2 Data preprocessing

We now try to learn the target concept behind the second Bongard problem in Figure 1.1. We assume the data preprocessing and transformation steps have been completed, providing us with an *attribute-value representation* of the given images. This is the simplest form of representing data: a table in which each row is one observation (called an *instance*) and each column is a measurable property (called an *attribute* or a *feature*) of each of the figures. In Table 1.1, we have numbered the boxes as indicated in Figure 1.1 (number 13 is the unlabeled example shown above), and show the values of the following features:

- The number of triangles (the same can be done for squares and circles)
- The ratio of black triangles over all triangles
- The ratio of triangles over all shapes
- The average Y-coordinate (between 0 and 1) of all black shapes
- The average Y-coordinate (between 0 and 1) of all white shapes
- The *target feature*: positive, negative or unknown

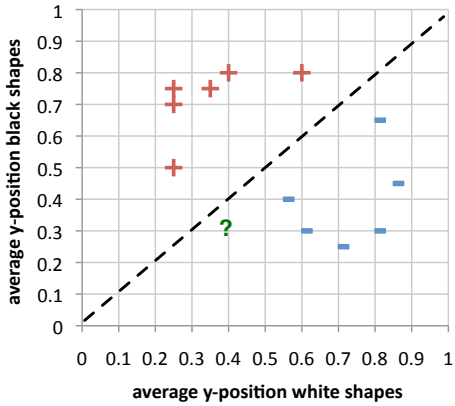


Figure 1.3: A scatterplot showing our observations using only the last two features and the hidden target concept (dashed line).

This representation transforms each observation into a single point in a high-dimensional space spanned by the identified features. The subspace spanned by the last two descriptive features is shown in Figure 1.3.
The target concept may be clear by now: in the positive examples, all black shapes are positioned above the white shapes and vice versa. With these two dimensions, the target concept is quite straightforward. If we generate many more examples beyond the 12 given by Bongard, all positive examples will be above the dashed diagonal line and all negative ones below.
In real-world situations, the features will usually not be this predictive, and many more features may have to be combined in order to provide a good model of the data. Also, more often than not, there is noise in the data: the labels or attribute values of some instances might be wrong, meaning that a completely correct solution is impossible, and that we must be careful not to model this noise (a problem known as *overfitting*). Finally, there may easily be millions of examples, such as credit card transactions or web searches.
We can now provide a more formal definition of our learning problem:

> Supervised classification is the task of learning from a set of classified training examples $(x, c(x))$, where $x \in X$ (the instance space) and $c(x) \in C$ (a finite set of classes), a classifier function $f : X \to C$ such that f approximates c (the target function) over X.

1.2.3 Modeling the data

One possible learning algorithm to address this classification problem is a *decision tree learner*, which assumes that the data can be modeled using a decision tree, as illustrated in Figure 1.4.

1.2.3.1 Training

A decision tree is *trained* by recursively choosing a feature and generating a test on that feature that splits the data in multiple parts. The quality of a split is defined by how cleanly it separates the values of the target feature, in this case how well the positive examples are separated from the negative ones. There exist several ways to calculate this, but perhaps the most common is the information-theoretic metric of *information gain ratio*, which quantifies to which degree the separation of labels is cleaner than before the extra split was added (a formal definition is given in Appendix A). We start off with a single node containing all observations, and after each split, the data points move down to their respective branch, and stored in so-called *leaf nodes*. The model is then further refined by choosing a leaf node and splitting it further.
When we train a decision tree learner[2] with our 12 examples, this yields the decision tree shown in Figure 1.4, consisting of two splits: first on the 'average Y-coordinate of all white shapes', then on the 'average Y-coordinate of all black shapes'. These splits are also shown in Figure 1.5: the first split, the vertical line, already yields a good model (only 1 example was misclassified), while the second split further refines it. Also shown is the *decision boundary*, separating positive from negative examples according to the final model.

1.2.3.2 Interpretation and testing

The decision tree's solution to the Bongard problem is: "The average Y-position of the white shapes is smaller or equal to 0.4, or otherwise the average Y-position of the black shapes is larger than 0.7". While this is correct according to Bongard's 12 examples, it is only an approximation of the actual concept, indicated by the diagonal line in Figure 1.3. Indeed, when looking at our test example (the one whose label we want to predict), we see that the decision tree learner classifies it as a positive example, while it is, in fact, a negative one.
To be fair, the decision tree learner could have approximated the target concept better if it was given more examples. In Figure 1.6 we show a problem with a similar target concept (a non-axis-parallel straight line) and 200 examples. We see that also in this case, the decision tree approximates the target with a step-like function. Since it can only make axis-parallel splits, it will never be able to match the target concept exactly, and can only approximate it by building a very complex decision tree, resulting in very tiny steps. However, another data transformation step (see Section 1.2.1) could have created a feature equal to the *ratio* of the final two descriptive features, in which case the decision boundary would have become an axis-parallel line, and a decision tree with a single node would have sufficed to correctly model it. The performance of a learning algorithm thus depends greatly on specific properties of the data at hand, and understanding such relationships is a key aim of this thesis.

[2]We used WEKA's J48 implementation, with the minimal leaf size set to 1.

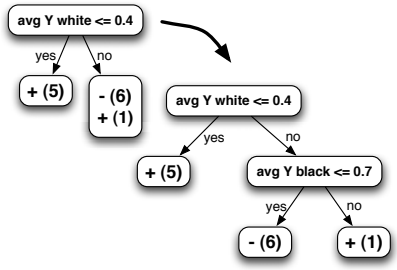


Figure 1.4: A decision tree. The numbers in the leaves show the number of instances in that leaf.

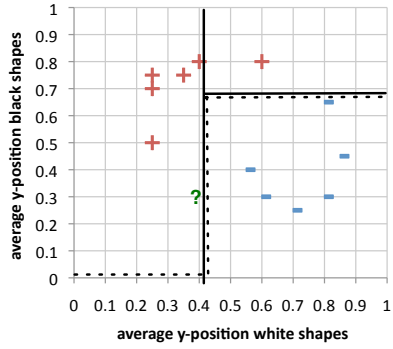


Figure 1.5: The data splits (full line) and decision boundary (dashed line) implied by the decision tree.

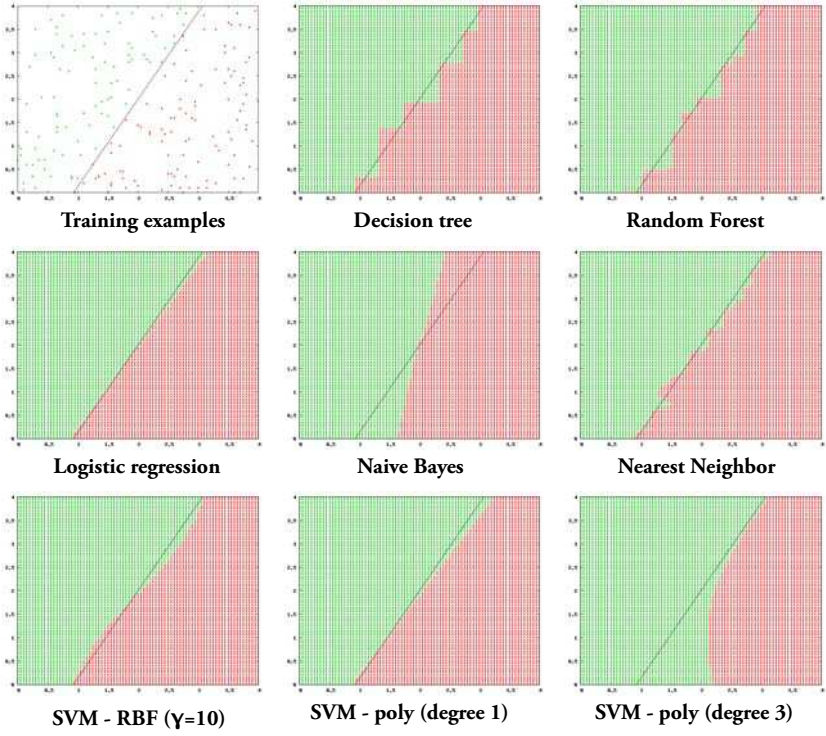


Figure 1.6: A classification problem with a linear pattern as target concept and 200 training examples, and the predictions of a range of learning algorithms over the entire instance space. Selected from Fawcett (2009).

1.3 Meta-learning

1.3.1 Model selection

Figure 1.6 clearly shows that, while each learning algorithm is able to approximate the target concept to some degree, clearly some algorithms are much better suited to model certain types of data: one algorithm may discover regularities completely missed by other algorithms. Furthermore, most algorithms have parameters that have a significant impact on the algorithms' ability to fit the data, as shown by the three Support Vector Machine (SVM) examples, so again a correct choice of parameters must be made. This problem is known as *model selection* or *algorithm selection*.

1.3.2 Learning bias

The reason why these learning algorithms perform differently on the same data is that they are all *biased* towards finding certain types of regularities. Indeed, the only information a learning algorithm has are the given examples. The *inductive leap* needed to predict the outcome future observations therefore requires at least some *assumptions* about how we can generalize from the given data points to future ones, and when these assumptions are wrong, the target concept will be approximated badly.
The set of assumptions used by a learning algorithm to choose one generalization over the other (other than strict consistency with the observed training instances) is called the *inductive bias* or *learning bias* (Mitchell 1980).
We can discern two different components of inductive bias, which are intimately tied to how learning algorithms seek patterns in the given data:

Language bias Every learning algorithm starts from a predefined *data model*, a structure assumed to be able to adequately capture regularities occurring in the data, e.g. a decision tree. It then *learns* by increasingly refining instances of this model, also called *hypotheses*, as it includes more observations. The space of all possible models is called the *hypothesis space* H. As such, from the viewpoint of the learning algorithm, we can rephrase our definition of supervised classification as the task of finding an hypothesis $h \in H$, such that h approximates the target function c over X. *Representational bias* is often used as a synonym.

Procedural bias While increasingly refining its hypothesis, one refinement will need to be selected over all possible refinements. The assumptions made about which refinement is generally better than others is called the computational or procedural bias, and usually consists of a set of *heuristics* designed to guide the search through the hypothesis space H, e.g. the information gain ratio in decision trees. As such, learning is in fact a search problem in which the space of all possible hypotheses is searched

for the one which fits the given data best. This bias may be as simple as a preference for the simplest hypothesis (Occam's razor), but is usually much more complex. In some cases, *background knowledge* may be available to help decide which hypothesis is more likely than another. For instance, when modeling chemical processes, we may use the fact that single bonds between atoms break much more easily than double bonds to choose between several hypothetical processes.

The necessity of bias for learning has already been described, in philosophical terms, by Hume in 1740:

> There can be no demonstrative arguments to prove, that those instances, of which we have had no experience, resemble those, of which we have had experience (...) We suppose, but are never able to prove, that there must be a resemblance betwixt those objects, of which we have had experience, and those which lie beyond the reach of our discovery (Hume 1740).

1.3.3 No free lunch

The *No Free Lunch* theorem for machine learning, also known as the *conservation law of generalization performance* (Schaffer 1994), even states that, if all possible data distributions are equally likely, any pair of learning algorithms will perform the same on average, or that "... for any algorithm, any elevated performance over one class of problems is exactly paid for in performance over another class" (Wolpert 2001). It effectively means we will never be able to build a single best, universal learning algorithm: one that, on average, performs better than all other algorithms over all imaginable data distributions. Of course, the premise that all possible data distributions are equally likely is a very strong one: it speaks of a universe in which everything is equally possible (try to learn in *that* situation), and some interpretations of the theorem state that it simply means that one cannot learn without a *prior*, without making some assumptions about the data distribution (Giraud-Carrier 2008), as we've discussed intuitively before.

1.3.4 Machine learning: An empirical science

Furthermore, because learning algorithms are heuristic in nature, it is very hard to theoretically *prove* that a new algorithm is better than others, even if only on a subset of problems. While there exists a certain amount of theory relating some heuristics to finding the hidden concept c, in many cases this relationship is very little understood. Moreover, unless artificially generated, the given data and its hidden concept are understood even less, otherwise we wouldn't need to model it in the first place.

As such, the performance of a certain algorithm will nearly always be evaluated *empirically*, meaning that machine learning is, to a large extent, an empirical science. Whether we want to test the performance of a new learning algorithm, or find out which of the many algorithms may be most suited for tackling a given data mining problem, we *implement* the algorithms and evaluate their performance on real-world datasets.

1.3.5 Meta-learning

The utility of any given learning algorithm thus ultimately depends on how well its learning bias matches the structure of the data at hand, and is measured in practice by running the algorithm on that data. *Meta-learning* is the science of discovering relationships in this meta-data: between measurable properties of the data and the algorithm bias on the one hand, and empirical performance data on the other hand. The (automatic) discovery of patterns in this meta-data helps us to better understand an algorithm's behavior on different types of data, and thus to design better algorithms, or to propose useful knowledge discovery processes for new problems.

1.3.6 The need for meta-learning

Meta-learning is particularly instrumental in answering two questions inherent in machine learning research. First, when designers introduce a new learning algorithm, how do they position it, performance-wise, in the landscape of existing algorithms? Conversely, when practitioners are faced with a new task, how do they know which algorithm to use? In either case, the only two viable alternatives in light of the No Free Lunch theorem are (Giraud-Carrier 2008):

Closed World Assumption Assume that all learning tasks likely to occur in real-world applications share a common underlying structure and form a *well-defined subset* of all imaginable tasks. In this case, find a collection of tasks representative for this subset to test algorithms. As a designer, *show* that a new algorithm is better than others on that collection. As a practitioner, pick any well-performing algorithm to tackle new problems.

Open World Assumption Don't assume an underlying structure in real-world data. As a designer, characterize as precisely as possible on *which* problems a novel algorithm outperforms others. As a practitioner, find a way to determine which algorithm(s) will perform well on the given problem.

1.3.6.1 Designing learning algorithms

The most widely-used approach in machine-learning consists in benchmarking algorithms against well-known repositories such as the UCI repository (Asuncion and Newman 2007), thus implicitly favoring the closed world assumption.

To date, however, no one has offered any characterization of *real-world* learning tasks, making the assumption that the problems in dataset repositories are representative for all real-world problems a conjecture at best. Moreover, quite a few studies only publish results on a small subset of benchmark problems, without defending why exactly that subset was chosen (Keogh and Kasetty 2003; LaLoudouana and Tarare 2003). Clearly, in this situation it would be better to follow the open world assumption and investigate on *which* problems a novel algorithm outperforms others, and for which types of data it needs further improvements. Currently however, such meta-analysis is extremely laborious and occurs only rarely in the literature. In this thesis, we shall address this problem by automatically gathering the necessary meta-data and facilitating its analysis, thus stimulating a much deeper understanding of empirical results.

1.3.6.2 Applying learning algorithms

Machine learning research has led to a very rich tapestry or learning algorithms, each proposing a radically new learning approach or tackling well-known limitations of prior algorithms. Unfortunately, because of limited meta-analysis, there is comparatively little insight gained in their individual applicability, and it is far from trivial to predict how well previous analysis will generalize beyond the problems tested against so far.

Any practical application of learning algorithms on new problems thus requires an *exploratory data analysis* (EDA) of various approaches. It is a costly trial-and-error process in which many algorithms, with many parameter settings, are run on many different transformations of the same data. It usually involves an expert guiding this process, based on personal experience with the algorithms involved. As discussed in Van Someren (2001), experts tend to prefer one or a few algorithms they know, and transform the data so it fits the algorithm better, possibly obfuscating important regularities in the process.[3] Giraud-Carrier (2008) notes that it is ironic that many machine learning experts try to convince business professionals to *listen* to what their data tell them and to extract the information embedded in their business data, while applied machine learning is guided mostly by hunches, anecdotal evidence, and individual experience, instead of *listening* to its own meta-data: the performance evaluations of learning algorithms on previous data mining problems.

Again, automizing the collection of meta-data on prior algorithm applications and facilitating its analysis would be of great use. By reusing these observations to discover interesting patterns, we can make informed decisions about which learning approaches are likely to work best on new problems. Not only is such an approach much more scientific, it would make detailed expertise readily available to the entire community of researchers and practitioners.

[3] He calls this a Procrustean approach, after the Greek character Procrustes who welcomed travelers to sleep in his guest bed, but only after cutting the feet of guests who were too tall for this bed, and stretching guests who were too short, which many did not survive.

1.4 Experiment Databases

The established approach in meta-learning is to gather as many algorithms as possible and run them on as many datasets as possible in order to build a large database of meta-examples to learn from. Indeed, this has been done on several occasions in the past, in projects such as StatLog (Michie et al. 1994) and METAL (METAL 2001) (see Chapter 3), which yielded many interesting insights. However, the field keeps evolving quickly, and new algorithms, preprocessing methods, learning tasks and evaluation procedures continue to emerge in the literature. Thus, it is impossible for a single study to cover this ever expanding space of learning approaches, and usually, the most recent tasks and algorithms - typically not yet covered in such studies - attract the most attention and would benefit most from meta-learning. Moreover, these studies were mostly aimed at practitioners looking for the best learning algorithms for a given problem, and much less at designers looking for insight into the performance of learning algorithms to steer the development of new techniques.

1.4.1 Collaborative Experimentation

Still, there is definitely no lack of recent learning algorithm meta-data. All around the globe, thousands of learning experiments are being executed on a daily basis, generating a constant stream of empirical information on learning techniques. Unfortunately, they are mostly interpreted with a single focus of interest and discarded afterwards. The information contained in these experiments, which might have many uses beyond their original intent, is therefore lost: any follow-up study or meta-learning analysis has to rerun all experiments, which is not only prohibitively expensive, but also practically cumbersome.
Other sciences are doing a much better job of curating and learning from empirical results, in particular e-Sciences: sciences that generate large volumes of experimental data, such as bio-informatics (e.g. gene sequences), astrophysics (e.g. streams of telescope images) and particle physics (e.g. particle collision data). They collect large amounts of experiments in public databases so they can be analyzed and reused by researchers all over the world and so that machine learning techniques can be used to automatically search for useful patterns in this data.
Like many of these sciences, data mining is also driven by large-scale experimentation: new algorithms are constantly evaluated on benchmark datasets, and for each new collection of data, extensive exploratory evaluations are executed to arrive at the best possible model. However, unlike these sciences, there exist no similar experimental repositories for data mining experiments, and data mining experiments themselves are currently not being documented and organized well enough to employ data mining techniques to search for insightful patterns in learning performance that would stimulate further machine learning research.

1.5 Goal of this thesis

This brings us to the goal of this thesis: to increase both the speed and vigor of machine learning research itself by offering tools to organize empirical machine learning information from all over the world, making it universally accessible and easy to analyze in depth. We also use these tools to corroborate existing knowledge in this area, and to gain a better understanding of the behavior of certain learning techniques.
We mean to achieve this through the design of *experiment databases*: online repositories designed to automatically collect the details of machine learning experiments by many different authors, and to intelligently organize them to enable fast and thorough analysis of the collected results. They offer a global platform for all kinds of meta-learning investigations, and enable large-scale, in-depth studies on state-of-the-art techniques that are very hard, if not impossible to perform when starting from scratch.
As such, we hope to engender a much more dynamic, *collaborative* approach to experimentation in machine learning, in which experiments can be freely shared, linked together and supplemented with meta-data on the used datasets and learning algorithms. Similar to developments in e-Sciences, we aim to create an "open scientific culture where as much information as possible is moved out of people's heads and labs, onto the network and into tools that can help us structure and filter the information" (Nielsen 2008).
Experiment databases also directly address some long-standing issues in machine learning research:

- Many questions about data mining algorithms (e.g. the effect of a parameter on a particular performance criterion) can be answered *on the fly* - in a single query to the database - using the combined results of many prior studies. Currently, this requires the manual setup of new experiments, a very time consuming and laborious procedure.
- Reuse of prior results will enable much larger empirical studies, yielding more generally valid insights. Indeed, testing learning algorithms under many circumstances (e.g. many different datasets and parameter settings) is so expensive that, in the current status quo, most studies limit themselves to a relatively small selection of datasets and parameter variations, making it hard to interpret how generalizable the findings are. It has been argued that this creates a false sense of progress (Hand 2006).
- Experiment repositories also serve as a venue to store all the details that make the experiments reproducible, which is crucial to verify prior results and to build on them. Unfortunately, data mining experiments are currently not being documented with the same rigor found in noncomputing experimental sciences. In fact, reproducibility is listed as one of the foremost concerns in a recent panel of the SIAM International Conference on Data Mining (Hirsh 2008).

1.6 Roadmap and Contributions

The thesis is organized into three main parts, and a fourth part summarizing our findings. The *Prelude* covers the practice and state-of-the-art in meta-learning research and its applications, *Part I* explicates the principled design of experiment databases for machine learning, and *Part II* uses a pilot implementation of such an experiment database to perform in-depth meta-learning studies, illustrating how easily this can be done. The *Finale* summarizes our findings and highlights avenues for future work. With each subtopic discussed below, we also mention the most relevant papers published in the course of this work.

1.6.1 Prelude

First, in **Chapter 2**, we provide a survey of the practice and state-of-the-art in the field of meta-learning. Most importantly, we define a framework for meta-learning investigations that extends previous frameworks in two important directions: the internal mechanisms of learning algorithms and the preprocessing of data. We also cast the literature in this framework, yielding a very complete overview of meta-learning research, and make two important observations. First, that new techniques are needed to efficiently share and exploit meta-data, supporting larger meta-learning studies. Second, that very few studies are aimed at designers of learning algorithms. We propose to address these issues by building a *community-based* meta-learning platform that also supports *descriptive* meta-learning studies aimed at understanding algorithm behavior rather than predicting it.
Our contributions here thus include a very complete but concise overview of meta-learning literature and the identification of ways in which to drastically improve further research. We also enlist and explain all the theoretical meta-data, such as measurable dataset and algorithm properties, that we store in our experiment database to support advanced meta-learning studies.
Next, **Chapter 3** provides a survey of the various architectures that have been developed, or simply proposed, to build KD support systems. Our main observation is that most of these systems are seemingly developed independently from each other, without really capitalizing on the benefits of prior systems. We therefore propose a new, open KD support architecture that combines the best aspects of earlier systems, and that solves many of their limitations. Our main contribution here is a detailed analysis of research in building KD support systems, the extraction of the most valuable ideas and a proposal for future, community-based KD support systems.

- Vanschoren, J. (2010) Meta-learning architectures: Collecting, organizing and exploiting meta-knowledge. In *Meta-learning*, (N. Jankowski, W. Duch, K. Grabczewski, ed.), Springer. (In press)

1.6.2 Part I

In Part I, we start out by motivating the creation of experiment databases in **Chapter 4**, specifically outlining the need for greater reproducibility, generalizability and interpretability in machine learning research. To learn from previous practical applications of experiment repositories, we look to so-called e-Sciences, where they have been applied successfully for some time now. Most importantly, we will use what we have learned from these sciences to build a conceptual framework for collaborative experimentation in machine learning, detailing the required building blocks that will be developed in the subsequent chapters. Finally, we show how experiment databases can be used to engender improved experimental methodologies in machine learning research.
Our main contributions here are the architecture and motivation for experiment databases and collaborative experimentation in machine learning.

- Vanschoren, J., Blockeel, H., Pfahringer, B. and Holmes, G. Experiment Databases. A new way to share, organize and learn from experiments. Machine Learning Journal. (Submitted)
- Vanschoren, J. and Blockeel, H. (2010). Experiment Databases. In *Inductive Databases and Constraint-Based Data Mining*, (S. Dzeroski, B. Goethals, P. Panov, ed.), Springer. (In press)
- Blockeel, H.[†] and Vanschoren, J.[†] (2007). Experiment databases: Towards an improved experimental methodology in machine learning. Lecture notes in computer science, 4702, 6-17. ECML-2007

From these e-Sciences we learn the importance of unambiguous, standardized experiment descriptions as well as the necessity for experiment repositories to extend easily to new developments. To allow researchers to extend such repositories to support new, specific aspects of their research, a formal, conceptual domain model must be designed that covers all important aspects of machine learning experimentation, but that also serves as a basis to propose further extensions in a collaborative fashion.
In the following chapters, we therefore first define a formal domain model, an *ontology*, for experimentation in machine learning, after which we use this ontology to define an XML-based language to exchange experiments, as well as a database model to organize all submitted results.
Chapter 5 introduces our ontology, called **Exposé**, providing a structured core vocabulary for describing machine learning experiments. The main contribution here is the ontology itself, which, even though the current version is still largely focussed on supervised classification and propositional datasets, provides a well-founded and formalized domain model of machine learning experimentation. It covers various types of experiments, the experimental context (including experimental designs), learning algorithm evaluation techniques and

[†]First authors, ordered alphabetically

evaluation measures, datasets and their properties and structure, and finally algorithms and their parameters, components, implementations, internal learning strategies and measurable properties. It also serves as a foundation to extend the work in this thesis to many other subfields of machine learning.

In **Chapter 6**, we cast the ontological vocabulary developed in the previous chapter into a formal, XML-based language, dubbed **ExpML**. We also describe how the ontological concepts and their relationships are translated to XML elements and syntax, so that ontological extensions can be quickly translated to updated ExpML definitions.

- Vanschoren, J., Blockeel, H., Pfahringer, B., Holmes, G. (2008). Organizing the worlds machine learning information. Communications in Computer and Information Science, 17, 693-708. ISOLA-2008

In **Chapter 7**, we investigate how machine learning experiments can be intelligently organized in searchable experiment databases. Starting from the Exposé ontology, we explain how we derived a detailed database model for classification experiments that allows to write queries about the many aspects of learning behavior that are covered by the ontology. Our contributions here are twofold. First, we provide database models for setting up local or global experiment databases. Second, our pilot experiment database implementation, which can be queried online, is also a valuable contribution to the community, containing over 650,000 experiments on many classification algorithms and many datasets. It is frequently being queried online[4], and extensions to the popular WEKA and KNIME toolboxes[5] are currently being developed to send results directly from those toolboxes to the experiment database, thus allowing thousands of users to submit new experiments at the click of a button.

- Vanschoren, J., Blockeel, H. A platform for collaborative experimentation in machine learning. Journal of Machine Learning Research (Submitted)
- Vanschoren, J. and Blockeel, H. (2009). A community-based platform for machine learning experimentation. Lecture Notes In Computer Science, 5782, 750-754. ECML-2009 **(Best demo award)**
- Vanschoren, J., Blockeel, H. (2008). Investigating classifier learning behavior with experiment databases. Data Analysis, Machine Learning and Applications - Annual Conference of the Gesellschaft für Klassifikation, 421-428. GfKL-2007
- Vanschoren, J. (2008). Experiment databases for machine learning. NIPS Workshop on Machine Learning Open Source Software at NIPS 2008.

[4] At the time of writing, usage statistics suggest that over 400 visitors have visited the website more than 100 times, and over 1000 results were exported to csv files for further analysis.

[5] See `http://www.cs.waikato.ac.nz/ml/weka/` and `http://www.knime.org/`

1.6.3 Part II

With a large experiment database now available, we will now use it to perform in-depth meta-learning studies. First, to hide the inherent complexities of experiment databases from researchers wishing to use them, **Chapter 8** presents two user interfaces: a software interface that can be used by data mining tools to programmatically submit experiments, as well as intuitive query interface that allows users to easily compose database queries and visualize the returned results. Our main contribution here is making it easy for researchers to submit and query for experiments.

Finally, in **Chapter 9**, we use these query tools and our pilot experiment database to perform in-depth meta-learning studies, resulting in many interesting findings. We summarize these findings in three types of studies, increasingly making use of the available meta-level descriptions, and offering increasingly generalizable results.

Model-level analysis. We first perform a very general ranking of learning algorithms and verify some earlier studies. We also make interesting observations concerning ensemble methods: the benefit of bagging over parameter optimization varies with the evaluation metric used, bagging primarily reduces variance error while boosting primarily reduces bias error, and boosting even seems to be useless for certain algorithms. Finally, we show that statistical significance tests can be included in queries to build algorithm rankings on the fly.

Data-level analysis. First, we show how the number of trees in a random forest interacts with the size of the dataset, and found that there exist situations where performance actually decreases as more trees are added. We also find that some SVM implementations tend to overfit data with large numbers of attributes and suggest algorithm improvements, thus illustrating how experiment databases can help algorithm design. By including data preprocessing steps, we also draw learning curves for several algorithms and show that they can cross. Finally, we build a surprisingly simple meta-decision tree modeling the circumstances under which one algorithm outperforms another and explain the result in terms of their internal learning mechanisms.

Method-level analysis. Using large amounts of bias-variance decomposition results, we create bias-variance profiles for all involved algorithms, and provide further evidence that low-bias algorithms are especially useful on large datasets.

- Vanschoren, J., Pfahringer, B., Holmes, G. (2008). Learning from the past with experiment databases. Lecture Notes in Artificial Intelligence, 5351, 485-496. PRICAI-2008
- Vanschoren, J., Blockeel, H., Pfahringer, B., Holmes, G. (2008). Experiment databases: Creating a new platform for meta-learning research. Joint Planning to Learn Workshop at ICML/UAI/COLT 2008, 10-15.
- Vanschoren, J., Van Assche, A., Vens, C., Blockeel, H. (2007). Meta-learning from experiment databases: An illustration. Annual Machine Learning Conference of Belgium and The Netherlands, 120-127.

Prelude

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 Two

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 2.2. After this, we focus on the setting of algorithm selection and outline a general framework in Section 2.3, 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 2.10.

2.1 Introduction

2.1.1 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 discussed in Chapter 1. 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, as discussed in Chapter 1. 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.

2.2 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 2.2.1, 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 2.2.2, 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 2.2.3, 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).

2.2.1 Ensemble learning

2.2.1.1 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*.

2.2.1.2 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.

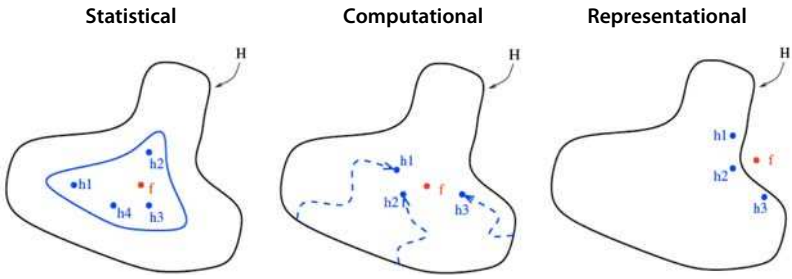


Figure 2.1: Depiction of three reasons why an ensemble learner could perform better than an individual base-learner. Adapted from Dietterich (2000).

For now, we only consider the first type of ensemble learning. To understand *why* such ensembles improve learning performance, take a look at Figure 2.1, 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 2.1.

Representational The third reason is depicted on the right of Figure 2.1 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.

2.2.1.3 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. In Chapter 9, we investigate whether the same base-learners can be used for different ensemble techniques, or whether it is better to use radically different ones.

The second type of ensemble learning hopes to profit mostly from the representational aspect of Section 2.2.1.2. 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:

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

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 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.

2.2.2 Transfer learning

2.2.2.1 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).

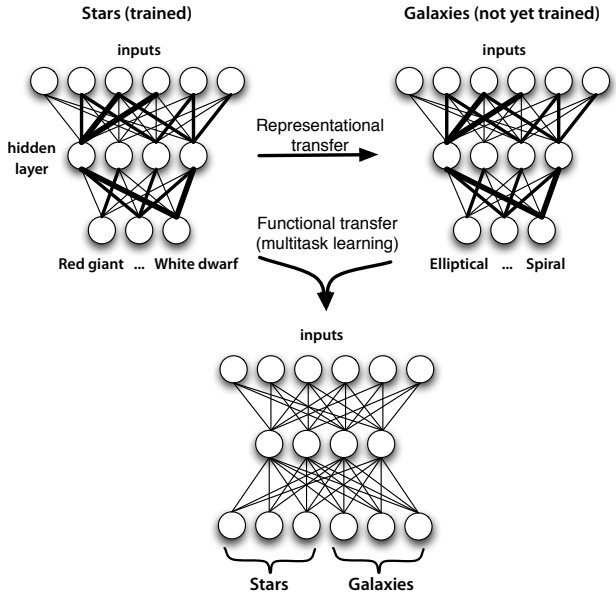


Figure 2.2: Two different types of transfer learning in neural networks. Adapted from Brazdil et al. (2009).

2.2.2.2 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 2.2. 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 tar-

get network) (Thrun and Mitchell 1995; Baxter 1996). This is shown by the first arrow in Figure 2.2: 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 2.2. 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.

2.2.2.3 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).

2.2.3 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.

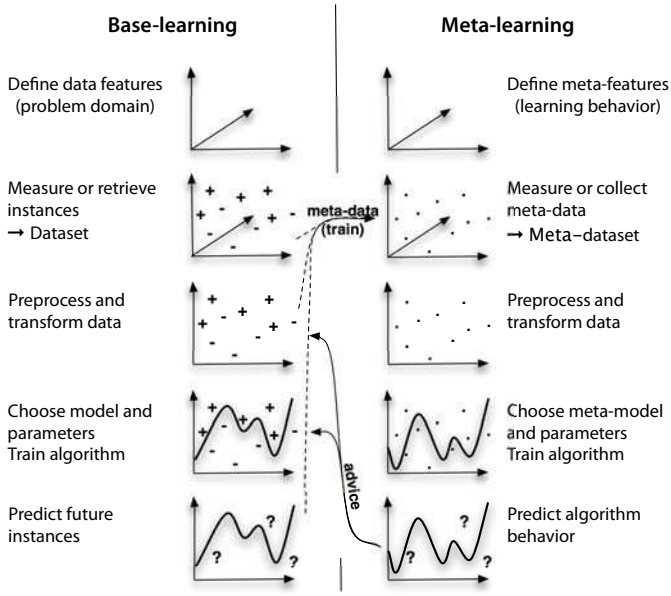


Figure 2.3: A comparison of base-learning and meta-learning.

2.2.3.1 Base-learning versus meta-learning

The interplay between base-learning and meta-learning can be described as shown in Figure 2.3. They both consist of the typical knowledge discovery steps discussed in Section 1.2.1. 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.

2.2.3.2 The No Free Lunch Theorem revisited

There is one implication of the No Free Lunch theorem we haven't yet considered: if a meta-learning algorithm can predict which algorithm will be suitable for any given problem, in itself being some kind of *ultimate* learning algorithm, doesn't this violate the No Free Lunch theorem?
Meta-learning departs from the No Free Lunch theorem in that it takes into account prior observations of real-life learning problems. As pointed out by Hartley (1994) and Giraud-Carrier and Provost (2005), we are not interested in a *universal* learner, a learning algorithm that is applicable independent of any assumptions, as is the case in the No Free Lunch theorem, only in one that is applicable on real-world problems, or maybe even a subset of those.
As such, there is no conflict with the No Free Lunch theorem: for all intents and purposes, it *is* possible to build a meta-learning system capable of performing better than single base-learning algorithms, because the meta-learner *learns* to specialize on the real-world meta-examples it encounters, while the base-learner's bias is fixed. On the other hand, if a new problem shows up that belongs to a radically different distribution than that of the meta-examples it was trained on, then, like any learning algorithm, it will not be able to produce good predictions. Just like we adapt our actions to perform better in the context of the world around us, meta-learning tries to adapt its learning approach based on (a specific subset of) real-world learning problems.

2.3 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.

2.3.1 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 2.4, 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:

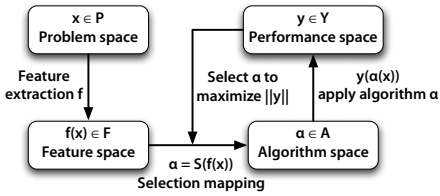


Figure 2.4: Rice's framework for algorithm selection. Adapted from Smith-Miles (2008a).

> 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.

2.3.2 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 2.5 (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 2.5). 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).

2.3.3 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 2.5. 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

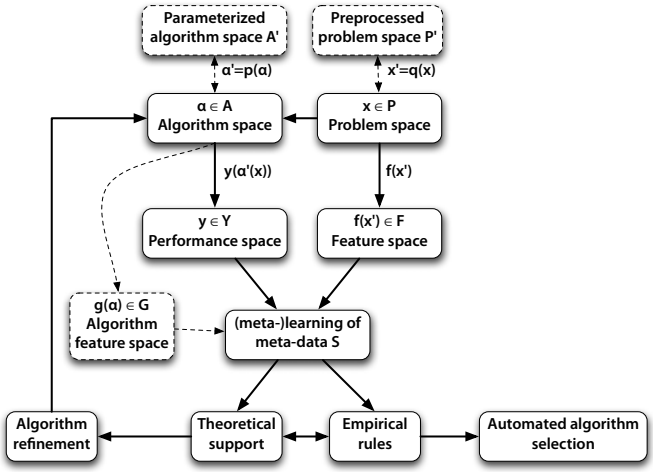


Figure 2.5: Proposed framework for meta-learning in Smith-Miles (2008a) (full lines), and our extensions (dashed lines) of this framework.

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.

2.4 The data meta-feature space F

As lllustrated in Rice's framework and in Figure 2.3, 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.

2.4.1 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.

2.4.2 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

2.4.3 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).

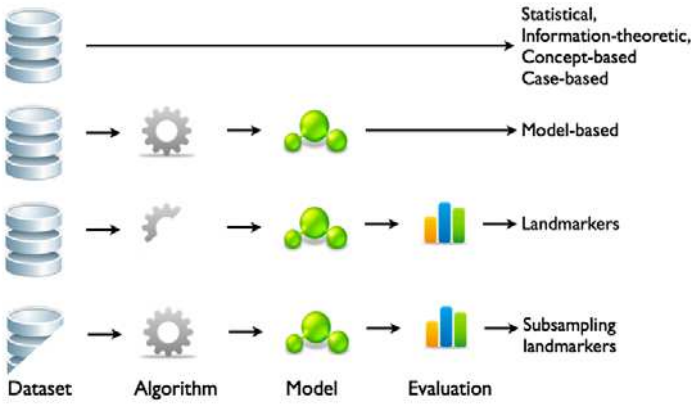


Figure 2.6: Different data characterization approaches. Adapted from Brazdil et al. (2009).

2.4.4 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 2.6, 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 2.7. A complex tree generally means that the hidden concept is also quite complex, although as we've seen in Section 1.2.3 this is not always the case. 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.

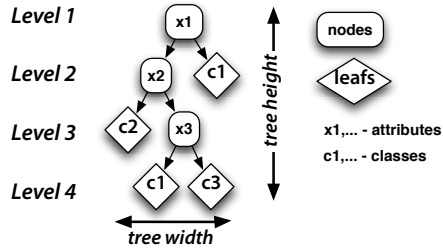


Figure 2.7: Structure of a decision tree. Adapted from Peng et al. (2002).

- $max_{branch}, min_{branch}, \mu_{branch}, \sigma_{branch}$ 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 standard 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.

2.4.5 Landmarking

2.4.5.1 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]. As we've discussed in Section 1.3.2, some learning

[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.

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.
- *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 2.9.1). *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).

2.4.5.2 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

[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

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

2.4.6 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.

2.4.7 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 2.6).

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.

2.5 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.

2.5.1 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. We will come back to this in Chapter 5.
- *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 cumulativity, 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.

2.5.2 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, and will also build some of these profiles in Chapter 9.

2.5.2.1 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.
- The (squared) *intrinsic error* is associated with the inherent uncertainty in the data.

Dietterich and Kong (1995) investigate the relationship between learning bias (see Section 1.3.2) 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 2.8. 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

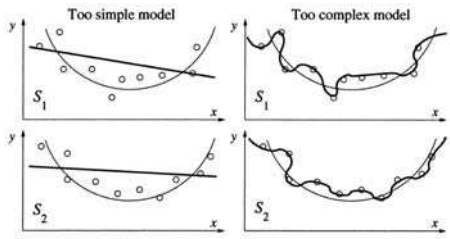


Figure 2.8: Bias versus variance error (Geurts et al. 2005).

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. We will investigate this algorithm property in depth in Section 9.3.

2.5.2.2 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.

2.5.2.3 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.

2.6 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.

2.6.1 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).

2.6.2 The preprocessed problem space P'

As discussed in Section 1.2.1, 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 2.5.2.
We should therefore not simply predict the correct algorithm: we should offer complete *workflows* of processes. Various approaches to do this will be discussed in Chapter 3.
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).

2.7 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

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

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 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.

2.8 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.

2.8.1 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. We will also discuss them in more depth in Section 5.3.4

2.8.2 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 (also see Section 3.3.2.4). 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).

2.9 The meta-learner S

Having collected all the necessary meta-data, the final step in the meta-learning process, illustrated in Figure 2.3, is to choose a meta-learner, train it on the meta-data and predict which algorithms are best suited for a given problem.

2.9.1 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.

2.9.2 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.

2.9.2.1 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 2.5.2) and histogram representations of dataset attribute properties (see Section A.7).

2.9.2.2 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.

2.9.2.3 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.

2.9.2.4 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)

[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.

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 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.

2.10 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 2.1, 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

Table 2.1: 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)

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 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 (both are discussed in Section 3.3.2). 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 4 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 2.4 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.
As discussed in Chapter 1, 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. We show many examples of their capabilities in Chapter 9.

> Human beings, who are almost unique in having the ability to learn from the experience of others, are also remarkable for their apparent disinclination to do so.
>
> *Douglas Adams*

Chapter Three

Intelligent Knowledge Discovery Support

3.1 Introduction

While a valid intellectual challenge in its own right, meta-learning finds its real *raison d'être* in the practical support it offers Data Mining practitioners (Giraud-Carrier 2008). Indeed, the whole point of understanding how to learn in any given situation is to go out in the real world and learn as much as possible, from any source of data we encounter! However, almost any type of raw data will initially be very hard to learn from, and about 80% of the effort in discovering useful patterns lies in the clever preprocessing of data (Morik and Scholz 2004). Thus, for machine learning to become a tool we can instantly apply in any given situation, or at least to get proper guidance when applying it, we need to build extended meta-learning systems that encompass the entire knowledge discovery process, from raw data to finished models, and that keep learning, keep accumulating meta-knowledge, every time they are presented with new problems.

The algorithm selection problem is thus widened into a *workflow creation* problem, in which an entire stream of different processes needs to be proposed to the end user. This entails that our collection of meta-knowledge must also be extended to characterize all those different processes.

In this chapter, we provide a survey of the various architectures that have been developed, or simply proposed, to build such extended meta-learning systems. They all consist of integrated repositories of meta-knowledge on the KD process and leverage that information to propose useful workflows. Our main observation is that most of these systems are very different, and were seemingly developed independently from each other, without really capitalizing on the benefits of prior systems. By bringing these different architectures together and highlighting their strengths and weaknesses, we aim to reuse what we have learned, and we draw a roadmap towards a new generation of KD support systems.

While this survey covers related work in the area of organizing and using meta-data on learning processes, and various lessons will be learned, it can be safely skipped without loss of continuity if one wants to proceed to the main parts of this thesis.

Despite their differences, we can classify the KD systems in this chapter in the following groups, based on the way they leverage the obtained meta-data:

Expert systems In expert systems, experts are asked to express their reasoning when tackling a certain problem. This knowledge is then converted into a set of explicit rules, to be automatically triggered to guide future problems.

Meta-models The goal here is to automatically predict the usefulness of workflows based on prior experience. They contain a meta-model that is updated as more experience becomes available. It is the logical extension of meta-learning to the KD process.

Case-based reasoning In general terms, case-based reasoning (CBR) is the process of solving new problems based on the solutions of similar past problems. This is very similar to the way most humans with some KD experience would tackle the problem: remember similar prior cases and adapt what you did then to the new situation. To mimic this approach, a knowledge base is populated by previously successful workflows, annotated with meta-data. When a new problem presents itself, the system will retrieve the most similar cases, which can then be altered by the user to better fit her needs.

Planning All possible actions in a KD workflow are described as operations with preconditions and effects, and an AI planner is used to find the most interesting plans (workflows).

Querying In this case, meta-data is gathered and organized in such a way that users can ask any kind of question about the utility or general behavior of KD processes in a predefined query language, which will then be answered by the system based on the available meta-data. They open up the available meta-data to help users make informed decisions.

Orthogonal to this distinction, we can also characterize the various systems by the *type of meta-knowledge* they store, although in some cases, a combination

of these sources is employed.

Expert knowledge Rules, models, heuristics or entire KD workflows are entered beforehand by experts, based on their own experience with certain learning approaches.

Experiments Here, the meta-data is purely empirical. They provide objective assessments of the performance of workflows or individual processes on certain problems.

Workflows Workflows are descriptions of the entire KD process, involving linear or graph-like sequences of all the employed preprocessing, transformation, modeling and postprocessing steps. They are often annotated with simple properties or qualitative assessments by users.

Ontologies An *ontology* is a formal representation of a set of concepts within a domain and the relationships between those concepts. They provide a fixed vocabulary of data mining concepts, such as algorithms and their components, and describe how they relate to each other, e.g. what the role is of a specific component in an algorithm. They can be used to create unambiguous descriptions of meta-knowledge which can then be interpreted by many different systems to reason about the stored information. A more in-depth discussion is provided in Chapter 5.

While we cover a wide range of approaches, the landscape of all meta-learning and KD solutions is quite extensive. However, most of these simply facilitate access to KD techniques, sometimes offering wizard-like interfaces with some expert advice, but they essentially leave the intelligent selection of techniques as an exercise to the user. Here, we focus on those systems that introduce new ways of leveraging meta-knowledge to offer intelligent KD support. We also skip KD systems that feature some type of knowledge base to monitor the current workflow composition, but that do not use that knowledge to advise on future problems, such as the INLEN system (Michalski et al. 1992; Kaufman 1997; Kaufman and Michalski 1998). This overview partially overlaps with previous, more concise overviews of meta-learning systems for KD (Brazdil et al. 2009; Giraud-Carrier 2008). Here, however, we provide a more in-depth discussion of their architectures, focussing on those aspects that can be reused to design future KD support systems.

In the remainder of this chapter, we will split our discussion in two: past and future. The past consists of all previously proposed solutions, even though some of these are still in active development and may be extended further. This will cover the following five sections, each corresponding to one of the five approaches outlined above. For each system, we consecutively discuss its architecture, the employed meta-knowledge, any meta-learning that is involved and finally its benefits and drawbacks. Finally, in Section 3.7, we provide a short summary of all approaches, before looking towards the future: we outline a platform for the development of future KD support systems aimed at bringing the best aspects of prior systems together. The meta-learning platform

developed in this thesis will play a central role in this platform, as well as a number of other so-called *third generation KD tools*, often still in their design phase, which are especially geared towards a *community-based approach* for gathering the necessary meta-knowledge and executing KD workflows.

3.2 Expert systems

3.2.1 Consultant-2

One of the first inroads into systematically gathering and using meta-knowledge about machine learning algorithms was Consultant-2 (Craw et al. 1992; Sleeman et al. 1995): an expert system developed to provide end-user guidance for the MLT machine learning toolbox (MLT 1993; Kodratoff et al. 1992). Although the system did not learn by itself from previous algorithm runs, it did identify a number of important meta-features of the data and the produced models, and used these to express rules about the applicability of algorithms.

3.2.1.1 Architecture

A high-level overview of the architecture of Consultant-2 is shown in Figure 3.1. It was centered around a knowledge base that stored about 250 rules, hand-crafted by machine learning experts. The system interacted with the user through question-answer sessions in which the user was asked to provide information about the given data (e.g. the number of classes or whether it could be expected to be noisy) and the desired output (e.g. rules or a decision tree). Based on that information, the system then used the stored rules to calculate a score for each algorithm. The user could also go back on previous answers to see how that would have influenced the ranking. When the user selected an algorithm, the system would automatically run it, after which it would engage in a new question-answer session to assess whether the user was satisfied with the results. If not, the system would generate a list with possible parameter recommendations, again scored according to the stored heuristic rules.

3.2.1.2 Meta-knowledge

The rules were always of the form $if(A_n)\ then\ B$, in which A_n was a conjunction of properties of either the *task description* or the *produced models*, and B expressed some kind of action the system could perform. First, the task description included qualitative measures entered by the user, such as the task type, any available background knowledge and which output models were preferable, as well as quantitative measures such as the number of classes and the amount of noise. Second, the produced models where characterized by the amount of time and memory needed to build them and model-specific features such as the average path length and number of leaf nodes in decision trees, the

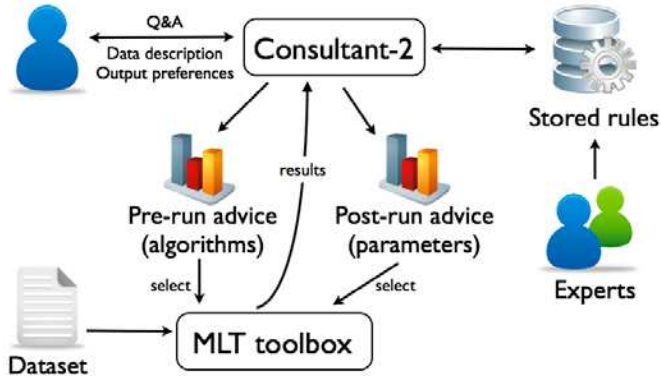


Figure 3.1: The architecture of Consultant-2. Derived from Craw et al. (1992)

number and average center distance of clusters and the number and significance of rules.[1] The resulting actions B could either adjust the scores for specific algorithms, propose ranges for certain parameter values, or transform the data (e.g. discretization) so as to suit the selected algorithm. Illustrations of these rules can be found in Sleeman et al. (1995).

3.2.1.3 Meta-learning

There is no real meta-learning process in Consultant-2, it just applied its predefined 'model' of the KD process. This process was divided into a number of smaller steps (selecting an algorithm, transforming the data, selecting parameters,...), each associated with a number of rules. It then cycles through that process, asking questions, executing the corresponding rules, and triggering the corresponding actions, until the user is satisfied with the result.

3.2.1.4 Discussion

The expert advice in Consultant-2 has proven to be successful in a number of applications, and a new version has been proposed to also provide guidance for data preprocessing. Still, to the best of our knowledge, Consultant-3 has never been implemented. An obvious drawback of this approach is the fact that the heuristic rules are hand-crafted. This means that for every new algorithm, new rules have to be defined that differentiate it from all the existing algorithms. One must also keep in mind that MLT only had a limited set of 10 algorithms covering a wide range of tasks (e.g. classification, regression and clustering), making it quite feasible to select the correct algorithm based on the syntactic

[1] These were used mostly to advise on the algorithm's parameters.

properties of the input data and the preferred output model. With the tens or hundreds of methods available today on classification alone, it might not be so straightforward to make a similar differentiation. Still, the idea of a database of meta-rules is valuable, though instead of manually defining them, they should be learned and refined automatically based on past algorithm runs. Such a system would also automatically adapt as new algorithms are added.

3.3 Meta-models

3.3.1 STABB and VBMS

The groundwork for many meta-learning systems were laid by two early precursors. STABB (Utgoff 1986) was a system that basically tried to automatically match a learner's bias to the given data. It used an algorithm called LEX with a specific grammar. When an algorithm could not completely match the hidden concept, its bias (the grammar) or the structure of the dataset was changed until it could. VBMS (Rendell et al. 1987) was a very simple meta-learning system that tried to select the best of three learning algorithms using only two meta-features: the number of training instances and the number of features.

3.3.2 The Data Mining Advisor (DMA)

The Data Mining Advisor (DMA) (Giraud-Carrier 2005) is a web-based algorithm recommendation system that automatically generates rankings of classification algorithms according to user-specified objectives. It was developed as part of the METAL project (METAL 2001).

3.3.2.1 Foundations: StatLog

Much of the theoretical underpinning of the DMA was provided by the StatLog project (Michie et al. 1994), which was aimed to provide a large-scale, objective assessment of the strengths and weaknesses of the various classification approaches existing at the time. Its methodology is shown in Figure 3.2.

First, a wide range of datasets are characterized with a wide range of newly defined meta-features (see Section A). Next, the algorithms are evaluated on these datasets. For each dataset, all algorithms whose error rate fell within a certain (algorithm-dependent) margin of that of the best algorithms were labeled *applicable*, while the others were labeled *non-applicable*. Finally, decision trees were built for each algorithm, predicting when it can be expected to be useful on a new dataset. The resulting rules, forming a rule base for algorithm selection, can be found in Michie et al. (1994).

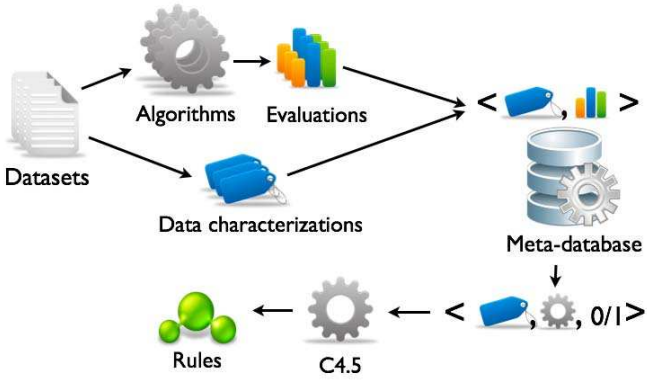


Figure 3.2: The StatLog approach.

3.3.2.2 Architecture

The architecture of the DMA is shown in Figure 3.3. First, the tool is trained by generating the necessary meta-data, just as in StatLog. However, instead of predicting whether an algorithm was applicable or not, it *ranked* all of them. New datasets can be uploaded via a web-interface, its meta-features automatically computed (below the dotted line in Figure 3.3). Bearing privacy issues in mind, the user can choose to keep the dataset, the data characterizations or both hidden from other users of the system. Next, the DMA will use the stored meta-data to predict the expected ranking of all algorithms on the new dataset. Finally, as a convenience to the user, one can also run a number of algorithms on the new dataset, after which the system returns their true ranking and performance. Figure 3.4 shows an example of the rankings generated by DMA. The meaning of the 'Predicted Score' is explained in Section 3.3.2.4.

3.3.2.3 Meta-knowledge

While the METAL project discovered various new kinds of meta-features (see Section 2.4), these were not used in the DMA tool. In fact, only a set of 7 numerical StatLog-style characteristics was selected[2] for predicting the relative performance of the algorithms. This is most likely due to the employed meta-learner, which is very sensitive to the curse of dimensionality.
At its inception, the DMA tool was initialized with 67 datasets, mostly from the UCI repository. Since then, an additional 83 tasks were uploaded by users (Giraud-Carrier 2005). Furthermore, it operates on a set of 10 classification algorithms, shown in Figure 3.4, but only with default parameter settings, and evaluates them based on their predictive accuracy and speed.

[2]Some of them were slightly modified. For instance, instead of using the absolute number of symbolic attributes, DMA uses the *ratio* of symbolic attributes.

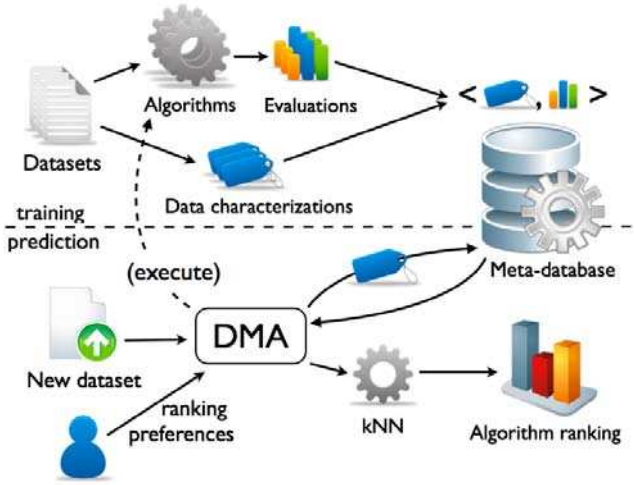


Figure 3.3: The architecture of DMA. Adapted from Brazdil et al. (2009).

Ranking table

Dowload results | register now! | learn more about the online advisor

Predicted Rank	Algorithm	Predicted Score	Status	Run	Accuracy	Time	True Rank	True Score
1.	c50rules	1.031	finished	--	0.2830000	?	6	1.003
2.	lindiscr	1.03	finished	--	0.2340000	?	1	1.048
3.	c50tree	1.026	--		--	--	--	--
4.	ltree	1.023	--		--	--	--	--
5.	clemMLP	1.017	finished	--	0.2680000	?	5	1.006
6.	c50boost	1.017	--		--	--	--	--
7.	ripper	1.009	--		--	--	--	--
8.	mlcnb	1	finished	--	0.2430000	?	2	1.036
9.	clemRBFN	0.948	--		--	--	--	--
10.	mlcib1	0.913	finished	--	0.3180000	?	10	0.938

(a) Emphasis on Accuracy

Ranking table

Dowload results | register now! | learn more about the online advisor

Predicted Rank	Algorithm	Predicted Score	Status	Run	Accuracy	Time	True Rank	True Score
1.	lindiscr	1.153	finished	--	0.2340000	?	1	1.154
2.	c50tree	1.144	finished	--	0.2940000	?	2	1.101
3.	c50rules	1.07	finished	--	0.2830000	?	4	1.058
4.	ltree	1.061	--		--	--	--	--
5.	mlcnb	1.051	--		--	--	--	--
6.	c50boost	1.009	--		--	--	--	--
7.	ripper	1.002	--		--	--	--	--
8.	clemMLP	0.909	--		--	--	--	--
9.	mlcib1	0.903	finished	--	0.3180000	?	8	0.931
10.	clemRBFN	0.842	--		--	--	--	--

(b) Emphasis on Training Time

Figure 3.4: An illustration of the output of the DMA. The first ranking favors accurate algorithms, the second ranking favors fast ones. Taken from Giraud-Carrier (2005).

3.3.2.4 Meta-learning

DMA uses a k-nearest neighbors (kNN) approach as its meta-learner, with k=3 (Brazdil et al. 2003). This is motivated by the desire to continuously add new meta-data without having to rebuild the meta-model every time.
A multi-criteria evaluation measure is defined (the *adjusted ratio of ratios*) which allows the user to add emphasis either on the predictive accuracy or the training time. This is done by defining the weight $AccD$ of the training time ratio. This allows the user to trade $AccD\%$ accuracy for a 10-time increase/decrease in training time. The DMA interface offers 3 options: $AccD$=0.1 (focus on accuracy), $AccD$=10 (focus on speed) or $AccD$=1 (equal importance). The user can also opt not to use this measure and use the *efficiency measure* of Data Envelopment Analysis (DEA) instead (Giraud-Carrier 2005). The system then calculates a kind of average over the 3 most similar datasets for each algorithm (the 'predicted score' in Figure 3.4) according to which the ranking is established.

3.3.2.5 Discussion

The DMA approach brings the benefits of meta-learning to a larger audience by automating most of the underlying steps and allowing the user to state some preferences. Moreover, it is able to continuously improve its predictions as it is given more problems (which are used to generate more meta-data). Unfortunately, it has a number of limitations that affect the practical usefulness of the returned rankings. First, it offers no advice about which preprocessing steps to perform or which parameter values might be useful, which both have a profound impact on the relative performance of learning algorithms. Furthermore, while the multi-criteria ranking is very useful, the system only records two basic evaluation metrics, which might be insufficient for some users. Finally, using kNN means no interpretable models are being built, making it a purely predictive system. Several alternatives to kNN have been proposed (Pfahringer et al. 2000; Bensusan and Giraud-Carrier 2000b) which may be useful in future incarnations of this type of system.

3.3.3 NOEMON

NOEMON (Kalousis and Hilario 2001b; Kalousis and Theoharis 1999), shown in Figure 3.5, follows the approach of Aha (1992) to compare algorithms two by two. Starting from a meta-database similar to the one used in DMA, but with histogram-representations of attribute-dependent features (see Section A.7), the performance results on every combination of two algorithms are extracted. Next, the performance values are replaced with a statistical significance test indicating when one algorithm significantly outperforms the other and the number of data meta-features is reduced using automatic feature selection. This

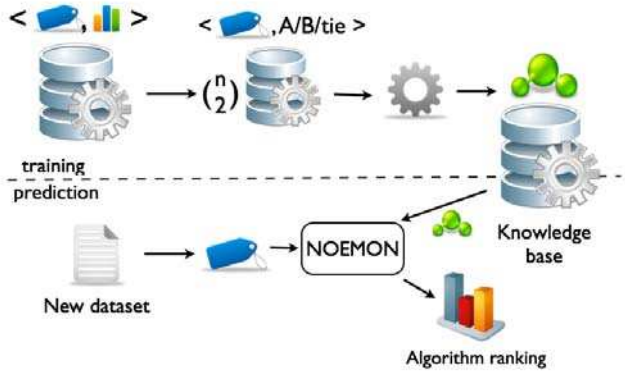


Figure 3.5: NOEMON's architecture. Based on Kalousis and Hilario (2001b).

data is then fed to a decision tree learner to build a model predicting when one algorithm will be superior or when they will tie on a new dataset. Finally, all such pairwise models are stored in a knowledge base.
At prediction-time, the system collects all models concerning a certain algorithm and counts the number of predicted wins/ties/losses against all other algorithms to produce the final score for each algorithm, which is then converted into a ranking.

3.4 Planning

The former two systems are still limited in that they only tackle the algorithm selection step. To generate advice on the composition of an entire workflow, we need additional meta-data on the rest of the KD processes. One straightforward type of useful meta-data consists of the preconditions that need to be fulfilled before the process can be used, and the effect it has on the data it is given. As such, we can transform the workflow creation problem into a *planning* problem, aiming to find the best plan, the best sequence of actions (process applications), that arrives at our goal - a final model.

3.4.1 The Intelligent Discovery Electronic Assistant (IDEA)

A first such system is the Intelligent Discovery Electronic Assistant (IDEA) (Bernstein et al. 2005). It regards preprocessing, modeling and postprocessing techniques as operators, and returns all plans (sequences of operations) that are possible for the given problem. It contains an ontology describing the preconditions and effects of each operator, as well as manually defined heuristics (e.g. the speed of an algorithm), which allows it to produce a ranking of all generated plans according to the user's objectives.

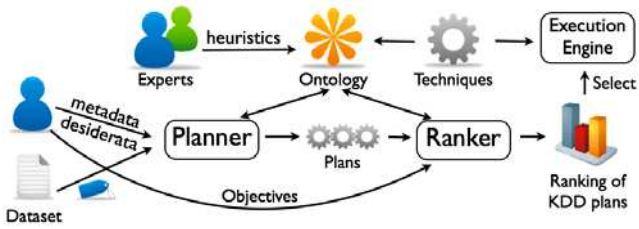


Figure 3.6: The architecture of IDEA. Derived from Bernstein et al. (2005).

3.4.1.1 Architecture

The architecture of IDEA is shown in Figure 3.6. First, the systems gathers information about the given task by characterizing the given dataset. Furthermore, the user is asked to provide additional metadata and to give weights to a number of heuristic functions such as model comprehensibility, accuracy and speed. Next, the planning component will use the operators that populate the ontology to generate all KD workflows that are valid in the user-described setting. These plans are then passed to a heuristic ranker, which will use the heuristics enlisted in the ontology to calculate a score congruent with the user's objectives (e.g. building a decision tree as fast as possible). Finally, this ranking is proposed to the user which may select a number of processes to be executed on the provided data. After the execution of a plan, the user is allowed to review the results and alter the given weights to obtain new rankings. For instance, the user might sacrifice speed in order to obtain a more accurate model. Finally, if useful partial workflows have been discovered, the system also allows to extend the ontology by adding them as new operators.

3.4.1.2 Meta-knowledge

IDEA's meta-knowledge is all contained in its ontology. It first divides all operators into preprocessing, induction or postprocessing operators, and then further into subgroups. For instance, induction algorithms are subdivided into classifiers, class probability estimators and regressors. Each process that populates the ontology is then described further with a list of properties, shown in Figure 3.7. It includes the required input (e.g. a decision tree for a tree pruner), output (e.g. a model), preconditions (e.g. 'continuous data' for a discretizer), effects (e.g. 'removes continuous data', 'adds categorical data' for a discretizer), incompatibilities (e.g. not(continuous data) for naïve Bayes) and a list of manually defined heuristic estimators (e.g. relative speed). Additionally, the ontology also contains recurring partial plans in the form of composite operators.

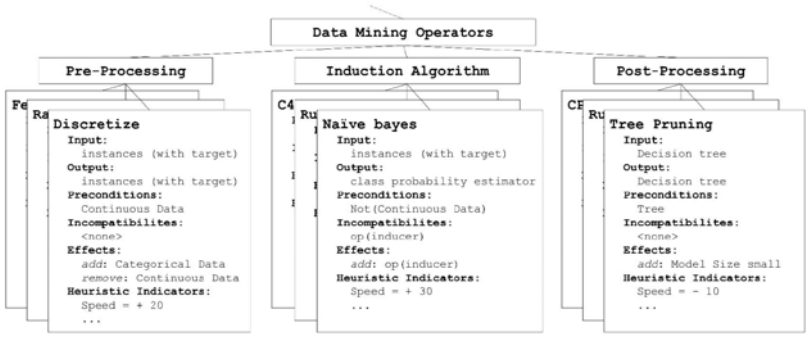


Figure 3.7: Part of IDEA's ontology. Taken from Bernstein et al. (2005).

3.4.1.3 Meta-learning

Though there is no actual meta-learning involved, planning can be viewed as a search for the best-fitting plan given the dataset, just as learning is a search for the best hypothesis given the data. In the case of IDEA, this is achieved by exhaustively generating all possible KD plans, hoping to discover better, novel workflows that experts never considered before. The provided meta-data constitute the initial state (e.g. 'numerical data') and the user's desiderata (e.g. 'model size small') make up the goal state.

3.4.1.4 Discussion

This approach is very useful in that it can provide both experts and less-skilled users with KD solutions without having to know the details of the intermediate steps. The fact that new operators or partial workflows can be added to the ontology can also give rise to *network externalities*, where the work or expertise of one researcher can be used by others without having to know all the details. The ontology thus acts as a central repository of KD operators, although new operators can only be added manually, possibly requiring reimplementation.

The limitations of this approach lie first of all in the hand-crafted heuristics of the operators. Still, this could be remedied by linking it to a meta-database such as used in DMA and learning the heuristics from experimental data. Secondly, the current implementation only covers a small selection of KD operators. Together with the user-defined objectives, this might constrain the search space well enough to make the exhaustive search feasible, but it is unclear whether this would still be the case if large numbers of operators are introduced. A final remark is that most of the used techniques have parameters, whose effects are not included in the planning.

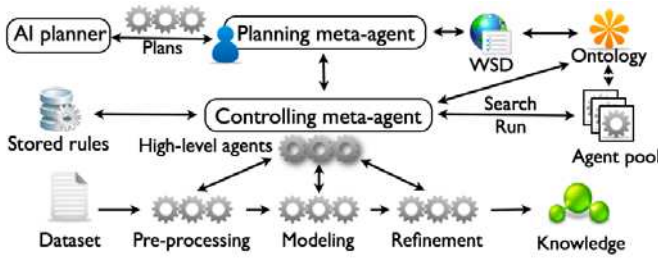


Figure 3.8: The architecture of GLS. Adapted from Zhong et al. (2002).

3.4.2 Global Learning Scheme (GLS)

The 'Global Learning Scheme' (GLS) (Zhong and Ohsuga 1994; Zhong et al. 2001; Zhong et al. 2002) takes a multi-agent system approach to KDD planning. It creates an "organized society of KDD agents", in which each agent only does a small KDD task, controlled by a central planning component.

3.4.2.1 Architecture

Figure 3.8 provides an overview of the system. It consists of a pool of agents, which are described very similarly to the operators in IDEA, also using an ontology to describe them formally. However, next to base-level agents, which basically envelop one DM technique each, there also exist *high-level agents*, one for each phase of the KD process, which instead point to a list of candidate agents that could be employed in that phase. The controlling 'meta-agent' (CMA) is the central controller of the system. It selects a number of agents (high-level agents at first) and sends them to the planning meta-agent (PMA). The PMA then creates a planning problem by transforming the dataset properties and user objectives into a world state description (WSD) for planning and by transforming the agents into planning operators. It then passes the problem to a STRIPS planner (Fikes and Nilsson 1971). The returned plans are passed back to the CMA, which then launches new planning problems for each high-level agent. For instance, say the data has missing values, then the returned plan will contain a high-level missing value imputation agent: the CMA will then send a range of base-level agents to the PMA to build a plan for filling in the missing values. The CMA also executes the resulting sub-plans, and calls for adaptations if they do not prove satisfactory.

3.4.2.2 Meta-knowledge

GLS's meta-knowledge is similar to that of IDEA. The ontology description contains, for each agent, the data types of in- and output, preconditions and

effects. In the case of a base-level agent, it also contains a KD process, otherwise, a list of candidate sub-agents. However, the same ontology also contains descriptors for the data throughout the KD process, such as the state of the data (e.g. raw, clean, selected), whether or not it represents a model and the type of that model (e.g. regression, clustering, rule). This information is used to describe the world state description. Finally, the CMA contains some static meta-rules to guide the selection of candidate agents.

3.4.2.3 Meta-learning

In Zhong et al. (2002), the authors state that a meta-learning algorithm is used in the CMA to choose between several discretization agents or to combine their results. Unfortunately, details are missing. Still, even if the current system does not learn from previous runs (meaning that the CMA uses the same meta-rules every time), GLA's ability to track and adapt to changes performed by the user can definitely be regarded as a form of learning (Brazdil et al. 2009).

3.4.2.4 Discussion

Given the fact that covering the entire KD process may give rise to sequences of tens, maybe hundreds of individual steps, the 'divide and conquer' approach of GLS seems a promising approach. In fact, it seems to mirror the way humans approach the KD problem, as shown in Figure 1.2: identify a hierarchy of smaller subproblems, combine operators to solve them and adapt the partial solutions to each other until we converge to a working workflow. Unfortunately, to the best of our knowledge, there are no thorough evaluations of the system, and it remains a work in progress. It would be interesting to replace the fixed rules of the CMA with a meta-learning component to select the most promising agents, and to use IDEA-like heuristics to rank possible plans.

3.5 Case-based reasoning

Planning is especially useful when starting from scratch. However, if successful workflows were designed for very similar problems, we could simply reuse them.

3.5.1 CITRUS

CITRUS (Engels 1996; Wirth et al. 1997) is built as an advisory component of Clementine, a well-known KDD suite. An overview is shown in Figure 3.9. It contains a knowledge base of available 'processes' (KD techniques) and 'streams' (sequences of processes), entered by experts and described with pre- and postconditions. To use it, the user has to provide an abstract task description, which is appended with simple data statistics, and choose between several modi of operation. In the first option, the user simply composes the

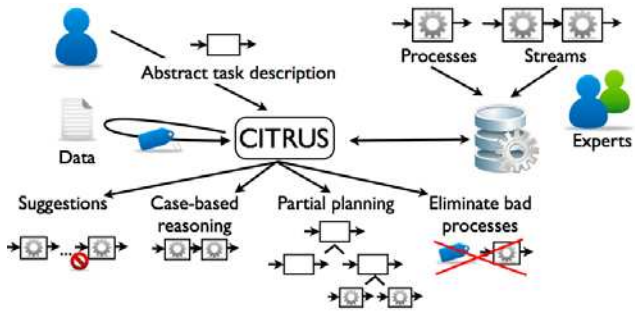


Figure 3.9: The architecture of CITRUS. Derived from Wirth et al. (1997).

entire KDD process (stream) by linking processes in Clementine's editor, in which case CITRUS only checks the validity of the sequence. In the second option, case-based reasoning is used to propose the most similar of all known streams. In the third option, CITRUS assists the user in decomposing the task into smaller subtasks, down to the level of individual processes. While pre- and postconditions are used in this process, no planning is involved. Finally, the system also offers some level of algorithm selection assistance by eliminating those processes that violate any of the constraints.

3.5.2 ALT

ALT (Lindner and Studer 1999) is a case-based reasoning variation on the DMA approach. Next to StatLog-type meta-features, it adds a number of simple algorithm characterizations (see Section 2.5). It has a meta-database of 'cases' consisting of data meta-features, algorithm meta-features and the performance of the algorithm. When faced with a new problem, the user can enter application restrictions beforehand (e.g. 'the algorithm should be fast and produce interpretable models'), and this information is then appended to the data meta-features of the new dataset, thus forming a new case. The system then makes a prediction based on the three most similar cases. The meta-data consists of 21 algorithms, 80 datasets, 16 StatLog data characteristics, and 4 algorithm characteristics.
The two previous approaches share a common shortcoming: the user still faces the difficult task of *adapting* it to her specific problem. The following systems try to alleviate this problem by offering additional guidance.

3.5.3 MiningMart

The MiningMart project (Morik and Scholz 2004) is designed to allow successful preprocessing workflows to be shared and reused, irrespective of how the data is stored. While most of the previously discussed systems expect a dataset in a certain format, MiningMart works directly on any SQL database of any size.

The system performs the data preprocessing and transformation steps in the (local) database itself, either by firing SQL queries or by running a fixed set of efficient operators (able to run on very large databases) and storing the results in the database again. Successful workflows can be shared by describing them in an XML-based language, dubbed M4, and storing them in an online case-base. The case description includes the workflow's inherent structure as well as an ontological description of business concepts to facilitate searching for cases designed for certain goals or applications.

3.5.3.1 Architecture

The architecture of the system is shown in Figure 3.10. To map uniformly described preprocessing workflows to the way data is stored in specific databases, it offers three levels of abstraction, each provided with graphical editors for the end-user. We discuss them according to the viewpoint of the users who wish to reuse a previously entered case. First, they use the business ontology to search for cases tailored to their specific domain. The online interface then returns a list of cases (workflows). Next, they can load a case into the *case editor* and adapt it to her specific application. This can be done on an abstract level, using abstract concepts and relations such as 'customer', 'product' and the relation 'buys'. They can each have a range of properties, such as 'name' and 'address', and can be edited in the *concept editor*. In the final phase, these concepts, relations and properties have to be mapped to tables, columns, or sets of columns in the database using the *relation editor*. All details of the entire process are expressed in the M4 language[3] and also stored in the database. Next, the *compiler* translates all preprocessing steps to SQL queries or calls to the operators used, and executes the case. The user can then adapt the case further (e.g. add a new preprocessing step or change parameter settings) to optimize its performance. When the case is finished, it can be annotated with an ontological description and uploaded to the case-base for future guidance.

3.5.3.2 Meta-knowledge

MiningMart's meta-knowledge can be divided in two groups. First, there is the fine-grained, case-specific meta-data that covers all the meta-data entered by the user into the M4 description, such as database tables, concepts, and workflows. Second, there is more general meta-knowledge encoded in the business ontology, i.e. informal annotations of each case in terms of its goals and constraints, and in the description of the operators. Operators are stored in a hierarchy consisting of 17 'concept' operators (from selecting rows to adding columns with moving windows over time series), 4 feature selection operators, and 20 feature construction operators, such as filling in missing values, scaling,

[3]Examples of M4 workflows can be downloaded from the online case-base: http://mmart.cs.uni-dortmund.de/caseBase/index.html

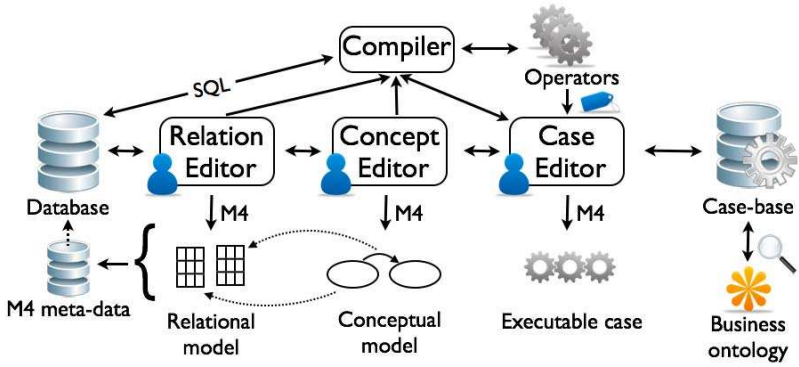


Figure 3.10: MiningMart architecture. Derived from Morik and Scholz (2004).

discretization and some learning algorithms used for preprocessing: decision trees, k-means and SVMs. The M4 schema also captures known preconditions and assertions of the operators, similar to the preconditions and effects of IDEA and GLS. Their goal is to guide the construction of valid preprocessing sequences, as was done in CITRUS. The case-base is available online (Morik and Scholz 2004) and currently contains 6 fully-described cases.

3.5.3.3 Meta-learning

While it is a CBR approach, there is no automatic recommendation of cases. The user can only browse the cases based on their properties manually.

3.5.3.4 Discussion

The big benefit of the MiningMart approach is that users are not required to transform the data to a fixed, often representationally limited format, but can instead adapt workflows to the way the source data is actually being stored and manipulated. Moreover, a common language to describe KD processes would be highly useful to exchange workflows between many different KD environments. Pooling these descriptions would generate a rich collection of meta-data for meta-learning and automated KD support. M4 is certainly a step forward in the development of such a language.
Compared to IDEA, MiningMart focusses much more on the preprocessing phase, while the former has a wider scope, also covering model selection and postprocessing steps. Next, while both approaches describe their operators with pre- and postconditions, MiningMart only uses this information to guide the user, not for automatic planning. MiningMart could, in principle, be extended with an IDEA-style planning component, at least if the necessary meta-data can also be extracted straight from the database.

There are also some striking similarities between MiningMart and GLS. GLS's agents correspond to MiningMart's operators and its controller (CMA) corresponds to MiningMart's compiler. Both systems use a hierarchical description of agents/operators, which are all described with pre- and postconditions. Still, while MiningMart focuses on preprocessing, GLS wants to cover the entire KD process), while MiningMart interfaces directly with databases, GLS requires the data to be prepared first, and while MiningMart stores the meta-data of the workflows for future use, GLS doesn't.

3.5.4 The Hybrid Data Mining Assistant (HDMA)

The Hybrid Data Mining Assistant[4] (HDMA) (Charest and Delisle 2006; Charest et al. 2006; Charest et al. 2008) also tries to provide advice for the entire knowledge discovery process using ontological (expert) knowledge, as IDEA does, but it does not provide a ranking of complete processes. Instead, it provides the user with expert advice during every step of the discovery process, showing both the approach used in similar cases and more specific advice for the given problem triggered by ontological knowledge and expert rules.

3.5.4.1 Architecture

An overview of the system is provided in Figure 3.11. It is based on two stores of information. The first is a repository of KDD 'cases', detailing previous workflows, while the second is an ontology of concepts and techniques and a number of rules concerning those techniques. The system is assumed to be associated with a DM toolkit for actually running and evaluating the techniques.
The user first provides a dataset, which is characterized partly by the system, partly by the user (see below). The given problem is then compared to all the stored cases and two scores are returned for each case: the similarity with the current case, and the 'utility' of the case, as assessed by previous users. After the user has selected a case, the system starts cycling through five phases of the KDD process, as identified by the CRISP-DM (Chapman et al. 1999) model. At each phase the user is provided with the possible techniques that can be used in that phase, the techniques that were used in the selected case, and a number of recommendations generated by applying the stored rules in the context of the current problem. The generated advice may complement, encourage, but also advice against the techniques used in the selected case. As such, the user is guided in adapting the selected case to the current problem.

3.5.4.2 Meta-knowledge

In the case base, each case is described by 66 attributes. The first 30 attributes include StatLog-like meta-features, but also qualitative information entered by

[4]This is not the official name, we only use it here to facilitate our discussion.

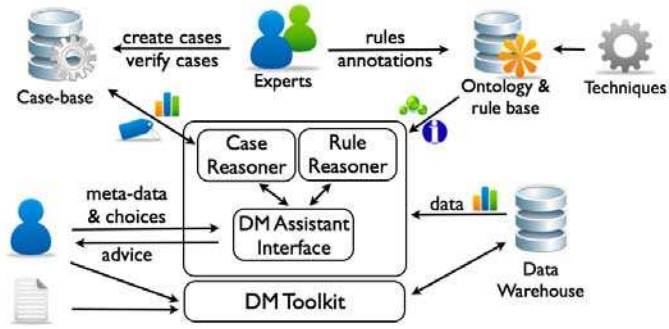


Figure 3.11: The architecture of HDMA. Adapted from Charest et al. (2008).

users, such as the type of business area (e.g. engineering, marketing, finance,...) or whether the data can be expected to contain outliers. Then, the proposed workflow is described by 31 attributes, such as the used preprocessing steps, how outliers were handled, which model was used, which evaluation method was employed and so on. Finally, 5 more attributes evaluate the outcome of the case, such as the level of satisfaction with the approach used in each step of the KD process. A number of seed cases were designed by experts, and additional cases can be added after evaluation.
The ontology, on the other hand, captures a taxonomy of DM concepts, most of which where elicited from CRISP-DM. A small part is shown in Figure 3.12. For instance, it shows that binarization and discretization are two data transformation functions, used to make the data more suitable to a learning algorithm, which is part of the data preparation phase of the CRISP-DM model. The dashed concepts are the ones for which exist specific KD-techniques (individuals). These individuals feature in expert rules that describe when they should be used, depending on the properties of the given problem. Some of these rules are heuristic (e.g. "use an *aggregation* technique if the *example count* is greater than 30000 and the data is of a *transactional* nature"), while others are not (e.g. "if you select the *Naive Bayes* technique, then a *nominal target* is required"). In total, the system contains 97 concepts, 58 properties, 63 individuals, 68 rules and 42 textual annotations. All knowledge is hand-crafted by experts and formalized in the OWL-DL ontology format (Dean et al. 2003) and the SWRL rule description language (Horrocks et al. 2004).

3.5.4.3 Meta-learning

The only meta-learning occurring in this system is contained in the case based reasoning. It uses a kNN approach to select the most similar case based on the characterization of the new problem, using a feature-weighted, global similarity measure. The weight of each data characteristic is pre-set by experts.

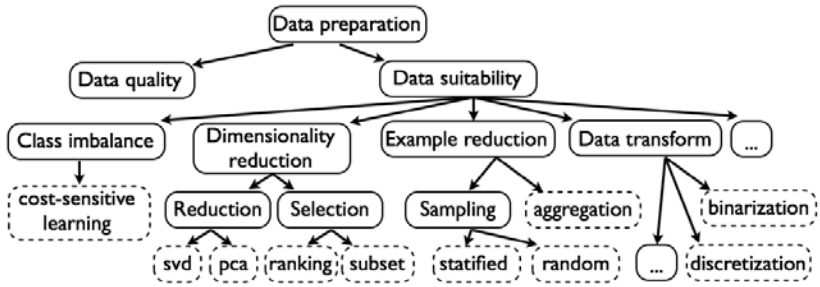


Figure 3.12: Part of the HDMA ontology. Adapted from Charest et al. (2008).

3.5.4.4 Discussion

HDMA is quite unique in that it leverages both declarative information, viz. concepts in the ontology and case descriptions, as well as procedural information in the form of rules. Because of the latter, it can be seen as a welcome extension of Consultant-2. It doesn't solve the problem of (semi-)automatically finding the right KD approach, but it provides practical advice to the user during every step of the KD process. Its biggest drawback is probably that it relies almost entirely on input from experts. Besides from the fact that the provided rules and heuristics may not be very accurate in some cases, this makes it hard to maintain the system. For every new technique (or just a variant of an existing one), new rules and concepts will have to be defined to allow the system to return proper advice. As with Consultant-2, some of these issues may be resolved by introducing more meta-learning into the system, e.g. advice on the proper weights for each data characteristic in the case based reasoning step, to update the heuristics in the rules and to find new rules based on experience. Finally, the case-based advice, while useful in the first few steps, may lose its utility in the later stages of the KD process. More specifically, say the user chooses a different preprocessing step as applied in the proposed case, then the subsequent stages of that case (e.g. model selection) may lose their applicability. A possible solution here would be to update the case selection after each step, using the partial solution to update the problem description.

3.5.5 NExT

NExT, the Next generation Experiment Toolbox (Bernstein and Daenzer 2007) is an extension of the IDEA approach to case-based reasoning and to the area of *dynamic processes*, in which there is no guarantee that the proposed workflow will actually work, or even that the atomic tasks of which it consists will execute without fault. First, it contains a knowledge base of past workflows and uses case-based reasoning to retrieve the most similar ones. More often than not, only parts of these workflows will be useful, leaving holes which need to be filled

with other operators. This is where the planning component comes in: using the preconditions and effects of all operators, and the starting conditions and goals of the KD problem, it will try to find new sequences of operators to fill in those holes.

However, much more can go wrong when reusing workflows. For instance, a procedure may have a parameter setting that was perfect for the previous setting, but completely wrong for the current one. Therefore, NExT has an ontology of possible problems related to workflow execution, including resolution strategies. Calling a planner is one such strategy, other strategies may entail removing operators, or even alerting the user to fix the problem manually.

Finally, it does not start over each time the workflow breaks. It records all the data processed up to the point where the workflow breaks, tries to resolve the issue, and then continues from that point on. As such, it also provides an online log of experimentation which can be shared with other researchers willing to reproduce the workflow.

NExT has only recently been introduced and thorough evaluations are still scarce. Nevertheless, its reuse of prior workflows and semi-automatic adaption of these workflows to new problems seems very promising.

3.6 Querying

A final way to assist users is to automatically answer any kind of question they may have about the applicability, general utility or general behavior of KD processes, so that they can make informed decisions when creating or altering workflows. While the previous approaches only stored a selection of meta-data necessary for the given approach, we could collect a much larger collection of meta-data of possible interest to users, and organize all this data to allow the user to write queries in a predefined query language, which will then be answered by the system based on the available meta-data. While this approach is mostly useful for experts knowing how to ask the questions and interpret the results, frequently asked questions could be wrapped in a simpler interface for use by a wider audience.

3.6.1 Advanced Meta-Learning Architecture (AMLA)

One example of this approach is the Advanced Meta-Learning Architecture[5] (AMLA) (Grabczewski and Jankowski 2007). It is a data mining toolbox architecture aimed at running ML algorithms more efficiently whilst inherently supporting meta-learning by collecting all meta-data from every component in the system and allowing the user to query or manipulate it.

[5]This is again not the officially coined name of the system.

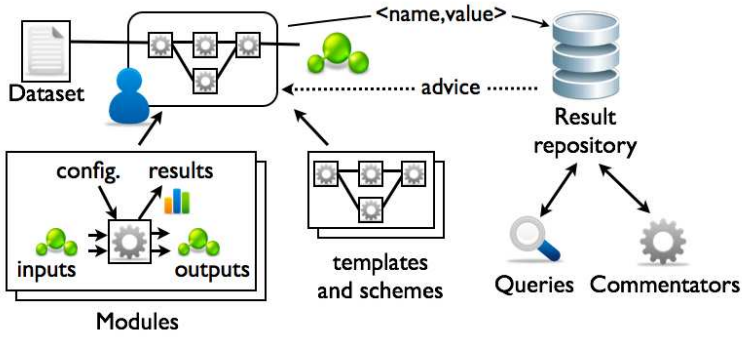


Figure 3.13: The architecture of AMLA. Derived from Grabczewski and Jankowski (2007).

3.6.1.1 Architecture

An overview of the system is provided in Figure 3.13. It encapsulates all standard KD operations with modules which can then be used as building blocks for KD processes. Each module has a set of inputs and outputs, each of which is a 'model', i.e. an actual model (e.g. a decision tree) or a dataset. It also has a special 'configuration' input which supplies parameter settings, and a special 'results' output, which produces meta-data about the process. For instance, in the case the process encapsulates a decision tree learner, the input would be training data, the configuration would state its parameters, the output would provide a decision tree model and the results would state, for instance, the number of nodes in the tree. All results are represented as name-value pairs and stored in a repository, which collects all results generated by all modules. The modules are very fine grained. For instance, there are separate modules for testing a model against a dataset (which export the performance evaluations to the repository) and for sub-components of certain techniques, such as base-learners in ensembles, kernels in SVMs and so on. Modules often used together can be combined in 'schemes', which again have their own inputs, outputs, configurations and results. Schemes can also contain unspecified modules, to be filled in when used, in which case they are called templates. There also exist 'repeater' modules which repeat certain schemes many times, for instance for cross-validation. This compositionality allows to build arbitrarily complex KD workflows, quicker implementation of variants of existing algorithms, and a more efficient execution, as modules used many times only have to be loaded once. The result repository can be queried by writing short scripts to extract certain values, or to combine or filter the results of previous queries. Finally, 'commentators' can be written to perform frequently used queries, e.g. statistical significance tests, and store their results in the repository.

3.6.1.2 Meta-knowledge

The stored meta-data consists of a large variety of name-value pairs collected from (and linked to) all previously used modules, templates and schemes. They can be of any type.

3.6.1.3 Meta-learning

Meta-learning is done manually by programming queries. The query's constraints can involve the modules having generated the meta-data or the type of properties (the name in the name-value pairs). As such, previously obtained meta-data can be extracted and recombined to generate new meta-knowledge. Secondly, templates with missing modules can also be completed by looking up which were the most successful completed templates in similar problems.

3.6.1.4 Discussion

This is indeed a very fundamental approach to meta-learning, in the sense that it keeps track of all the meta-data generated during the design and execution of KD processes. On the other hand, each query has to be written as a small program that handles the name-value pairs, which might make it a bit harder to use, and the system is still very much under construction. For instance, at this stage of development, it is not entirely clear how the results obtained from different workflows can be compared against each other. It seems that many small queries are needed to answer such questions, and that a more structured repository might be required instead.

3.7 A new platform for intelligent KD support

3.7.1 Summary

An overview of the previously discussed architectures is shown in Table 3.1. The columns represent consecutively the portion of the KD process covered, the system type, how it interacts with the user, what type of meta-information is stored, the data it has been trained on, which KD processes it considers, which evaluation metrics are covered, which meta-features are stored and which meta-learning techniques are used to induce new information, make predictions or otherwise advise the user.
As the table shows, and the systems' discussions have indicated, each system has its own strengths and weaknesses, and cover the KD process to various extents. Some algorithms, like MiningMart and DMA provide a lot of support and gather a lot of meta-information about a few, or only one KD step, while others try to cover a larger part of the entire process, but consider a smaller number of techniques or describe them with less information, usually provided

Table 3.1: Comparison of prior meta-learning architectures

	KD step[a]	type	user interface	type meta-info	data scope	KD process scope	evaluation scope	meta-feature scope	meta-learning technique
Consultant-2		Expert system	Q&A sessions	heuristic expert rules	NA	10 algorithms from MLT	speed, model-properties	simple statistics, prior knowledge	static meta-model
DMA		meta-model	Webinterface: algo ranking	collection of experiments	67 datasets + 83 from users	10 classific. algorithms	accuracy and speed	7 modified StatLog features	kNN and ranking
NOEMON		meta-model	Algorithm ranking	collection of decision trees	77 datasets (UCI)	3 classific. algorithms	accur., speed and memory	idem StatLog + histograms	Vote over models and ranking
IDEA		Planning	Ranking of KD plans	ontology of KD operators with heuristics	NA	10 preproc., 6 ML algo's, 5 postproc.	accuracy, speed, model-properties	IOPE[b] + basic data properties	AI planning
GLS		Planning	User reviews partial plans	ontology of KD operators + agent meta-rules	NA	7 preproc., 7 ML algo's, 2 postproc.	NA	NA	emergent agent behavior + AI planning
CITRUS		CBR	Clementine editor	case base + process constraints	NA	Clementine processes	depends on operator	workflow description	CBR, checking constraints
ALT		CBR	Returns 'best' algorithm	collection of cases (experiments)	80 datasets (UCI+more)	21 classific. algorithms	accuracy and speed	StatLog features + 4 algorithm feats	Case-based reasoning
MiningMart		CBR	Editors: create or adapt cases	collection of cases + business ontology	NA (any database)	41 preprocessors	NA	business description of each workflow	none (manual CBR)
HDMA		CBR	Step-by-step, show cases + advice	set of KD cases + set of heuristic expert rules, ontological	NA	selection of WEKA algorithms	user ratings	30 data features, 31 solution features, 5 user ratings	Case-based reasoning
NExT		CBR + Planning	Interaction to solve workflow issues	ontology of workflow issues and solutions	NA	NA	NA	IOPE[b] + workflow description	CBR + AI planning
AMLA		Querying	Process editor + Query lang.	Name-value pairs from any component	NA	extensible	extensible	extensible	querying + fill in templates

[a]The boxes stand consecutively for the Data Selection, Preprocessing+transformation, Data Mining, and Postprocessing step.
[b]IOPE=Inputs, Outputs, Preconditions and Effects of KD operators

by experts. All systems also expect very different things from their users. Some, especially CITRUS, GDL and AMLA, put the user (assumed to be an expert) firmly in the driver's seat, leaving every important decision to her. Others, like Consultant-2, GLS, MiningMart, HDMA and NExT allow the user to interfere in the workflow creation process, often explicitly asking for input. Finally, the meta-model systems and IDEA almost completely automate this process, offering suggestions to users which they may adopt or ignore. Note that, with the exception of Consultant-2, none of the systems performing algorithm selection also predict appropriate parameters, unless they are part of a prior workflow.
A few systems obviously learned from each other. For instance, DMA, NOEMON and ALT learned from StatLog, NExT learned from IDEA and HDMA learned from prior CBR and expert system approaches. Still, most systems are radically different from each other, and there is no strong sense of convergence to a general platform for KD workflow generation.

3.7.2 Desiderata

We now look forward, striving to combine the best aspects of all prior systems:

Extensibility Every KD support system that only covers a limited number of techniques will at some point become obsolete. It is therefore important that new KD techniques can be added easily. (GLS, AMLA)

Integration Ideally, the system should be able to execute the workflow. It should also be able to call on some existing KD tools to execute processes, instead of reimplementing every KD process in a new environment. Indeed, as new types of algorithms are created and new data preprocessing methods are developed, it becomes infeasible to re-implement this continuous stream of learning approaches. (Consultant-2, HDMA)

Self-maintenance Systems should be able to update their own meta-knowledge as new data or new techniques become available. While experts are very useful to enrich the meta-knowledge with successful models and workflows, they cannot be expected to offer all the detailed meta-data needed to learn from previous KD episodes. (DMA, NOEMON, ALT, AMLA)

Common language As more and more KD techniques are introduced, it becomes infeasible to locally run all the experiments needed to collect the necessary meta-data. It is therefore better to take a *community-based* approach, in which descriptions of KD applications can be generated by, and shared with, the entire community. To achieve this, a common language should be used to make all the stored meta-data interchangeable. MiningMart)

Ontologies Meta-knowledge should be stored in a way that makes it machine-interpretable, so that KD support tools can use it effectively. This is reflected by the use of ontologies in many of the discussed systems. Second, consensus should also be sought to establish a common vocabulary

for the concepts and relations used in KD research, as a basis for common languages. (IDEA, GLS, MiningMart, HDMA, NExT)

Querying The stored meta-data should also be stored in a way that facilitates manual querying by users. Indeed, we cannot foresee all the possible uses of meta-data beforehand, and should therefore enable the user to perform custom investigations. (AMLA)

Workflow reuse and adaptation Since most KD applications focus on a limited number of tasks, it is very likely that there exist quite a few prior successful workflows that have been designed for that task. Any intelligent KD support system should therefore be able to return similar workflows, but also offer extensive support to adapt them to the new problem. (MiningMart, HDMA, NExT)

Planning When no similar workflows exist, or when parts of workflows need to be redesigned, using the KD processes' preconditions and effects for planning is clearly a good approach, although care should be taken that the planning space is not prohibitively large. (IDEA,GLS,NExT)

Learning Last but not least, the system should get better as it gains more experience. This includes planning: over time, the system should be able to optimize its planning approach, e.g. by updating heuristic descriptions of the operators used. (DMA, NOEMON, GLS, all CBR approaches)

3.7.3 Architecture

These aspects can be combined in the KD support platform shown in Figure 3.14. It is based on the work in this thesis and on the description of the DM laboratory in Kalousis et al. (2008).

3.7.3.1 A community-based approach

This platform is set in a *community-based* environment, with many people using the same KD techniques. It could cover a very general domain, such as KD as a whole, or a more specific one, such as bio-technology, which may result in more specific types of meta-data and more specific ontologies.

Notice that first of all, a common language is used to exchange meta-data between the KD assistant and the tools with which it interacts. First, on the right, there are the many DM/KD toolboxes that implement a wide variety of KD processes. They exchange workflows, descriptions of algorithms and datasets, produced models and evaluations of the implemented techniques.

On the left, there are *web services*. In the last couple of years, there has been a strong movement away from locally implemented toolboxes and datasets, and towards KD processes and databases offered as online services. These *Service Oriented Architectures* (SOAs) (Foster 2005) define standard interfaces and protocols that allow developers to encapsulate tools as services that clients can access without knowledge of, or control over, their internal workings. In the

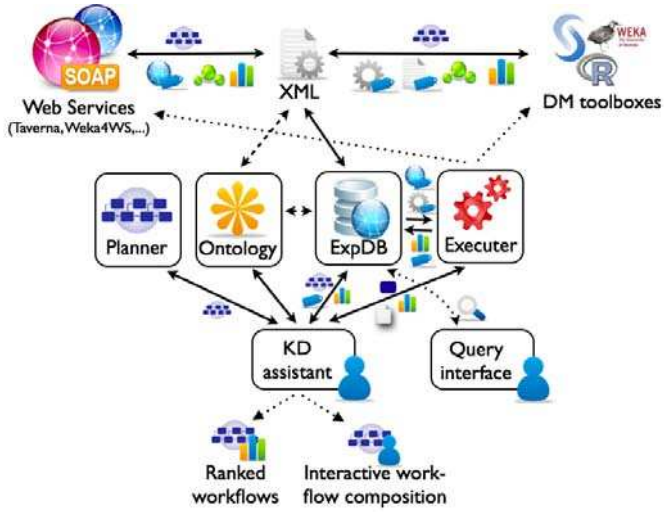


Figure 3.14: A proposed platform for intelligent KD support.

case of a database, this interface may offer to answer specific queries, e.g. a database of airplane flight may return the flight schedule for a specific plane. In the case of a learning algorithm, the interface may accept a dataset (or a database implemented as a web service) and return a decision tree. Existing tools are discussed in Section 3.7.4.3.

While we did not explicitly include experts as a source of meta-knowledge, we assume that they will build workflows and models using the available web services and toolboxes.

When the KD assistant interacts with these services, it will exchange workflows, descriptions of the web services (where to find them and how to interact with them), produced models and evaluation results. Given the rise of web services, XML is a very likely candidate as the modeling language for this common language. Web services interact with each other using SOAP (Simple Object Access Protocol) messages, and describe their interface in WSDL (Web Services Description Language), both of which are described in XML. XML is also used by many KD/DM toolboxes to serialize their data and produced models.

3.7.3.2 A DM/KD ontology

The vocabulary used for these descriptions should be described in a common ontology. Despite many proposed ontologies, there is of yet no consensus on an ontology for DM or KD, but we hope that such an ontology will arise over the coming years. Imagine the internet without HTML, and it is obvious that a common language for KD is essential for progress in this field. The ontology

should also provide detailed descriptions of KD processes, such as their place in the hierarchy of different processes, the internal structure of composite learning algorithms, preconditions and effects of all known KD operators, and the parameters that need to be tuned in order to use them. Additional information can be added to extend the ontology to engender further KD support (such as the list of KD issues and solutions in NExT).

3.7.3.3 A meta-data repository

All generated meta-data is automatically stored and organized in a central repository. This repository, which we call an *experiment database* (ExpDB), is the main focus of this work. It collects all the details of all performed experiments, including the workflows used and the obtained results, thus offering a complete log of the experimentation which can be used to reproduce the submitted studies. Moreover, using the ontology, it automatically organizes all the data so it can be easily queried by expert users, allowing it to answer almost any kind of question about the properties and the performance of the used KD techniques, using the meta-data from many submitted studies. It serves as the long-term memory of the KD assistant, but also as that of the individual researcher and the community as a whole. It will be frequently polled by the KD assistant to extract the necessary meta-data, and should contain a query interface for manual investigations as well. The database itself can be wrapped as a web service, allowing automatic interaction with other tools and web services.

3.7.3.4 Planning and execution

The KD assistant interacts with two more components: an AI planner used to solve any planning problems, and an executer component which runs the actually implemented KD algorithms in KD/DM toolboxes or web services to execute workflows, or perhaps to do other calculations such as computing meta-features if they are not implemented in the KD assistant itself. The executer polls the ExpDB to obtain the necessary descriptions and locations of the featured web services, algorithms or datasets.

As for the output generated by the KD assistant, we foresee two important types of advice (beyond manual querying), although surely many more kinds of advice are possible. The first is a ranked list of workflows produced by the KD assistant (even if this workflow only consists of a single learning algorithm). The second, possibly more useful approach is a semi-automatic interactive process in which the user actively participates during the creation of useful workflows. In this case, the KD assistant can be associated with a workflow editing tool (such as Taverna), and assist the users as they compose workflows, e.g. by checking the correctness of a workflow, by completing partial workflows using the planner, or by retrieving, adapting or repairing previously entered workflows.

3.7.4 Implementation

3.7.4.1 ExpDBs and ontologies

While such a KD support system may still be some way off, recently, a great deal of work has been done that brings us a lot closer to realizing it.
First of all, experiment databases are the main focus of this thesis, and will be covered in Chapter 4. We also propose a DM ontology in Chapter 5, and an XML-based language for exchanging KD experiments in Chapter 6. Note that building such an ontology should be a collaborative process, and we will therefore build upon some other recently proposed ontologies in DM, such as OntoDM (Panov et al. 2008; Panov et al. 2009), DMO (Kalousis et al. 2008; Hilario et al. 2009; Kietz et al. 2009), KDDONTO (Hidalgo et al. 2009) and KD ontology (Záková et al. 2008; Záková et al. 2009).
Some parts of these ontologies extend beyond the scope of this thesis, but are perfectly fit for realizing the proposed platform. For instance, DMO (Kietz et al. 2009) covers the preconditions and effects of all KD operators, expressed as rules in the Semantic Web Rule Language (SWRL) (Horrocks et al. 2004).

3.7.4.2 Planning

Concerning planning, several approaches have been outlined that translate the ontological descriptions of KD operators to a planning description based on the standard Planning Domain Description Language (PDDL) by using an *ontological reasoner* to query their KD ontologies before starting the actual planning process (Klusch et al. 2005; Liu et al. 2007; Sirin et al. 2004). Other approaches integrate a reasoning engine directly in the planner, so that the planner can directly query the ontology when needed (Kietz et al. 2009; Záková et al. 2008; Záková et al. 2009).
Klusch et al. (2005) and Liu et al. (2007) use a classical STRIPS planner to produce the planning, while Sirin et al. (2004) and Kietz et al. (2009) propose an Hierarchical Task Network (HTN) planning approach (Sacerdoti 1974), in which each *task* has a set of associated *methods*, which decompose into a sequence of (sub)tasks and/or *operators* that, when executed in that order, achieve the given task. The main task is then recursively decomposed until we obtain a sequence of applicable operators. Somewhat similar to GLS, this divide-and-conquer approach effectively reduces the planning space.
Záková et al. (2009) uses an adaptation of the heuristic Fast-Forward (FF) planning system (Homann and Nebel 2001). Moreover, it allows the completed workflows to be executed on the Orange DM platform, and vice-versa: workflows composed in Orange are automatically annotated with their KD ontology so that they can be used for ontology querying and reasoning. It does this by mapping their ontology to the ontology describing the Orange operators.
Finally, Kalousis et al. (2008) propose a system that will combine planning and meta-learning. It contains a kernel-based, probabilistic meta-learner which

dynamically adjusts transition probabilities between DM operators, conditioned on the current application task and data, user-specified performance criteria, quality scores of workflows applied in the past to similar tasks and data, and the users profile (based on quantified results from, and qualitative feedback on, her past DM experiments). Thus, as more workflows are stored as meta-knowledge, and more is known about the users building those workflows, it will learn to build workflows better fit to the user.

3.7.4.3 Web services

The development of service oriented architectures for KD, also called *third-generation DM/KD*, has gathered steam, helped by increasing support for building workflows of web services.
Taverna (Roure et al. 2009), for instance, is a system designed to help scientists compose executable workflows in which the components are web services, especially for biological in-silico experiments. Similarly, Triana (Taylor et al. 2007) supports workflow execution in multiple environments, such as peer-to-peer (P2P) and the Grid. Discovery Net (Rowe et al. 2003) and ADMIRE (Le-Khac et al. 2006) are platforms that make it easier for algorithm designers to develop their algorithms as web services and Weka4WS (Talia et al. 2005) is a framework offering the algorithms in the WEKA toolkit as web services.
Finally, Orange4WS (Podpecan et al. 2009) is a *service-oriented KD* platform based on the Orange toolkit. It wraps existing web services as Orange workflow components, thus allowing to represent them, together with Orange's original components, as graphical widgets for manual workflow composition and execution. It also proposes ways to wrap local algorithms as web services.

3.8 Conclusions

In this chapter, we have provided a survey of the different solutions proposed to support the design of KD processes through the use of meta-knowledge and highlighted their strengths and weaknesses. We observed that most of these systems are very different, and were seemingly developed independently from each other, without capitalizing on the benefits of prior systems. Learning from these prior architectures, we proposed a new, community-based platform for KD support that combines their best features, and showed that recent developments have brought us close to realizing it. Still, important challenges remain, most notably the development of a flexible and scalable experiment database and the definition of a common language to share complex KD meta-data. In the following chapters, we explicate how these challenges can be met.

Prelude Conclusions

In the prelude to this thesis, we have covered the practice and state-of-the-art in meta-learning research and its applications, providing a foundation for the work in the subsequent parts.
First, in **Chapter 2**, we have provided a survey of research in the field of meta-learning. We identify three increasingly difficult meta-learning settings: ensemble learning (learning from models previously built on the *same* data), transfer learning (learning from models built on *similar* data), and the modeling of learning behavior over *different* kinds of data, each time focussing on *why* these approaches improve the speed and/or accuracy of the underlying base-learners.
Focusing on this last setting, we define a framework for meta-learning to highlight the key questions and proposed solutions in each component of the general meta-learning process:

- The data meta-feature space F, covering simple, statistical, information-theoretic, concept-based, case based and model based properties of the data, in addition to landmarking.
- The algorithm meta-feature space G, covering qualitative as well as measurable properties of the data.
- The problem space P, which we extended with the preprocessed problem space P', covering all possible ways to preprocess a certain dataset.
- The algorithm space A, which we extended with the parameterized algorithm space A'.
- The performance space Y of evaluation measures
- The meta-learner S.

For the latter, we classified all approaches that could be found in literature based on the type of advice they can give: recommendations, rankings, performance estimates and descriptive models. We also describe in detail which algorithms have been tried in previous studies for each of these goals.
Finally, we recast the literature in the extended framework introduced in this chapter, and two important observations were made. First, that most of them use a limited amount of datasets and algorithms, and that, 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. Second, that 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. We will address these issues in the next part of this thesis by building community-wide, organized repositories of meta-data on learning processes. The extended framework for meta-learning proposed here will be central to their design.

Next, **Chapter 3** provides a survey of the various architectures that have been developed, or simply proposed, to build KD support systems. They all consist of integrated repositories of meta-knowledge on the KD process and leverage that information to propose useful workflows. Our main observation is that most of these systems are very different, and were seemingly developed independently from each other, without really capitalizing on the benefits of prior systems.

By bringing these different architectures together and highlighting their respective strengths and weaknesses, we compiled a list of desiderata for the development of future KD support system. We also propose a new KD support architecture that combines the best aspects of earlier systems, and that solves many of their limitations by taking a community-based approach built around experiment databases that allow many people using the same KD techniques to collaborate and benefit from each other's meta-data. While such a KD support system may still be some way off, we show that recently, a great deal of work has been done that brings us a lot closer to realizing it.

Part I

Organizing Machine Learning Information

Outline Part I

In the previous chapters, we provided a perspective overview of the state-of-the-art and progress made in the field of meta-learning and its most practical application: offering advice to practitioners based on prior experience. This analysis revealed a number of avenues for improvement, and in this part of the thesis, we describe how we can put these into practice.

One important observation was that meta-learning approaches are typically aimed at practitioners, not at *designers* of learning algorithms. Still, as illustrated in Figure 2.5, meta-learning is a very important component of algorithm design: it is used to evaluate how new algorithms perform in general or under specific conditions, to investigate *why* they behave as such, and to improve them based on the gained insight. Currently, this is a very laborious enterprise requiring manual experiment setups, which slows down progress and limits the depth with which these investigations can be performed.

Conversely, the thousands of experiments run by algorithm designers to evaluate their algorithms are not being used to update the collection of meta-data that drive meta-learning approaches. As such, meta-learning studies need to repeat those experiments, which is prohibitively expensive. Indeed, we observed that most meta-learning studies only use a limited number of datasets, preprocessors, algorithms or parameter variations.

In the following chapters, we address these issues by offering a *community-based* approach to experimentation in machine learning, in which the massive streams of experiments that are being executed to evaluate learning algorithms are automatically shared with researchers all over the world, and organized into repositories, *experiment databases*, that intelligently organize all the meta-data so it can be investigated in depth. Whereas the StatLog and METAL projects shared meta-data between meta-learning researchers, experiment databases allow for a much more profound, community-wide exchange of experiment details. By pooling the meta-data generated by meta-learning studies (e.g. dataset and algorithm characterizations), algorithm designers (e.g. detailed evaluations of new algorithms) and practitioners (e.g. exploratory evaluations of algorithms on new datasets), we create a tremendously valuable resource for all types of investigations into the behavior of learning algorithms.

We discuss the motivations behind these experiment databases in depth in Chapter 4. Next to providing a rich source of data which can be mined to discover patterns in learning behavior and leveraged to provide targeted advice on how to tackle new KDD problems, there are also important benefits for algorithm designers and for machine learning research in general. Next, we show

how similar developments in other empirical sciences, such as astrophysics and bioinformatics, have boosted their progress, and use what we learned from these sciences to create a framework for the implementation and use of experiment databases in machine learning.

A first hurdle on the road towards the free exchange of machine learning experiments is the development of a common language in which to describe them. In Chapter 5, we provide an ontology (a formal, machine interpretable domain model) of machine learning experimentation, called Exposé. It provides a common, unambiguous vocabulary that covers the concepts and relationships inherent to machine learning experimentation, as well as many details of well-known algorithms. Although the ontology is easily extensible and general in intent, we will focus on the task of supervised classification in the remainder of this thesis.

In Chapter 6, we use this ontology to define an XML-based language for describing experiments, called ExpML, so that experiments can be automatically annotated and shared between software agents, such as data mining toolboxes and public experiment databases. We describe the minimal amount of information needed to make experiments fully reproducible and discuss the syntax used to describe experiment components, such as algorithms, datasets and evaluation metrics, as well as exact experimental setups and the obtained results.

Finally, in Chapter 7, we discuss the implementation of the experiment database itself, and more specifically, how to intelligently organize the involved meta-data so that queries can be written about any aspect of the stored experiments, their components, and their outcomes.

See first, think later, then test. But always see first. Otherwise you will only see what you were expecting. Most scientists forget that.

Douglas Adams

Chapter Four

Experiment Databases

All around the globe, thousands of machine learning experiments are being executed on a daily basis, generating a constant stream of empirical information on learning techniques. Unfortunately, they are mostly interpreted with a single focus of interest and discarded afterwards. Yet, the information contained in these experiments might have many uses beyond their original intent and, if properly stored, could be of great use to boost future research. In this chapter, we propose the use of *experiment databases*: databases designed to collect the details of these experiments, and to intelligently organize them in online repositories to enable fast and thorough analysis of a myriad of collected results. They engender a much more dynamic, *collaborative* approach to experimentation, in which experiments can be freely shared, linked together, and immediately reused by researchers all over the world. The use of such public repositories is common practice in many other scientific disciplines, aiming to create an "open scientific culture where as much information as possible is moved out of people's heads and labs, onto the network and into tools that can help us structure and filter the information" (Nielsen 2008).

First, we highlight the drawbacks inherent to contemporary machine learning research in Section 4.1, and show how they can be alleviated through the use of experiment databases in Section 4.2. Next, we discuss similar efforts in other experimental sciences in Section 4.3, draw upon them to design a conceptual framework for experiment databases in Section 4.4.2, and show how these databases engender a more rigorous experimental methodology in Section 4.5.

4.1 Motivation

"Study the past", Confucius said, "if you would divine the future". This surely applies in machine learning as well. Whether we aim to develop new learning algorithms or analyze new pieces of data, it is essential to correctly interpret the results of earlier analysis to gain a deeper understanding of the behavior of prior learning approaches, the effects of their parameters and the utility of data preprocessing.
As discussed in Section 1.3.4, machine learning is - to a large extent - an empirical science, which means that much of this historical information comes in the form of algorithm evaluations. Much like in many other empirical sciences, we collect *empirical evidence* of the behavior of learning algorithms by *observing* them as they run on real-world datasets. If we have a new theory about how algorithms will perform under certain conditions, we *test* that theory by running controlled experiments evaluating those algorithms under exactly those conditions, hopefully discovering empirical laws that drive the field forward. But there are other uses as well: we often want to investigate under which conditions one algorithm outperforms another, how an algorithm parameter should be tuned to fit the given data, or simply whether an algorithm will be fast enough to be used in a certain application. This leads to thousands of empirical studies appearing in the literature. Unfortunately, it is not straightforward to correctly interpret these published results and use them as stepping stones for further research: they often lack the details needed to reproduce them, and it is typically difficult to interpret how generally valid they are.

4.1.1 Reproducibility and Reuse

Indeed, while much care and effort goes into these studies, they are essentially conducted with a single focus of interest and summarize the empirical results accordingly. The individual experiments are usually not made publicly available, thus making it impossible to reuse them for further or broader investigation. Moreover, because of space restrictions imposed on publications, it is often practically infeasible to publish all details of the experimental setup, making it, in turn, very hard for other researchers to reproduce the experiments and verify if the results are interpreted correctly.
This lack of reproducibility has been warned against repeatedly. Hirsh (2008) lists the limited scholarship of machine learning studies as one of the most important challenges highlighted during a panel discussion of the 2007 SIAM International Conference on Data Mining:

> *Unfortunately, we often do not see results documented with the rigor found in other noncomputing experimental sciences. [...] As a scholarly community, we need to be able to document experimental results in sufficient detail to allow reproducibility.*

Pedersen (2008) notes that

> *... as a community we accept that our publications don't provide enough space to describe our elaborate 21st century empirical methods in sufficient detail to allow for re-implementation and reproduction of results. [...] We publish page after page of experimental results where apparently small differences determine the perceived value of the work. In this climate, convenient reproduction of results establishes a vital connection between authors and readers.*

Finally, (Sonnenburg et al. 2007) criticize the lack of open source distributions of the developed learning algorithms:

> *... few machine learning researchers currently publish the software and/or source code associated with their papers (Thimbleby 2003). This contrasts for instance with the practices of the bioinformatics community, where open source software has been the foundation of further research (Stajich and Lapp 2006). The lack of openly available algorithm implementations is a major obstacle to scientific progress in and beyond our community.*

Recently, some conferences have started to require that all submitted research be fully reproducible (Manolescu et al. 2008), adding notices to all publications stating whether or not the results could be verified. Still, there exist no common protocols for making published experiments reproducible, and very few software tools that facilitate this process.

4.1.2 Generalizability and Interpretation

A second issue is that of generalizability: in order to ensure that results are generally valid, the empirical evaluation needs to be equally general, meaning that it must cover many different conditions such as various parameter settings and various kinds of datasets, e.g. differing in size, skewness, noisiness or with or without being preprocessed with basic techniques such as feature selection. Unfortunately, many studies limit themselves to algorithm benchmarking, often exploring only a small set of different conditions, and averaging the results. It has long been recognized that such studies are in fact only 'case studies' (Aha 1992), and should be interpreted with caution.
In fact, Hand (2006) suggests that the many benchmarking studies appearing in the literature, in which each new algorithm seems to outperform prior ones, provide a false sense of progress:

> *...no method will be universally superior to other methods: relative superiority will depend on the type of data used in the comparisons, the particular data sets used, the performance criterion and a host of other factors. Moreover, the relative performance*

> *will depend on the experience the person making the comparison has in using the methods, and this experience may differ between methods: researcher A may find that his favorite method is best, merely because he knows how to squeeze the best performance from this method. [...] an apparent superiority in classification accuracy, obtained in laboratory conditions, may not translate to a superiority in real-world conditions...*

Moreover, a number of studies have illustrated that sometimes, overly general conclusions can be drawn. In time series analysis research, for instance, it has been shown that many studies were biased toward the datasets being used, leading to contradictory results (Keogh and Kasetty 2003). The authors therefore suggest that datasets should be selected that cover the entire spectrum of dataset properties such as size, skewness and the amount of noise. Furthermore, Perlich et al. (2003) describe how the relative performance of logistic regression and decision trees depends strongly on the *size* of dataset samples. Since the vast majority of empirical studies do not take the sample size into account, this raises questions as to what extent the reported results can be generalized. Hoste and Daelemans (2005), in turn, show that in text mining, the relative performance of lazy learning and rule induction is easily dominated by the effect of parameter optimization, data sampling, feature selection, and their interaction.

These studies underline that there are good reasons to thoroughly explore different conditions, or at least to clearly state under which conditions certain conclusions may or may not hold. Otherwise, it is very hard for other researchers to correctly interpret the results.

4.2 Experiment databases

Given the amount of effort invested in empirical assessment, and the importance of correct interpretations thereof, it would be highly useful to have a standardized and automated way of publishing experiments in full detail, including all details on how they were obtained, thus providing a full and fair account of conducted research and a source of unambiguous information on the covered algorithms for further investigation and application. Therefore, we propose the use of *experiment databases*: databases specifically designed to collect all the details on large numbers of past experiments, and to organize them to facilitate their analysis.

The idea of databases that log and automatically organize one's machine learning experiments was first proposed by Blockeel (2006) as an elegant way to remedy the low reproducibility and generalizability of most machine learning experiments. Still, he did not present details on how to construct such a database. In this and the following chapters, we will describe how we can

implement them in practice, and more importantly, how they can be used to engender a global infrastructure for sharing experiments performed by many different researchers, and make them immediately available to everyone.
A key question here is how to automatically organize all experiments so that their results can be optimally reused. To achieve this, every new experiment is broken down to its components (such as the algorithm, parameter settings and dataset used), and its results are related to the exact configuration of those components stored in fine-grained database tables covering every aspect of the experiment setup. It then only takes a query (e.g in SQL) to ask for all results under specific conditions. For instance, requesting the parameter settings of an algorithm and its performance results allows to track the general effect of each parameter. Additionally requesting the dataset size allows to highlight what influence that may have on those parameters. Such queries thus allow to quickly peruse the results under different conditions, reorganizing them at will, thus enabling fast and thorough analysis of large numbers of collected results. The expressiveness of database query languages warrants that many kinds of hypothesis can be tested by writing only one or perhaps a few queries. The returned results can also be interpreted unambiguously, as all conditions under which the returned results are valid are stated in the query itself.

4.2.1 Meta-learning

It may be clear that such large, organized repositories of empirical meta-data serve as a great platform for meta-learning studies. To allow in-depth meta-learning studies such as covered in Chapter 2, it suffices to extend the stored descriptions of datasets and algorithms with the same insightful theoretical meta-features. Experiment databases allow to easily annotate datasets and algorithms with such properties, so that they can be added by any researcher. As such, the myriad of *empirical* results, past and present, are immediately linked to all known *theoretical* properties of algorithms and datasets, providing new grounds for deeper analysis.
As such, it becomes much easier for any researcher to perform advanced meta-learning investigations. For instance, algorithm designers can simply extend their queries to include these theoretical properties and gain precise insights on how their algorithms perform on different kinds of data or how they relate to other algorithms. These insights can then lead to improved algorithm implementations or detailed guidelines on their use. We will see many examples of this capability in Chapter 9. Moreover, practitioners performing exploratory data analysis can also benefit from such resources by doing quick searches for the most interesting approaches (e.g. 'Is this algorithm still fast enough if I feed it large numbers of attributes?') or information on how to use specific algorithms (e.g. 'What parameter seems most important to tune, and what would be a good value range?'). As such, non-expert users of certain algorithms can check which algorithms are worth trying and how they can be tuned.

4.2.2 Overview of benefits

We can summarize the benefits of sharing machine learning experiments and storing them in public databases as follows:

Reproducibility The database stores all details of the experimental setup, thus attaining the scientific goal of truly reproducible research.

Reference All experiments, including algorithms and datasets, are automatically organized in one resource, creating a useful 'map' of all known approaches, their properties, and results on how well they fared on previous problems. As such, we get a detailed overview of how algorithms from many studies perform relative to one another, and many aspects of learning behavior, that may only be known to some experts, can be instantly explored by writing a query. This also includes *negative results*, which usually do not get published in the literature, but may still attribute important information: when designing new algorithms, knowing which ideas did not work is just as valuable as knowing which ones did.

Visibility It adds visibility to (newly) developed algorithms, as they may appear in queries.

Reuse It saves time and energy, as previous experiments can be readily reused. Especially when one wishes to benchmark a new algorithm on commonly used datasets, there is no need to run older algorithms over and over again, as their evaluations are likely to be available online. This would also improve the quality of many algorithm comparisons, because the original authors probably know best how to tune their algorithms, and because one can also easily take the stored dataset properties into account to find out how they affect the relative performance of algorithms, instead of just averaging the results over all datasets.

Generalizability It enables larger and more generalizable studies. Studies covering many algorithms, parameter settings and datasets are hugely expensive to run, but could become much more feasible if a large portion of the necessary experiments are available online. Even if many experiments are missing, one can use the existing experiments to get a first idea, and run additional experiments to fill in the blanks. And even when all the experiments have yet to be run, the automatic storage and organization of experimental results markedly simplifies conducting such large scale experimentation and thorough analysis thereof. During querying it is also immediately clear just how general any observed trends are, since all conditions under which the returned results are valid are stated as constraints in the query itself.

Integration The formalized descriptions of experiments also allow the integration of such databases in data mining tools, for instance, to automatically log and share every experiment in a study or to reuse past experiments to speed up the analysis of new problems.

4.3 Experiment Repositories in e-Sciences

In this section, we survey the implementation and use of experiment repositories in other empirical sciences in order to draw lessons in setting up similar repositories for machine learning.

4.3.1 Surviving the data deluge

The idea of sharing empirical results is certainly not new: it is an intrinsic aspect of many other sciences, especially *e-Sciences*: computationally intensive sciences that generate large volumes of experimental data, and depend on the internet as a global, collaborative workspace to analyze all this data. Examples are fields like genomics, neuroscience, high-energy physics and astrophysics, where gene sequencers, MRI scanners, particle accelerators and earth-based and space telescopes produce massive streams of data of increasingly high resolution. To make all this data accessible to scientists all over the world, it is organized in central resources and presented in common data formats which allow automatic interaction with data analysis tools. Data mining techniques have become an essential part of these e-Sciences: they are crucial to automatically find meaningful patterns in this data deluge (e.g. to detect whether any sky objects have changed between two observations).
It is worth noting that machine learning and data mining themselves are also increasingly driven by large-scale experimentation: clusters of computers are often employed to evaluate new algorithms on benchmark datasets or to execute extensive exploratory evaluations on new collections of data to arrive at the best possible model. It thus seems logical to also collect these massive streams of experiments in public databases so they can be analyzed and reused by researchers all over the world, and especially to use data mining techniques on this meta-data to engender the same automatic discovery of insightful patterns. In the remainder of this section, we cover the fields where these repositories are most evolved: bio-informatics, astrophysics and high-energy physics.

4.3.2 Bioinformatics

4.3.2.1 Microarray Databases

Probably the best known example in bio-informatics are *microarray databases*, specifically *DNA-microarrays*: high-throughput screening experiments aimed at pinpointing the functions of individual genes.
Some background: every living cell contains a large set of genes. However, in each type of cell, only a fraction of these genes are *expressed*, resulting in unique properties. A cell carries out its functions through *gene expression*, in which genes are copied and converted into specialized molecules. Most genes encode proteins, in which case the information from a gene is first *transcribed*

into messenger RNA (mRNA), in turn translating into proteins. Studying the kinds and amounts of mRNA produced by a cell teaches us something about which genes are turned on and thus about the *function* of those genes.
A *DNA-microarray* is a small membrane or glass slide containing samples of many different known genes (actually single strands of those genes called *DNA templates*). They exploit the ability of a given mRNA molecule to bind, or *hybridize*, to the DNA template from which it originated. Scientists isolate mRNA from each cell type, generate the complementary molecule (cDNA), and attach a fluorescent tag. By incubating a microarray with a mix of two differently tagged samples (usually a target and a control) and exciting the fluorescent tags with a laser, we can determine which and how much mRNA is bound to each site on the array by recording the resulting color (red, green or yellow if both samples hybridize) and intensity. As such, we can measure the *expression levels* of hundreds or thousands of genes in a single experiment. Figure 4.1 shows the results of such an experiment.
Microarray experiments provide much more information than can be analyzed by a single study. The need for reproducibility, as well as recognition of the potential value of microarray results beyond the summarized descriptions found in most papers, have led to the creation of public repositories of microarray data, such as ArrayExpress (Brazma et al. 2003), and submitting experiments to these repositories has become a condition for publication in over 50 journals (Ball et al. 2004).

4.3.2.2 Three supporting initiatives

These repositories are supported by three complementary initiatives.
Minimal information The first is a set of guidelines stating the Minimum Information About a Microarray Experiment (MIAME) (Brazma et al. 2001). It defines the information that is required to permit another researcher to understand the experiment and the data (Stoeckert et al. 2002). They include the makeup of the microarray, details of the sample and any treatments it underwent, and other information such as who did the experiment and which scanners were used. This information is stored in a local, MIAME-compliant database, which may store other details as well.
Standard language The second is a standard, formal language to exchange experiments between researchers: the MicroArray Gene Expression language (MAGE) (Spellman et al. 2002) defines both an object model used to design databases (MAGE-OM) and an XML-based standard for data exchange (MAGE-ML) (Stoeckert et al. 2002), with complementary software to load it into databases. If researchers want to reuse data from other studies, they search public repositories, download the MAGE-ML documents and import them into their local database. On publication, the researcher indicates exactly which hybridizations are associated with the paper and exports all data as MAGE-ML documents to be uploaded to a public repository.

Ontologies The third component is an ontology to describe biological samples and their manipulations. This is necessary to remove ambiguities and to facilitate automatic processing. For instance, when indicating that the cell under study is from a certain type of mouse, a human will be able to tell if one species is a subspecies of another, but a computer won't. Instead of building an all-encompassing ontology, the MGED Ontology[1] focuses on microarray-related concepts that reflects the MIAME guidelines and MAGE structure. It can be extended further by existing ontologies for different subfields, such as the Gene Ontology (Ashburner et al. 2000) of genes and their products, or the National Center for Biotechnology Information (NCBI) taxonomy for organisms. However, with many sometimes competing ontologies, some harmonization is needed. The Open Biomedical Ontology (OBO) Foundry[2] tries to define a set of ontologies that are freely available, use a consistent syntax, and that complement (rather than compete with) other ontologies of that set.
Several tools have arisen that use these standards to allow the exploration of large bodies of gene expression data. GeneExplorer[3], for instance, is shown in Figure 4.2. Each row corresponds to a gene on the microarray, each column corresponds to a different sample, e.g. from different cells, and the colors indicate the expression levels of all the genes: between 8 times less than the mean (green) and 8 times greater than the mean (red). It also uses hierarchical clustering to group samples based on similarity in their expression levels.

4.3.2.3 Similar standards

Their success has instigated similar approaches in related fields, such as the MIAPE guidelines for experiments in proteomics (the large-scale study of proteins), which are stored in several repositories based on the experimental technique used, such as PRIDE for mass spectrometry data (Vizcaino et al. 2009).

4.3.2.4 The robot scientist

One remaining drawback is that experiment description is still partially performed manually. Still, some projects are automating the process further. The Robot Scientist (King et al. 2009; Soldatova et al. 2006) is a fully automated scientific discovery system trying to identify the functions of genes in yeast cells. It has ontologies and standards for all the physical aspects of experiment execution, such as the locations of all the instruments and the positions of wells on a microplate. Moreover, it uses machine learning to generate hypotheses about the functions of genes and builds ontological descriptions about what has been learned from past experiments. It has autonomously made several novel scientific discoveries.

[1] http://www.mged.org/ontology
[2] http://www.obofoundry.org/
[3] http://gmod.org/wiki/GeneXplorer

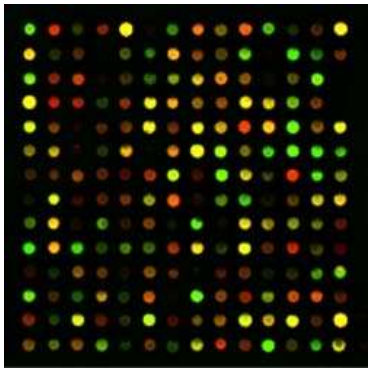


Figure 4.1: A hybridized DNA-microarray.

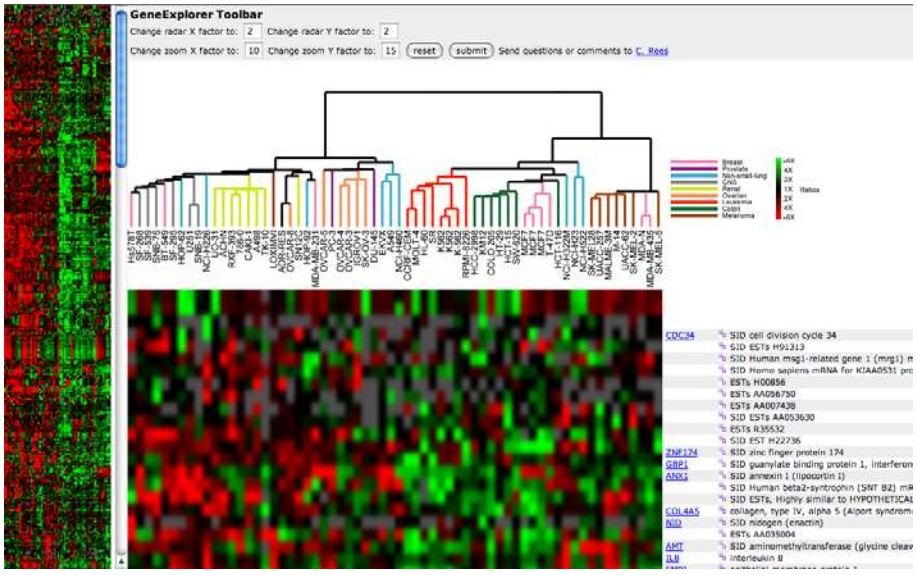


Figure 4.2: GeneExplorer

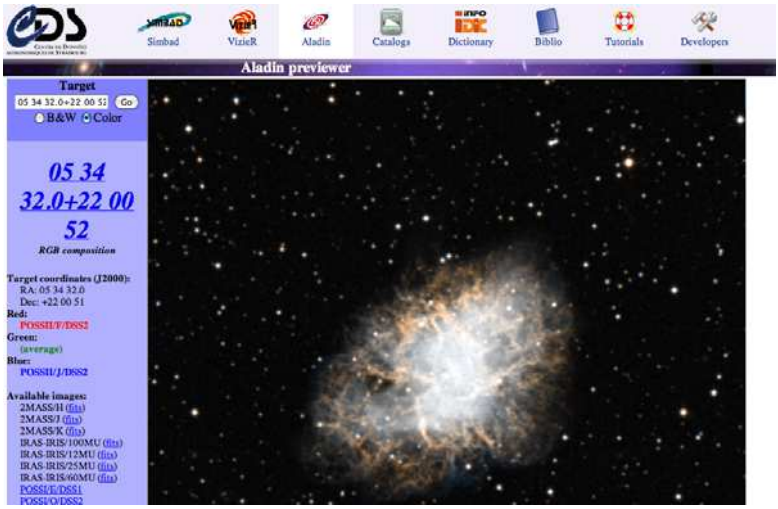


Figure 4.3: A query on the ALADIN interactive sky atlas.

4.3.3 Astrophysics

While little more than a decade ago, astrophysical observations used to be kept on private servers at observatories, a concerted effort in providing improved access to this data has resulted in the creation of *Virtual Observatories* (Szalay and Gray 2001), combining the astronomical data from different telescopes. Specialized software systems are used to allow transparent access to these heterogeneous data collections. For instance, Figure 4.3 shows the interface of the ALADIN interactive sky atlas[4] (Bonnarel et al. 2000): simply entering a coordinate (in time *and* space!) immediately returns a layered image of the sky at that coordinate, with many options to delve deeper into the data. Since many instruments are being used for systematic surveys of the universe, this provides astronomers with an unprecedented catalog - a World-Wide Telescope - to study the evolving universe. They are supported by an extensive list of different protocols driving automation, such as an XML format for tabular information (VOTable) (Ochsenbein et al. 2004) and astronomical binary data (FITS), an Astronomical Data Query Language (ADQL) (Yasuda et al. 2004) and informal ontologies (Derriere et al. 2006) called Unified Content Descriptors (UCDs). The data is stored in databases all over the world and is queried for by a variety of portals (Schaaff 2007). It is now seen as indispensable to analyze the constant flood of data. In addition, all the astronomy literature is online and is cross-indexed with the observations[5].

[4] http://aladin.u-strasbg.fr/aladin.gml
[5] http://simbad.u-strasbg.fr/simbad/

4.3.4 Physics

Various subfields of physics also share their experimental results. Low-energy nuclear reaction data can be expressed using the Evaluated Nuclear Data File (ENDF) format and collected into searchable ENDF libraries.[6] In high-energy particle physics, the HEPDATA[7] website scans the literature and downloads the experimental details directly from the machines performing the experiments. Finally, XML-formats and databases have been proposed for high-energy nuclear physics as well (Brown et al. 2007).

4.4 Designing Experiment Databases

We will now summarize what we have learned from these sciences and use this to build similar collaborative infrastructures for machine learning.

4.4.1 Lessons learned

First and foremost, all these sciences seem to have evolved towards online, public infrastructures for experiment exchange, using more or less the same three components:

A formal representation language: to enable a free exchange of experimental data, a standard and formal representation language needs to be agreed upon. Such a language may also contain guidelines about the information necessary to ensure reproducibility.

Ontologies: defining a coherent and unambiguous description language is not straightforward. It generally requires a careful analysis of the concepts used within a domain and their relationships. This is formally represented in *ontologies* (Chandrasekaran and Josephson 1999): machine manipulable models of the domain providing a controlled vocabulary in which the interpretation of each concept is clearly described.

A searchable repository: to reuse experimental data, we need to locate it first. Experiment repositories therefore still need to organize all data to make it easily retrievable.

Still, the context in which machine learning experiments are typically executed offers some advantages which we can exploit. First, compared to the *in vitro* experiments in bioinformatics, the *in silico* experiments in machine learning should be much easier to keep track of, perhaps even automated completely. Indeed, microarray databases, for all their benefits, are sometimes criticized for imposing too much 'administrative' work (Stoeckert et al. 2002). In machine learning, a common software interface could be developed, in different

[6] http://www.nndc.bnl.gov/exfor/endf00.jsp
[7] http://durpdg.dur.ac.uk/hepdata/

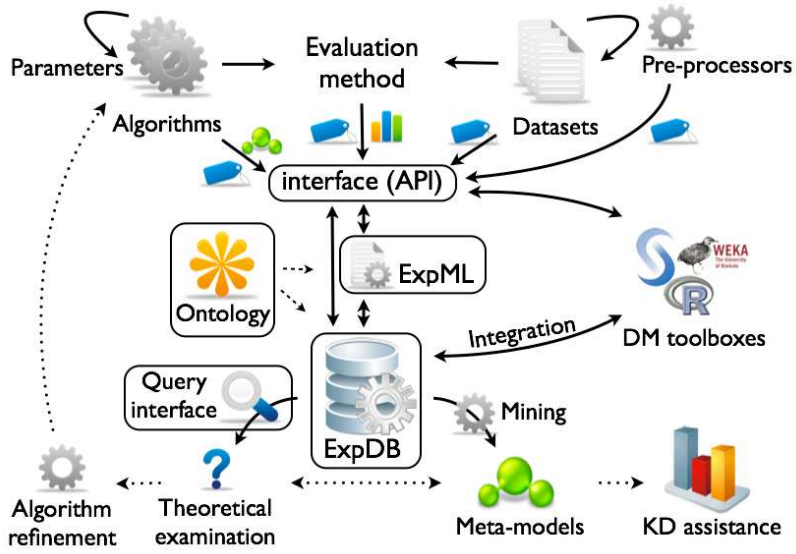


Figure 4.4: The architecture of experiment databases.

programming languages, that uniformly represents the performed experiments and exports them in the formal representation language mentioned above. As such, experiments could be shared at the click of a button.

Second, in contrast to scientific equipment, we can store algorithms and other pieces of code into the database as well. As such, all information necessary to reproduce the stored experiments can be found easily, and algorithms and scripts will be more easily reused and improved. As discussed in Sonnenburg et al. (2007), there are many benefits to publishing even preliminary versions of algorithms: it will be much easier for authors to get feedback, others may even fix some remaining bugs or extend the algorithm, and good implementations will often be reused (and cited). The stored algorithms should be cross-linked with existing repositories for machine learning algorithms[8].

Finally, while different kinds of machine learning experiments exist, most of them have an inherent structure that we can use to organize the experiments. We can thus go further than storing experiments 'per study' as is currently done in bioinformatics. While there is no real machine learning equivalent to the celestial coordinate system used in astrophysics, one can imagine the space of possible experiments that can be executed using a certain number of algorithms and a certain number of datasets. As such, we can organize experiments in this space to relate them to each other and make them easily retrievable.

[8] http://mloss.org

4.4.2 Conceptual Framework

These considerations lead to the conceptual framework for collaborative experimentation shown in Figure 4.4. Note that it also implements our framework for meta-learning shown in Figure 2.5. It consists of the following components, which will be covered in detail in the subsequent chapters of this text:

Interface First, to facilitate the automatic exchange of machine learning experiments, an application programming interface (API) should be provided, see Chapter 8, that builds uniform experiment instances out of all details of the experimental setup (indicated by tag symbols), and the obtained results. This API can then be used by software agents, e.g. data mining workbenches (shown on the right in Figure 4.4) or custom algorithm implementations, to automatically exchange experiments with experiment databases or other software agents.

ExpML To promote the free exchange of experiments, a common language should be introduced for describing experiments in machine learning, let's call it ExpML. As such, experiments can be streamed between software agents. This language should be very flexible, and make sure that all details necessary for reproducibility are provided. We will introduce such a language in Chapter 6.

Ontology The vocabulary and structure of ExpML files is provided by an ontology of machine learning experimentation. It provides a formal domain model that can be adapted and extended on a conceptual level, thus fostering collaboration between many researchers. Any conceptual extensions to the domain model can then be translated into updated or new description languages and database models for specific types of experiments. It will be described in Chapter 5.

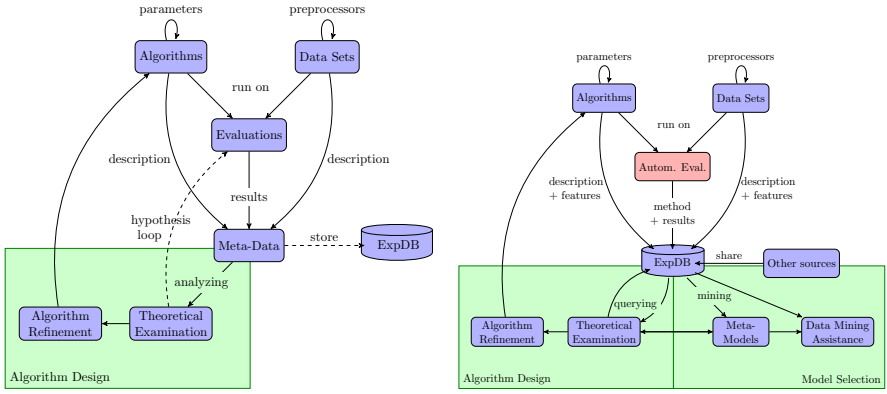
ExpDB Experiment databases (ExpDBs) collect experiments and organize all the contained information. ExpDBs can be setup locally, e.g. for a single person or a single lab, or it can be a public one open to submissions by many different researchers. Their design will be described in Chapter 7.

Query interfaces The data stored in ExpDBs is accessed through query interfaces. We will describe several such interfaces in Chapter 8.

The bottom of the figure shows different ways to tap into the stored information, which will be illustrated in detail in Chapter 9:

Querying allows a researcher to formulate questions about the stored experiments, and immediately get all results of interest. Such queries could be aimed at discovering ways in which an algorithm can be improved (e.g. see Section 9.1.2), thus completing the algorithm development cycle.

Mining A second use is to automatically look for patterns in algorithm performance by mining the stored meta-data. The insights provided by such *meta-models* can then be used to design better algorithms or to select and apply them in knowledge discovery applications (Brazdil et al. 2009).



(a) Current (b) New

Figure 4.5: Experimental methodologies in machine learning

Integration Data mining toolboxes could also interface with ExpDBs directly, for instance to download the results of experiments that have been run before by a different user of that toolbox. Conversely, they could also automatically export all performed experiments to an ExpDB to analyze the results more easily, to allow reuse, or simply to keep a detailed log of all performed experimentation.

4.5 Using Experiment Databases

There are many ways in which these experiment databases could be used in machine learning research. In this section, we describe how their application can solve the issues of reproducibility, generalizability and interpretability highlighted in Section 4.1.

4.5.1 Personal logs

In their simplest form, they serve as personal logs that automatically keep track of performed research, ensuring that researchers can reproduce their own experiments at a later point in time, and shared upon publication. This situation is depicted in Figure 4.5(a). Any question we may have about the behavior of learning algorithms needs to be answered by setting up new experiments. It is a laborious procedure in which we need to collect datasets, algorithms and evaluation procedures, and set up experiments specifically to test each new hypothesis. Moreover, the problem of generality remains: each experiment typically occurs under many constraints (e.g. default parameter settings), as measuring each possible effect is very labor-intensive.

4.5.2 An improved methodology

However, Figure 4.5(b) illustrates how experiment databases can also enable an improved methodology, first proposed by Blockeel (2006). Instead of designing experiments to test a specific hypothesis, one can design them to cover, as well as possible, the space created by all the variables of the problem, and store all these results in a database. Several strategies can be applied to select a range of suitable experiments (see Section 4.5.4), which could then be run and submitted automatically to an experiment database to organize the results.

One hypothesis after another can then be tested simply by querying the database for the relevant experiments and interpreting the returned results. While in this methodology, more experiments are needed to be able to evaluate the learning algorithms under a variety of conditions, the results will be much more general. Indeed, we know exactly how generalizable the results are, since all constraints are listed explicitly in the query. We can thus be confident that we interpreted the results correctly.

An added benefit is that, when testing algorithms under many different circumstances, there is a large probability that the results will also be of interest to other researchers working with the same kind of methods and vice versa: a portion of the necessary experiments may already have been run before, in which case we can just reuse them. Sharing experiments in an organized fashion boosts the value of experimental results: due to the network effect created by several researchers contributing their findings on various learning methods, queries on this meta-data will paint an increasingly detailed picture of learning performance, and less and less experimentation will be needed to compare new algorithms to prior ones, meaning that algorithm designers can focus all their time and resources on refining their algorithms, and that practitioners get a large amount of meta-data to guide their search for the best algorithms for future problems.

4.5.3 An illustration

To further illustrate the difference between the two approaches, say Ann, using the first methodology in Figure 4.5(a), wants to test the effect of dataset size on the complexity of trees learned by C4.5, a decision tree-based learning algorithm. To do this, she selects a number of datasets of varying sizes, runs C4.5 (with default parameters) on those datasets, and interprets the results. Bob, a proponent of the new methodology proposed here (and shown in Figure 4.5(b)), would instead build a large database of C4.5 runs (with various parameter settings) on a large number of datasets, possibly reusing a large amount of experiments already existing in experiment databases. To perform these experiments, he could use DM tools that automatically generate a large array of experiments according to his specifications (e.g. the experimenter component in WEKA), and store the ensuing results directly into an ExpDB. Bob then

queries the database for C4.5 runs, selecting the dataset size and tree size for all runs with default parameter settings (explicitly mentioning this condition in his query), and plotting them against each other. If Ann wants to test whether her results on default settings for C4.5 are representative for C4.5 in general, she needs to set up new experiments. Bob, on the other hand, only has to write a second query, this time not including the condition. This way, he can easily investigate under which conditions a certain effect will occur and be more confident about the generality of his results. Moreover, when there seem to be complex interactions between all of C4.5's parameter settings, he could simply use all the available experiments to automatically build a model that describes these interactions, as will be illustrated in Section 9.2.3.2.

The initial investment with respect to experimentation will likely pay off in the long run, especially since many hypotheses can easily be tested, returned results are more general, and more experiments can be reused. For instance, say another researcher is interested in the runtime (or another performance metric) of C4.5 on these experiments: since this is recorded in the experiment database as well, these experiments will not have to be repeated and can be reused for free.

4.5.4 Designing generalizable experiments

How do we design experiments so that they cover a wide range of different conditions? First, we select the algorithm(s) of interest from a large set of available algorithms. To choose their parameter settings, one can specify a probability distribution for each different parameter according to which values should be generated. In the simplest case, this could be a uniformly sampled list of reasonable values. Covering the dataset space can be done identically to the selection of datasets in meta-learning, covered in Section 2.6.

These algorithms, parameter settings and datasets create the space of possible experiments. To ensure that their final results are generalizable, we need to cover this space as well as possible, but also populate it in a reasonably dense way to be able to ask very specific queries, e.g. showing the effect of a specific algorithm parameter on specific kinds of datasets.

A straightforward way to do this is a fine-grained grid search, but given that the experiment space can be very high-dimensional, this may not always be feasible. One could also look at techniques from active learning or Optimal Experiment Design (OED) (Cohn 1996) to focus on the most interesting experiments given the outcome of previous experiments. Another simple, yet effective way is selecting random values for all variables of our experiments. To imagine how many experiments would be needed in this case, assume that each of these variables has on average v values (numerical parameters variables are discretized into v bins). Running $100v$ experiments with random values for all parameters implies that for each value of any single parameter, the average outcomes of about 100 experimental runs will be stored. This seems sufficient

to be able to detect most correlations between outcomes and the value of this parameter. To detect n-th order interaction effects between parameters, $100v^n$ experiments would be needed. Taking, for example, $v = 20$ and $n = 2$ or $n = 3$, this yields respectively 40,000 and 800,000 experiments, a large number, but (especially for fast algorithms) not infeasible with today's computation power. Note how this contrasts to the number of experimental runs typically reported on machine learning papers. Still, the factor 100 is the price we pay for ensuring reusability and generalizability. The more people run and share generalized experiments, the more experiments we can reuse, making generalizability practically free in the end. The v^n factor is unavoidable if one wants to investigate n^{th} order interaction effects between parameters. Most existing work does not even study effects higher than the second order.

4.6 Conclusion

In this chapter, we have described how the use of experiment databases can boost machine learning research. Experiment databases are databases specifically designed to collect all the details on large numbers of past experiments, performed by many different researchers, and to make them immediately available to researchers all over the world. They use the world wide web as a global, user-driven collaborative workspace in which researchers can freely exchange experimental results. Often, there are many more uses of such data than can be covered in single studies, and by combining data from different sources, the resulting pool of data enables studies that were infeasible before.

We started off by outlining the main benefits these ExpDBs would bring to machine learning research. To learn from previous practical applications of such databases, we looked to so-called e-Sciences, where they have, in some form or another, been applied successfully for some time now, particularly in bioinformatics, astrophysics and high-energy physics. From this, we learned that they all seem to have evolved towards online, public infrastructures for experiment exchange, using more or less the same three components: formal experiment description languages, ontologies, and searchable repositories.

In machine learning, we can improve further on these systems by automating experiment submission, storing the algorithms and datasets as well, and using the inherent structure of machine learning experiments to relate them to each other in an homogenous way and make specific results more easily retrievable. These considerations led to a conceptual framework for collaborative experimentation which forms the backbone for the remainder of this thesis.

Finally, we show how ExpDBs can form an integral component of a collaborative, much more dynamic and rigorous experimental methodology for machine learning research, in which many questions about learning algorithms could be answered on the fly, and which solves three important issues with contemporary studies: reproducibility, generalizability and interpretability.

> Language is a city to the building of which every human being brought a stone.
>
> *Ralph Waldo Emerson*

Chapter Five

The Exposé Ontology

Sharing machine learning experiments with each other starts with speaking the same language. Moreover, if we want to automatically organize thousands of experiments generated by various researchers all over the world, this language should be interpretable by machines. Designing a coherent and unambiguous formal language is not straightforward, especially since experimentation in machine learning is a very involved process including various statistical techniques and many different setups. Indeed, a very fine-grained description will be needed if we wish to answer questions about detailed aspects of the involved learning algorithms and datasets.

In this chapter, we establish an ontology, called Exposé, in which the concepts used within machine learning experiments and their relationships are carefully described and expressed in a formal domain model. It provides a common, unambiguous vocabulary for researchers wishing to describe their experiments, and it explicates the inherent structure of machine learning experiments. This will be instrumental to develop valid ways to express experiments in formal representation languages (see Chapter 6) and to store them in an organized fashion (see Chapter 7).

While machine learning and data mining are extensive and creative sciences, not easily captured in a domain model, researchers do share common experimental procedures which we can express. Also, while learning algorithms are complex, composite objects, we can elucidate their structure given a sufficiently flexible domain model and a sufficiently extensive vocabulary.

The field of ontology engineering (Calero et al. 2006) provides techniques for designing such models, especially in collaboration with many researchers. Indeed, ontologies are built to evolve: they can be modified, extended and refined to cover ever more types of experiments, tasks and algorithms. There are thus a logical choice for the principled design of community-based experiment databases.
In a way, we start small. We will focus on supervised classification and our experiments are limited to algorithm evaluations on static, propositional datasets. Still, this already covers a decent amount of contemporary experimentation and includes many concepts common to other subfields.
We start with explaining what ontologies are and why we need them in Section 5.1. Next, in Section 5.2 we discuss various previously proposed ontologies for data mining. Finally, in Section 5.3, we introduce our Exposé ontology for describing machine learning experimentation, starting with the top-level concepts, and consecutively highlighting the way we represent experiments, experimental setups, model evaluation measures, algorithm performance estimation techniques, datasets, algorithms and their components, and internal learning mechanisms. We will also focus on some of the concepts, algorithms and techniques which will emerge in Chapter 9 when we explore the data stored in our experiment database. Section 5.4 concludes.

5.1 Introduction

5.1.1 Definition

An *ontology* is defined as a "formal, explicit specification of a shared conceptualization" (Studer et al. 1998), in which a *conceptualization* is an abstract model of some phenomenon in terms of relevant concepts. It is used to capture *knowledge* about some domain of interest: it describes the *concepts* in the domain and also the *relationships* that hold between those concepts (Horridge et al. 2009). In the context of this work, they can be described more specifically as formal *representation vocabularies* (Chandrasekaran and Josephson 1999), meaning that they provide an unambiguous vocabulary for a certain domain based on logical statements. The fact that the stored knowledge is formally expressed means it can be interpreted and manipulated by machines. There exist various languages for describing ontologies. In this work, we will use the OWL language (an anagram of Web Ontology Language), which is an extension of the XML-based RDF (Resource Description Framework) format allowing greater interpretability. There also exist different OWL sublanguages with various degrees of expressiveness. Here, we'll use OWL-DL, which is based on Description Logic (Baader et al. 2005), a formal knowledge representation language for which practical logical *reasoning algorithms* exist to reason about the stored knowledge (Horridge et al. 2009).

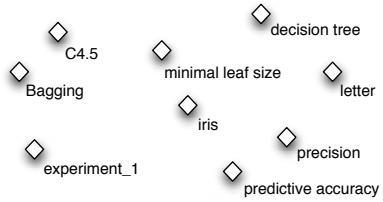


Figure 5.1: Representation of individuals.

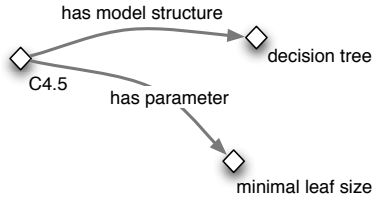


Figure 5.2: Representation of properties.

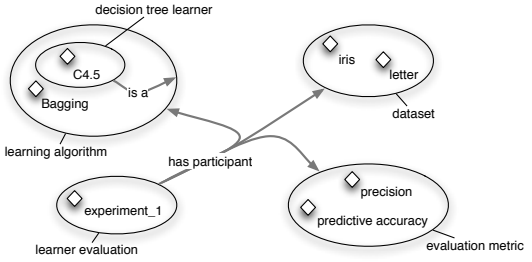


Figure 5.3: Representation of classes (containing individuals).

5.1.2 Example

The best way to explain what ontologies are and what they look like is to give an example. To express knowledge formally, the OWL language allows to define *classes* of *individuals*, *properties* of individuals, and *operations* on classes:

Individuals represent objects in the domain of interest. In this work, this could be the objects shown in Figure 5.1, such as C4.5 (a learning algorithm), letter (a dataset) and precision (an evaluation metric).

Properties are binary relations between individuals: they link the two individuals together. As shown in Figure 5.2, this could include the property 'has parameter' which links the individual 'minimal leaf size' to the individual 'C4.5' algorithm, or that 'C4.5' uses a 'decision tree' as its representational model. Properties can also be transitive or symmetric.

Classes are *sets* that contain individuals, as shown in Figure 5.3. They are described using formal descriptions that state all the requirements for an object to be a member of that class. In Figure 5.3, it is stated that a learner evaluation always has the indicated three participants.

Operations are operations on classes such as taking the union, intersection and complement that can be used in logical descriptions.

Classes are typically organized into hierarchies of subclasses (with the 'is-a' relationship), also known as taxonomies. Subclasses specialize (are *subsumed* by) their superclasses. For instance, in Figure 5.3, C4.5 belongs to the class 'decision tree learner', which is a subclass of 'learning algorithm'. This means that being a decision tree learner implies that you are a learning algorithm, and thus inherit all properties of learning algorithms. In OWL-DL, hierarchical relationships between classes can be computed automatically in a reasoner. Say we define a 'decision tree learner' as being a 'learning algorithm' that uses a 'decision tree' as its model, then all algorithms that use some kind of decision tree model will automatically be regarded as decision tree learners. Such automatic inference (or 'classification') is especially useful to keep large ontologies maintainable and avoid human errors. It also improves the reuse of ontologies in different applications, where different hierarchies may be of interest than the one defined manually.

5.1.3 Why do we need ontologies?

There are many good reasons to develop an ontology for experimentation in data mining.

An unambiguous vocabulary A first reason is that such an ontology provides us with an exact and formal domain model that we can use to design consistent and extensible experiment description languages and database models. For instance, one could simply state that an experiment evaluates a learning algorithm on a dataset and reports the evaluation results according to some metric, but this is highly ambiguous. First, there are many kinds of experiments in machine learning, so we'll need to be more explicit. Second, the term 'learning algorithm' is also ambiguous: is it an abstract algorithm or an implementation? Moreover, it can have parameter settings that we need to take into account. Then there are ensemble algorithms depend on other learning algorithms (base-learners): an algorithm can thus be the main learner in one experiment and only a base-learner in the next. How do we measure the effect of the ensemble technique? Datasets can also be preprocessed to various degrees and they can be split into training and test data in many different ways.
If we want to develop experiment databases which are to collect and relate the results of different studies, we need to provide a clear terminology that captures the complex structure of learning algorithms and their evaluation. Otherwise, the results of any two researchers will likely be incomparable.

A basis for collaboration A second reason is that the development of languages and databases for experiment exchange is essentially a community affair. Data mining and machine learning are extensive and ever-expanding fields, so it should be easy for anyone to suggest changes or extensions to keep these infrastructures in sync with current research. An ontology offers a conceptual basis

for such collaboration involving many researchers: it allows to modify, extend and refine our domain model on a conceptual level, after which these extensions or modifications can be translated to updated representation languages (i.e. XML elements and syntax) and database models (i.e. tables, fields, data types and constraints), as shall be discussed in the following chapters. Related to this is the ability to reuse previously defined knowledge. If the ontology is to be extended to cover experiments in a new task, it can likely reuse many of the concepts already defined for other tasks.

Intelligent software integration A third reason is that ontologies can be automatically queried by software agents (Sirin and Parsia 2007). As discussed in Chapter 8, we plan to use this aspect in future work to build more flexible, user-friendly query interfaces. Moreover, as mentioned in 3.7.4.3, learning algorithms are increasingly being reimplemented as services available on the web. Ontologies are central to this development, since they offer formal descriptions of the algorithms and datasets that can be interpreted by other software agents. As such, ontologies support the automatic exchange of empirical evaluations between web services and online experiment repositories or any other tool that is interested in this information.

Improved meta-learning As noted in Section 2.10, the only fundamental reason why a learning algorithm performs well on a dataset is because its bias matches the data, which led to the conclusion that we need better ways of describing the bias of learning algorithms. Indeed, almost all meta-learning studies handle learning algorithms as black boxes, without taking the internal components of learning algorithms into account. A detailed ontology of learning algorithms can help "pry open these black boxes to sort out salient algorithm features such as the structure and parameters of the models built, the data partitions effected in data space, the cost function used and the optimization strategy adopted to minimize this cost function" (Hilario et al. 2009). Linking the performance results of many algorithms to the internal components which define their bias could provide important new insights.

The unification of machine learning As stated in Goble et al. (2006), scientific progress increasingly depends on pooling know-how and results, making connections between ideas, people and data, and finding and reusing knowledge and resources generated by others in perhaps unintended ways. It is about harvesting and harnessing the collective intelligence of the scientific community. In this context, ontologies define a common vocabulary for people in different machine learning subfields, so ideas and results will spread more easily. Work on ontologies could one day lead to a generally accepted framework to unify the area of data mining (Dzeroski 2007), seen as one of the most challenging open problems in data mining research today (Yang and Wu 2006).

5.2 Previous work

For all these reasons, the design of ontologies for machine learning received quite a bit of attention in recent years, and many ontologies have been proposed for various goals. We can categorize them in two classes: deep or heavyweight ontologies and lightweight ontologies.

5.2.1 Heavyweight ontologies

Heavyweight ontologies are built *top-down*. They start from predefined top-level concepts and relationships. They then try to build new ontologies using those top-level concepts, making sure that no descriptions are violated. It is harder to build such ontologies because they require a deep understanding of the involved concepts. Their benefits are that they integrate easily with other sciences (thus fostering collaboration), that concepts are completely unambiguous, and that more powerful computational inference techniques can be used based on predefined relationships.

First, **EXPO** (Soldatova and King 2006) is a top-level ontology that models scientific experiments in general. It formalizes experimental principles so that empirical research can be uniformly expressed and automated as much as possible. It covers concepts such as hypotheses, (un)controlled variables, experimental designs and experimental equipment, to name a few.

OntoDM (Panov et al. 2009) is a general ontology for data mining with the aim of providing a unified framework for data mining research. It attempts to cover the full width of data mining research, containing high-level concepts, such as data mining tasks and algorithms, and more specific concepts related to certain subfields, such as constraints for constraint-based data mining.

5.2.2 Lightweight ontologies

In contrast, lightweight ontologies are built *bottom up* with a specific application in mind. Rather than establishing universal facts, they simply represent knowledge in a formal way so it can be queried. They are more practical to build and use, but are typically not generally valid outside of the context of the application for which they were designed. They don't integrate as nicely with other ontologies, but often, a loose coupling can still be very useful.

DAMON (DAta Mining ONtology) (Cannataro and Comito 2003), one of the first ontologies for data mining, was meant to offer domain experts a taxonomy for looking up tasks, methods and software tools given a certain goal.

KDDONTO (Diamantini et al. 2009) is a more formal (OWL-DL) ontology with the same goal: to discover suitable KD algorithms and to express compositions of KD processes. It covers the inputs (dataset, parameters) and outputs (models) of the algorithms and any pre- and postconditions for their use.

KD ontology (Záková et al. 2008) describes planning-related information about datasets and KD algorithms. It is used in conjunction with an AI planning algorithm: pre- and postconditions of all KD operators are extracted from the ontology and converted into a standard PDDL planning problem so it can be solved by the planner (Klusch et al. 2005). It is used in an extension of the Orange toolkit (Demšar et al. 2004) to automatically plan and offer the best KD workflow (Záková et al. 2009).
The **eProPlan** ontology (Kietz et al. 2009) also describes all KD operators with their in/outputs and pre/postconditions, but is meant to be used in a *collaborative* KD support system, under development, that generates (partial) workflows, checks and repairs workflows built by users, retrieves workflows previously built by other users, and provides explanations with its workflows.
DMOP, the Data Mining Ontology for Workflow Optimization (Hilario et al. 2009), models the internal structure of learning algorithms, and is explicitly designed to support algorithm selection. It covers concepts such as the structure and parameters of predictive models, the involved cost functions and optimization strategies. It is meant to be integrated with the eProPlan ontology to support the generation of optimal KD workflows.

5.3 The Exposé Ontology

We now turn to the description of our ontology for machine learning experimentation, called Exposé. Next to introducing the necessary vocabulary and structure for the ExpML representation language and the database described in the following two chapters, we will also pay extra attention to a number of learning algorithms and techniques which will be used and discussed in the meta-learning investigations performed in Chapter 9. The entire ontology currently defines over 850 concepts concerning the structure of the experimentation process, as well as many different dataset and algorithm characteristics, experimental design methods, evaluation functions and algorithm descriptions.
It is important to note that this is a straw-man proposal that is intended to instigate discussion. Ontologies have varying levels of maturity, from ontologies proposed by individuals, over pre-standard versions accepted by a group of people, to generally accepted ontologies used by many. Our ontology is still fairly young. It has been influenced and adapted by quite a few people, mostly through close collaboration with the authors of the other ontologies mentioned in the previous section, but is by no means generally accepted yet. One of the biggest caveats is that the current ontology, while general in intent, focuses on predictive classification, only one of several machine learning tasks. The ensuing representation language and database will thus be limited in the same way.
In designing Exposé, we paid close attention to existing guidelines for ontology design (Noy and McGuinness 2002; Karapiperis and Apostolou 2006):

Top-level ontologies It is considered good practice to start from generally accepted and unambiguously described concepts and relationships (Panov et al. 2009). We started from the Basic Formal Ontology (BFO)[1] covering top-level scientific concepts and the OBO Relational Ontology (RO)[2] offering a predefined set of relationships.

Ontology reuse If possible, (parts of) other ontologies should be reused to build on the knowledge (and the consensus) expressed in those ontologies. When designing Exposé, we reused general machine learning concepts from the OntoDM ontology (Panov et al. 2008), experimentation-related concepts from the EXPO ontology (Soldatova and King 2006), and concepts related to internal algorithm mechanisms from the DMOP ontology (Hilario et al. 2009). In fact, Exposé bridges the gap between the very specific concepts of DMOP and the very general ones of OntoDM, thus providing an important contribution to the harmonization of various data mining ontologies. Any future extensions of any of these other ontologies can directly be used to update Exposé, and vice-versa.

Design patterns Ontology design patterns[3] are reusable, successful solutions to recurrent modeling problems. For instance, we mentioned that a learning algorithm can act as an individual learner in one experiment, and as a base-learner for an ensemble learner in the next. This is a case of an agent-role pattern, in which an agent (algorithm) only plays a certain role in a process in some occasions, but not always. A predefined relationship, 'realizes', is used to indicate that a individual is able to fulfill a certain role. We have used such patterns as often as we could.

Quality criteria General criteria include *clarity* (descriptions of the concepts should make the meaning of each concept clear), *coherence* or *consistency* (there should be no logical contradictions), *extensibility* (future uses should be anticipated) and *minimal commitment* (only support the intended knowledge sharing activities). These criteria are rather qualitative, and were only evaluated through discussions with other researchers.

Finally, many Exposé concepts were extracted from earlier working versions of our experiment database, thus providing concepts and relationships that proved practically useful to organize experimental information. Vice versa, many limitations of those earlier versions were solved through Exposé's much more principled design, and the range of questions that can be answered through querying in the current version will be evaluated in Chapter 9.

The complete ontology can be downloaded from the experiment database website (`http://expdb.cs.kuleuven.be`), and explored and edited using any OWL-DL editor, e.g. the Protégé editor (v4)[4] we used to build Exposé.

[1] http://www.ifomis.org/bfo
[2] http://www.obofoundry.org/ro/
[3] http://ontologydesignpatterns.org
[4] http://protege.stanford.edu/

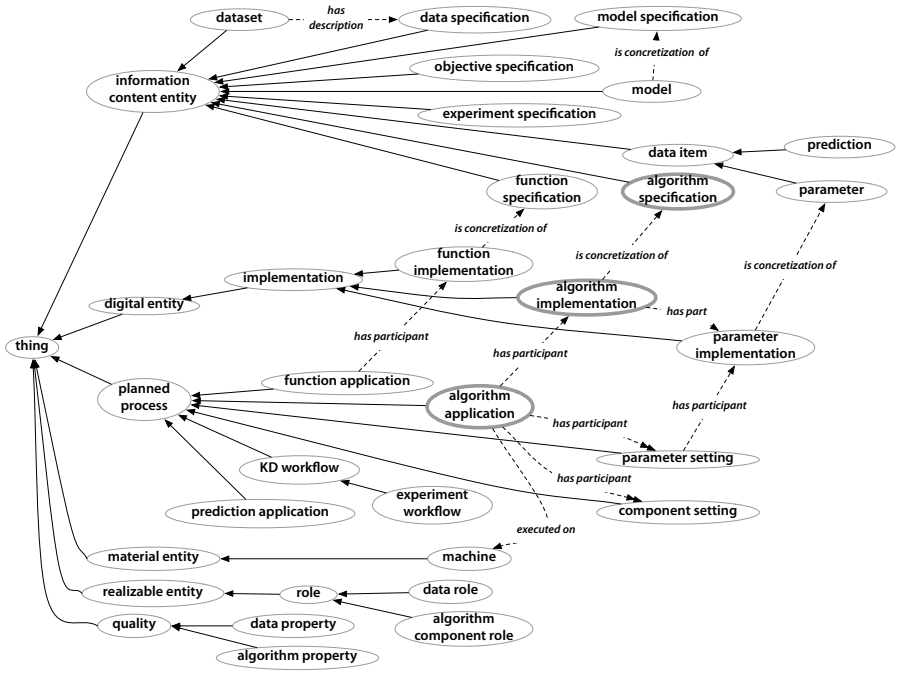


Figure 5.4: An overview of the top-level concepts in the Exposé ontology.

5.3.1 Top-level View

Figure 5.4 shows the most important top-level concepts and relationships. The full arrows symbolize an 'is-a' relationship, meaning that the first concept is a subclass of the second, and the dashed arrows symbolize other common relationships. Double arrows indicate one-to-many relationships, for instance, an *algorithm application* can have many *parameter settings*.

The three most important categories of concepts are *information content entity*, which covers datasets, models, and abstract descriptions of objects (e.g. algorithms), *implementation*, and *planned process*, a sequence of actions meant to achieve a certain goal. When talking about experiments, this distinction is very important. For instance, the concept 'C4.5' can mean the abstract algorithm, a specific implementation or an execution of that algorithm with specific parameter settings, and we want to distinguish between all three.

As such, ambiguous concepts such as 'learning algorithm' are broken up according to different interpretations (indicated by bold ellipses in Figure 5.4): an abstract *algorithm specification* (e.g. in pseudo-code), a concrete *algorithm implementation*, code in a certain programming language with a version number, and a specific *algorithm application*, a deterministic function with fixed param-

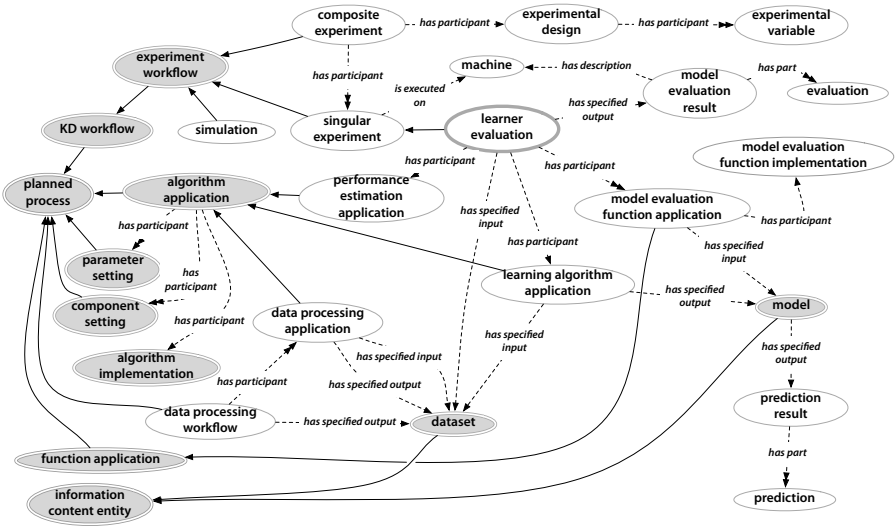


Figure 5.5: Experiments in the Exposé ontology.

eter settings, run on a specific *machine* with an actual input (a *dataset*) and output (a *model*), also see Figure 5.5. The same distinction is used for other algorithms (for data preprocessing, evaluation or model refinement), mathematical functions (e.g. the kernel used in an SVM), and parameters, which can have different names in different implementations and different value settings in different applications. When a function or an algorithm is used as part of another algorithm, e.g. a kernel method or an ensemble, it is called a *component*, and will also be fixed in an algorithm application.

Finally, there are also *qualities*, properties of a specific dataset or algorithm (see Figures 5.11 and 5.12), and *roles*: an algorithm can play the role of a base-learner in an ensemble (also see Figure 5.13) and a dataset can be a training set in one experiment and a test set in the next.

5.3.2 Experiments

Figure 5.5 shows the ontological description of experiments, with the top-level concepts from Figure 5.4 drawn in double ellipses. In essence, experiments are *workflows*, sequences (or directed graphs) of operators, which are defined by *inputs*, *outputs* and *participants*. The general nature of workflows allows the description of many kinds of experiments. Still, some types of experiments, e.g. simulations of agents operating in a certain environment (Frawley 1989) should be represented differently. Some experiments can also consist of many smaller experiments, and use a particular *experimental design* to investigate the effects of various *experimental variables*, e.g. parameter settings. We'll come back to

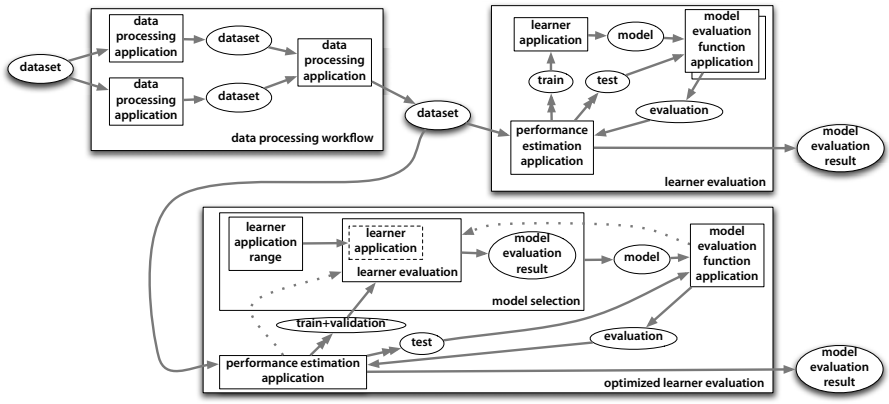


Figure 5.6: Experiment workflows.

this in the next section.

We will now focus on a particular kind of experiment: a *learner evaluation* (indicated by a bold ellipse). This type of experiment applies a specific learning algorithm (with fixed parameters) on a specific input dataset and evaluates the produced model by applying one or several model evaluation functions, e.g. predictive accuracy (see Section 5.3.4). In predictive tasks, a performance estimation technique, e.g. 10-fold cross-validation (see Section 5.3.5), is applied to generate training- and test sets, evaluate the resulting models and aggregate the results. After it is executed on a specific *machine*, it will output a *model evaluation result* containing the outcomes of all evaluations and, in the case of predictive algorithms, the predictions made by the models. Other outputs, such as a final model, are generated by applying the learning algorithm again on the entire dataset.

Finally, more often than not, the dataset will have to be preprocessed first. Since experiments are workflows, we can define setups in which various *data processing applications* preprocess the data before it is passed on to the learning algorithm. Figure 5.6 explicates such a workflow. The ovals represent the in- and outputs: datasets, models and model evaluation results. A *data processing workflow* is an (often used) sequence of data processing steps. Figure 5.6 shows one with three preprocessors. A *learner evaluation* takes a dataset as input and applies performance estimation techniques and model evaluation functions to evaluate a specific learner application. This is a very strict definition of how the learning algorithm is evaluated, making sure it can be interpreted unambiguously. Of course, there are other types of learner evaluations. The bottom part of Figure 5.6 shows a more complex evaluation procedure which we will discuss in Section 5.3.5.

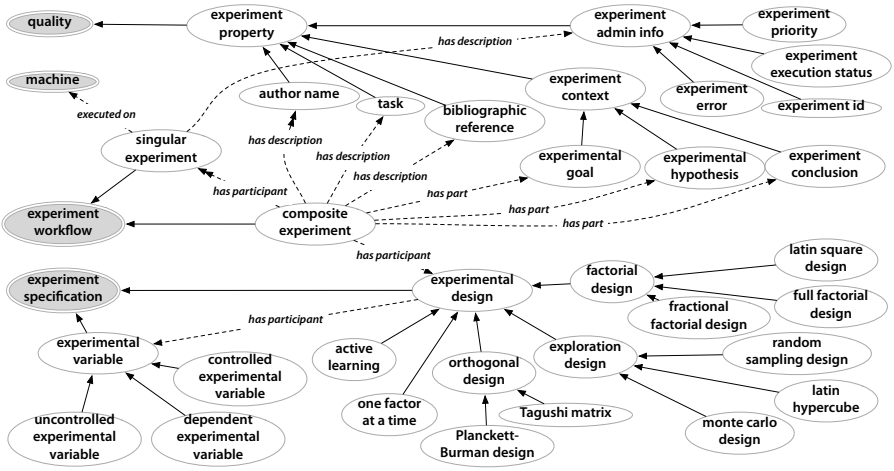


Figure 5.7: The context of a scientific experiment.

5.3.3 Experiment context

Figure 5.7 models the *context* in which scientific investigations are conducted, which is of great importance to describe exactly *how* experiments were set up and to be able to correctly interpret their outcome. Many of these concepts are originally defined in the EXPO ontology (Soldatova and King 2006). Since we treat experiments as fine-grained evaluations that can be reused to answer various questions, we differentiate between a *singular experiment*, a single algorithm run, and a *composite experiment*, a series of experiments set up with a certain goal. The latter can be described by its authors, references to publications in which it features, and the *experiment context*, such as the *goal* of the experiment, the *hypothesis* to be tested and the *conclusions*.

Finally, a composite experiment varies a number of *experimental variables*, which can be (un)controlled or (in)dependent, according to a specific *experimental design* (Kuehl 1999; Sacks et al. 1989) defining which values to assign to each of these variables. In many machine learning experiments, these variables will include the choice of algorithm, their parameter settings or the dataset used. Such designs can be very simple, such as the *one-factor-at-a-time* (OFAT) design, in which one variable is varied while the others are held constant (e.g. on the default value of a parameter setting), but they can also be more complicated, such as factorial, exploratory or orthogonal designs (Ryan 2007). A final, very interesting technique is *active learning* (Cohn et al. 1996; Beygelzimer and Dasgupta 2009), in which new experiments are selected based on the experiments executed before and some goal, e.g. the uncertainty of the outcome of the new experiment or the density of experiments in the space of possible variable assignments.

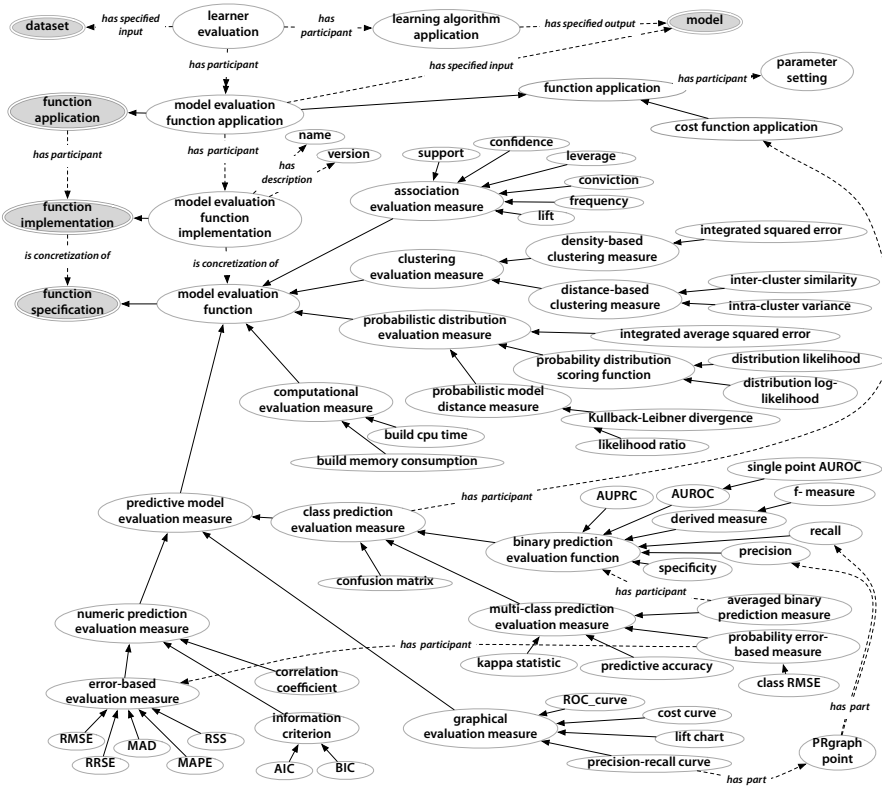


Figure 5.8: Learner evaluation measures in the Exposé ontology.

5.3.4 Learner evaluation

Next to describing how the experiments were set up, we also need to describe exactly how the learning algorithms were evaluated. The ontology currently covers 96 performance measures used in various learning tasks, and some of the more popular ones are shown in Figure 5.8. With the exception of a few measures which will be used repeatedly in Chapter 9, we do not intend to discuss them in detail here since they are covered in many textbooks (Witten and Frank 2005; Bishop 2006). Instead, we will focus on their role in the evaluation process.

Evaluation functions To reiterate, a *learner evaluation* is a *singular experiment* that evaluates a single *learning algorithm application* on a single input *dataset*, by applying a number of *model evaluation functions* on the resulting model. Depending on the task, and thus the kind of model, many different

evaluation functions have been defined. In some tasks, all available data is used to build a model, and the evaluation functions are then simply applied on that model. For instance, when performing clustering (Jain et al. 1999) (the assignment of a set of observations into subsets, called clusters, so that observations in the same cluster are similar in some sense), we typically want to maximize the distance between two clusters (the *inter-cluster similarity*) and minimize the sum of the distances from every point to its cluster's center (the *intra-cluster variance*). There are many more specific ways of defining these measures: for instance, the *single-link* inter cluster similarity measures the distance between the two closest points belonging to different clusters.
Still, as said before, we will focus on supervised predictive modeling, illustrated in the bottom half of Figure 5.8. In this case, the dataset is split up into a training set used to train the model and a test set to evaluate it. Also, instead of using the model itself, it is sufficient (and often more practical) to use the model's predictions to evaluate it. In the case of a binary classification problem (involving only two classes: positive and negative), the total number of predictions can be divided into the number of true positives (TP), true negatives (TN), false positives (FP, predicted positive but actually negative) and false negatives (FN). These values are typically stored in a *confusion matrix* or *contingency table* (also defined for multi-class predictions): it stores, for each set of classes (i, j), the number of cases where class i was predicted as class j. Using the number of predictions of each type, different measures can be defined, such as *predictive accuracy* ($\frac{TP+TN}{TP+TN+FP+FN}$), simply the percentage of correct predictions, *precision* ($\frac{TP}{TP+FP}$), the probability that a positively predicted example is actually positive, and *recall* ($\frac{TP}{TP+FN}$), the probability that a positive example is also predicted positively. The latter two are heavily used in information retrieval tasks. Some measures are derived from others, such as the *f-measure*, the harmonic mean of precision and recall ($2\frac{precison \cdot recall}{precision+recall}$).
It is also important to note that there are many situations in which a false negative might be much worse than a false positive (e.g. when testing for a disease), or vice versa, in which case a *cost function* should be included that returns the actual cost of a misclassification.
In multi-class problems, involving c classes, the same measures can also be used by transforming the multi-class prediction into c binary predictions, in each of which one of the c classes is positive and the others negative, and by taking the weighted average of the performance over these c binary problems, weighted by the number of examples in each class. Some other measures are derived from numeric prediction (regression) problems. For instance, the *root mean squared error* (RMSE) takes the square root of the sum of squares of the differences between the predicted (p_i) and actual (a_i) numeric values of each of the n examples ($\sqrt{\frac{(p_1-a_1)^2+...+(p_n-a_n)^2}{n}}$). In a classification context, the same metric is used by taking the difference between the actual and predicted class probabilities, or 0/1 for an incorrect/correct classification.

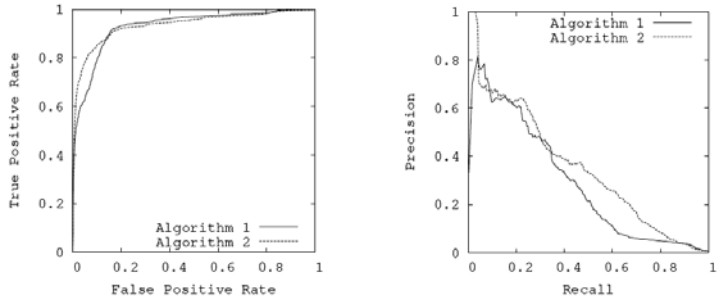


Figure 5.9: The ROC curves (left) and precision-recall curves (right) of two algorithms. The larger the area under the curve, the better the performance.

Graphical evaluations A more powerful way to interpret the performance of a learning algorithm is to store an entire graph instead of a single number, yielding *graphical evaluation measures*. For probabilistic classifiers (which return a probability for each class instead of a single class prediction), it is possible to vary the *classification threshold value* which is used to decide whether an example is positive. When we take two trade-off evaluation metrics, plot these evaluations against each other for each threshold value, and connect the resulting points, we obtain a curve: when precision is plotted against recall we yield a *precision-recall curve* (see bottom right of Figure 5.8); when the true positive rate ($\frac{TP}{TP+FN}$) is plotted against the false positive rate ($\frac{FP}{FP+TN}$), this produces a *receiver operating characteristic*, or *ROC-curve*. Figure 5.9 shows an example of such curves and how they can be used to compare algorithms. The relationship between ROC and precision-recall curves is discussed in Davis and Goadrich (2006). The area under these curves is an often-used evaluation measure, called the *area under the precision-recall curve (AUPRC)* and *AUROC*, also called AUC, respectively. Finally, an alternative way to generate these curves is to vary an algorithm parameter, yielding a new evaluation for each parameter setting, thus obtaining a measure for the algorithm's performance irrespective of the parameter's value.

Provost et al. (1998) and Demsar (2006) have argued that using only *predictive accuracy* can be misleading when comparing algorithms, and that ROC curves are a more accurate measure. In turn, Drummond and Holte (2000) show that ROC-curves can be overly optimistic when class skewness is high, and promote the use of cost curves instead. Hand (2009) shows that the AUC can be misleading if the ROC curves cross and that they inherently use different misclassification cost distributions for different classifiers, a situation which is corrected in an improved AUC-based measure called the H-measure.

In conclusion, there are many ways to evaluate learning algorithms, so it is crucial to explicate exactly what has been measured.

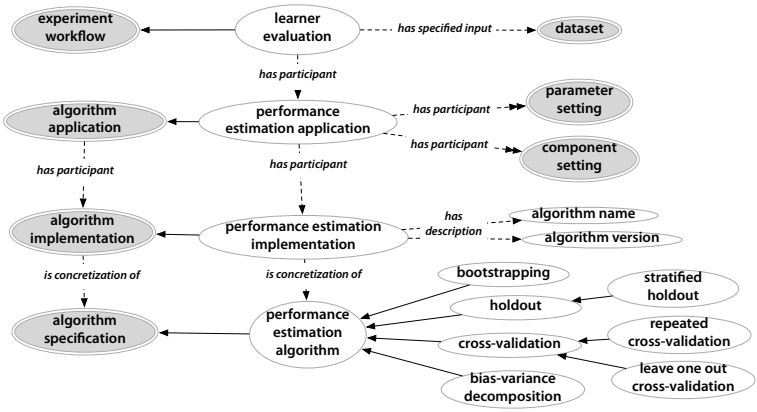


Figure 5.10: Learner evaluation procedures in the Exposé ontology.

5.3.5 Algorithm performance estimation

It is important to realize that when evaluating a model based on a test set, the evaluation obtained is only an *estimate* of its true performance on the target population. If we have huge amounts of data, the test set is likely to be very representative for the target population, but for smaller amounts of data we cannot be so sure. To make maximal use of the available data, the *learner evaluation* can employ a more intricate *performance estimation algorithm* to eradicate, as much as possible, any bias introduced by dividing the data into training and test sets.

These are shown in Figure 5.10. The most basic technique is the *holdout*: simply remove a portion of the data (e.g. one-third), train the algorithm on the remaining examples, and test its predictions on the holdout set. As said before, you may be unlucky: the holdout set may not be representative, or indeed all examples of a specific class may be removed, so that the learner simply does not know about it. One could do a *stratified holdout*, in which the distribution of classes is preserved as well as possible, but this is still a primitive safeguard.

A more fundamental way to attenuate any bias is to repeat the whole process k times with several random samples and average the performance evaluations afterwards, thus reducing the impact of any unlucky dataset splits. One way to do this is the *bootstrap* (see Section 2.2.1.3): sample with replacement until we get a training set as large as the original datasets (but containing duplicates), use the examples that were not sampled as test examples, and repeat this k times. While useful for small datasets, each training set will contain, on average, only 63.2% of the examples.

One of the most popular performance estimation procedures is *k-fold cross-*

validation: divide the dataset randomly in k equal parts (dubbed *folds*), stratified or not, and perform n holdout evaluations, each one using one of the folds for testing and the remaining $k-1$ for training. Experimental testing shows that cross-validation is almost completely unbiased for $n = 10$, generating training sets containing 90% of the examples (Markatou et al. 2006). Bias can be reduced further by running several cross-validation procedures, yielding, for instance, *5x2 cross-validation*: 5 iterations of 2-fold cross-validation (Dietterich 1998), after which the results over all iterations are averaged. A final variant is *leave-one-out cross-validation*, in which k is the total number of instances in the dataset. This provides a maximal amount of training examples, and zero bias, but it is of course much more expensive. Still, this is not a complete list: many variants of these estimation techniques exist.

A final algorithm performance estimation technique is the *bias-variance decomposition*, previously described in Section 2.5.2.1. There exist several ways to perform this decomposition (Kohavi and Wolpert 1996; Webb and Conilione 2005), but they all involve running a large number (e.g. 200) of bootstraps, and tracking whether examples are systematically being misclassified or only on some iterations.

Comparing algorithms When comparing learning algorithms, it is important to make sure they are trained on the same data. For a 10-fold cross-validation procedure, this means that the exact same folds must be used to train each learning algorithm. However, Bradford and Brodley (2001) demonstrate experimentally that using only one partition of the instances into cross-validation folds may lead to statistically erroneous conclusions when used to compare algorithms, providing further motivation to perform multiple cross-validation runs. In any case, if we wish to reuse prior evaluations of rival algorithms (e.g. when benchmarking new algorithms), it is important that these folds can be easily retrieved or generated anew. In this respect, note that these performance estimation procedures are algorithms in our ontology, meaning they also have implementations with version numbers and applications with parameter settings (e.g. the number of folds and the random seed used to split the data). As such, the evaluation procedures can always be repeated exactly.

The experimental setup becomes even more convoluted when algorithm parameters are to be optimized. Indeed, while a *learner evaluation* always evaluates a specific learner application (with specific parameter settings), sometimes one wants to evaluate a learning algorithm under optimized parameter settings. Here, it is important that the data used to optimize the parameter settings is different from the training data, and the data used to estimate the performance of the resulting model must be different from both. This requires another kind of experiment, called an *optimized learner evaluation*, illustrated in the bottom half of Figure 5.6, which evaluates a *learner application range*, a set of learner applications with various parameter settings. It is a two-step approach

in which first a portion of the data is used in a *model selection* procedure to find the optimal parameter settings, after which the optimal model is evaluated on a separate test set. The model selection procedure contains standard *learner evaluations* (which could be stored separately as well), in which the data is split again into training and so-called *validation* or *optimization sets*. The performance estimation procedure is typically the same in the two steps. When this is a cross-validation procedure, it is often called a *double* or *two-step cross-validation* procedure: it outputs a single evaluation for the algorithm, aggregating the performance of models optimized on several data folds. Again, we store all used implementations and parameter settings so the results can be interpreted unambiguously.

Statistical significance tests Finally, statistical tests are needed to establish whether the performance differences between algorithms are statistically significant. For classification, Dietterich (1998) evaluates five commonly used statistical tests, based on the probability of *Type I error*, incorrectly detecting a difference when no difference exists, and the *power* of the test, the ability to detect algorithm differences when they do exist. The first two tests are the difference of the proportions of incorrectly classified examples (Snedecor and Cochran 1989), and the resampled paired t-test based on taking several random train/test splits. While especially the latter is often used, they have been shown to have a large Type I error. The also very popular 10-fold cross-validation paired t-test performs better in this respect, and is also the most powerful test, but still exhibits a somewhat elevated probability of Type I error. The 5x2cv paired t-test, based on 5 iterations of 2-fold cross-validation performs even better, yielding acceptable Type I error while still being very powerful. Finally, McNemar's test (Everitt 1992), which involves building a contingency table recording the number of examples misclassified by either, both, or none of the two algorithms, has very low Type I error, but is a bit less powerful than the 5x2cv paired t-test. As such, the 5x2cv test is recommended for algorithms that are fast enough to run 10 times, while McNemar's test is recommended for those that can be run only once. Alpaydin (1999) later constructed a more robust 5x2cv F test with a lower Type I error and higher power, and Nadeau and Bengio (2003) proposed a corrected resampled t-test that adjusts the variance based on the overlaps between subsets of examples. Bouckaert and Frank (2004) investigated experiment replicability, found the 5x2cv t-test unsatisfactory and opted for the corrected resampled t-test.

While these test are executed on one dataset at a time, after which typically the number of win-ties-losses are reported, Demsar (2006) proposes statistical tests for comparing algorithms on multiple datasets: the Wilcoxon signed ranks test (Wilcoxon 1945) for comparison of two classifiers and the Friedman test (Friedman 1940) with the corresponding post-hoc tests for comparison of more classifiers over multiple data sets. We shall be using the latter to compare

learning algorithms over large numbers of datasets in Chapter 9.

5.3.6 Datasets

Next to the evaluation procedure, there are two more components needed to perform a learner evaluation: the learning algorithm and the dataset. Figure 5.11 shows the most important concepts used to describe the latter. Several types of datasets are defined: collections such as image and document corpora, relational data such as databases and graphs, and propositional datasets such as item sequences, time series and attribute-value tables. For now, only the last type of data is developed in detail.

Specification The *data specification* (in the middle section of Figure 5.11) describes the structure of the dataset. A dataset can have a number of *data features* and *data instances*. If the dataset is labeled, it has one or more *target features*, which are nominal in *classification data* and numeric in *regression data*. Individual instances can be further described with *instance properties* indicating, for instance, whether the instance is actually labeled. Individual features can be described with their name and type, e.g. numeric (integer, real,...) or a set of possible nominal values. This constitutes a minimal description of the data, similar to the file header in the ARFF dataset format. For other types of data (e.g. relational data) this specification will have to be extended. Finally, a dataset has descriptions, including the dataset name, version, download url and a textual description of its origin, which are needed to make the dataset easily retrievable.

Roles A specific dataset can play different *roles* in different experiments (top of Figure 5.11): it can be a training set in one evaluation and a test set in the next. This is mostly useful in experiments where we want to define this explicitly, or when we want to store the results of the evaluations on individual folds for additional meta-data. The same datasets can also assume different roles outside a data mining context.

Data properties As said before, we wish to link all empirical results to theoretical metadata, called *properties*, about the underlying datasets to perform meta-learning studies. These *data properties* are shown in the bottom half of Figure 5.11, and may concern individual instances, individual features or the entire dataset. Both *feature properties* such as feature skewness or mutual information with the target feature, as well as *dataset properties* of various kinds have been discussed in detail in Section 2.4. They can be attached to new or existing dataset definitions and are thus also linked to all experiments on those datasets.

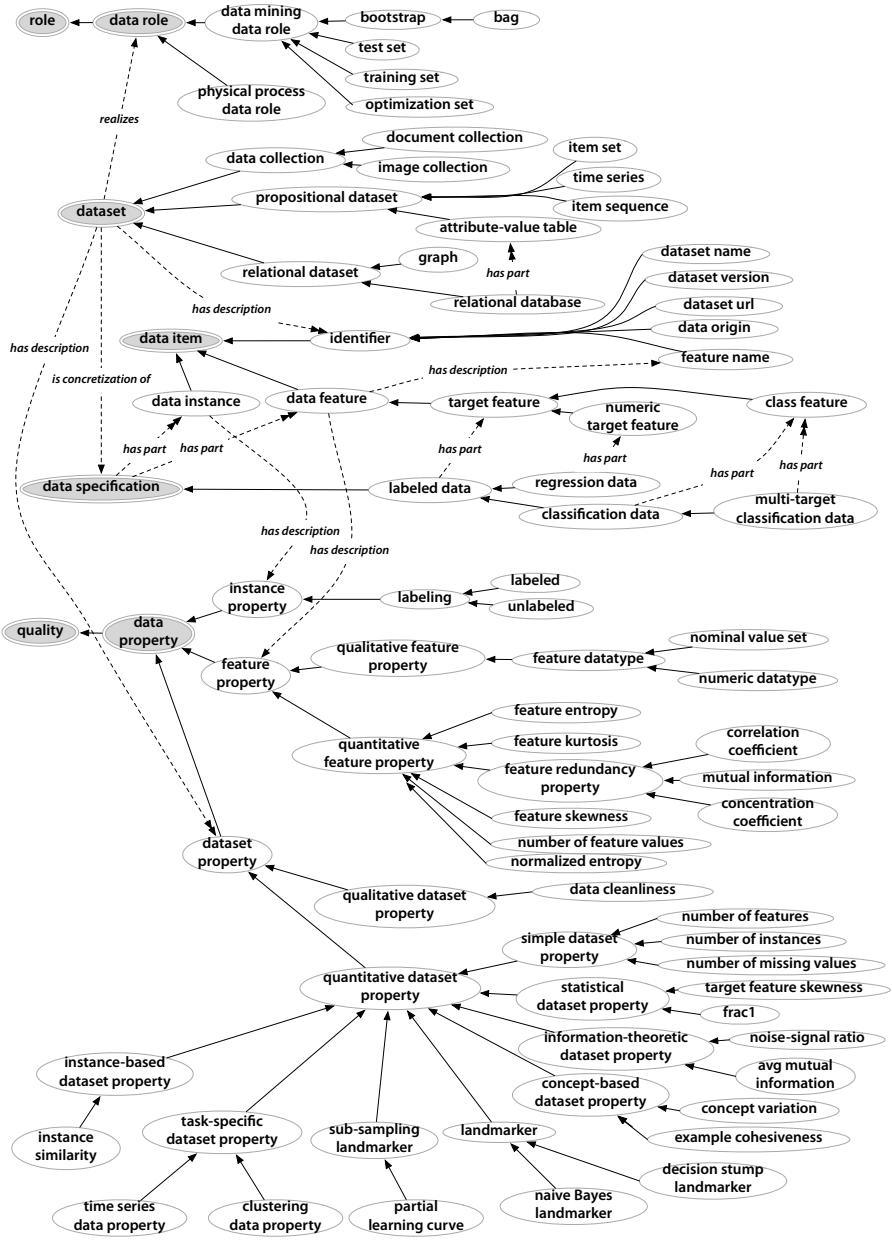


Figure 5.11: Datasets in the Exposé ontology.

5.3.7 Algorithms

One last aspect of the ontology we wish to highlight is the description of algorithms. To allow the exploration of algorithm performance under different configurations and parameter settings, and include all aspects that influence their performance in queries, we need a detailed vocabulary to describe them.

Algorithm implementations Figure 5.12 shows how algorithms and their configurations are expressed in our ontology. As mentioned in Section 5.3.1, *algorithm applications* are *algorithm implementations* with specific parameter and component settings. *Parameter settings* attach a value to a *parameter implementation*, which depending on the algorithm in which they are implemented, can have different internal names and different default values. Algorithm implementations are described with all information needed to retrieve and use them, such as their name, version, url, programming language, and the library they belong to (if any). Moreover, they can be further described with certain informative properties, e.g. their susceptibility to noisy data, as discussed in Section 2.5. Like dataset properties, these are linked to all experiments using these algorithm implementations and can be used to analyze the results.

Algorithm parameters Parameter implementations are concretizations of general algorithm parameters. While this is not explicitly shown in Figure 5.12, well-known parameters of learning algorithms are defined and linked to the learning algorithms or functions to which they belong. For instance, a decision tree typically has a parameter stating the minimal amount of examples in a leaf node before it splits, and a radial basis function (RBF) kernel typically has a 'gamma' parameter reflecting the width of the basis functions. As such, we can use these general names in queries, instead of the often obscure names used in algorithm implementations. Finally, note that we refer to algorithm parameters as *hyperparameters*, to distinguish between model parameters: the parameters that define the structure of certain representational models.

Algorithm components Next to parameter settings, some algorithms also have internal components: other algorithms or mathematical functions which have a great influence on the behavior of the learning algorithms and which can be selected by the user. As these components can often be used by themselves as well, we explicate the *roles* they can play when used in a learning algorithm, such as base-learners in ensemble learners, distance functions in clustering and nearest neighbor algorithms and kernels in kernel-based learning algorithms. See Figure 5.13. As such, we can describe the make-up of a learning algorithm application exactly. Moreover, different *implementations* of a general algorithm may also include internal data processing techniques (e.g. data discretization) to make them more universally applicable. When analyzing their results, it is

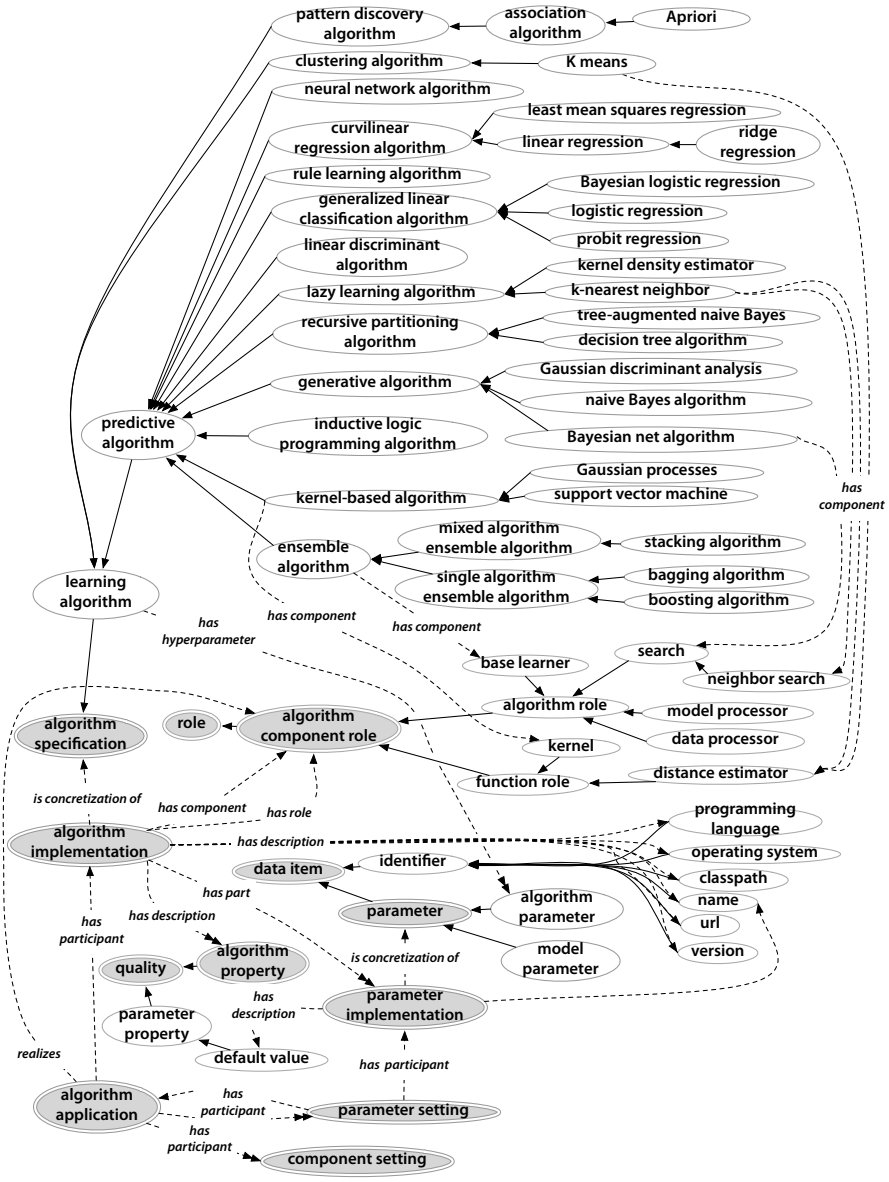


Figure 5.12: Algorithms and their configurations in the Exposé ontology.

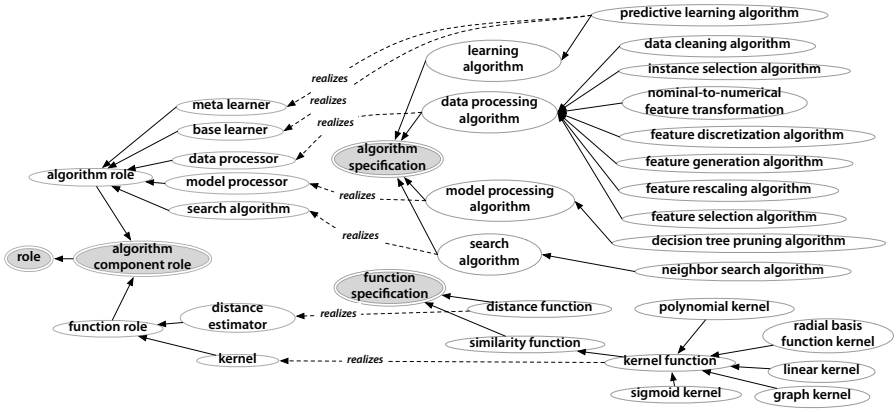


Figure 5.13: Algorithms and functions can act as algorithm components.

important that we know exactly what data processing is performed internally in different algorithm implementations.

The top half of Figure 5.12 shows the classes of algorithms that are currently defined in the ontology, with a strong focus on predictive algorithms, and which components they generally have. For instance, a support vector machine (SVM) is always assumed to have a kernel, and when the SVM is applied, the kernel and the kernel implementation will be fixed. Note that many of these components can be parameterized as well, and their parameter settings are thus defined in their specific algorithm or function application description, not as parameters of their overarching learning algorithm. This is to ensure that these parameters can be queried for in a uniform way. We won't discuss the workings of all these algorithms in this text, since they are amply described in many machine learning textbooks.

Algorithm mechanisms Finally, as discussed in Section 5.1.3, we wish to offer better ways to describe the bias of learning algorithms, and thus need to look at the internal learning mechanisms on which they are built. Figure 5.14 describes how the ontology explicates three mechanisms that have a great influence on a learner's bias: the *model structure*, the structure used to represent regularities in the data, as well as the type of *decision boundaries* this model entails, the *optimization strategy* used to refine the representational model to better fit the data at hand and, for predictive algorithms, the *prediction strategy* used to apply the model and predict the outcome of unseen examples. Extensive hierarchies are defined for each of these aspects, often inspired by the DMOP ontology (Hilario et al. 2009). Still, this hierarchy is certainly not complete and should be extended further. As with *algorithm properties*, the principle aim of this description is to include these mechanisms in meta-learning queries.

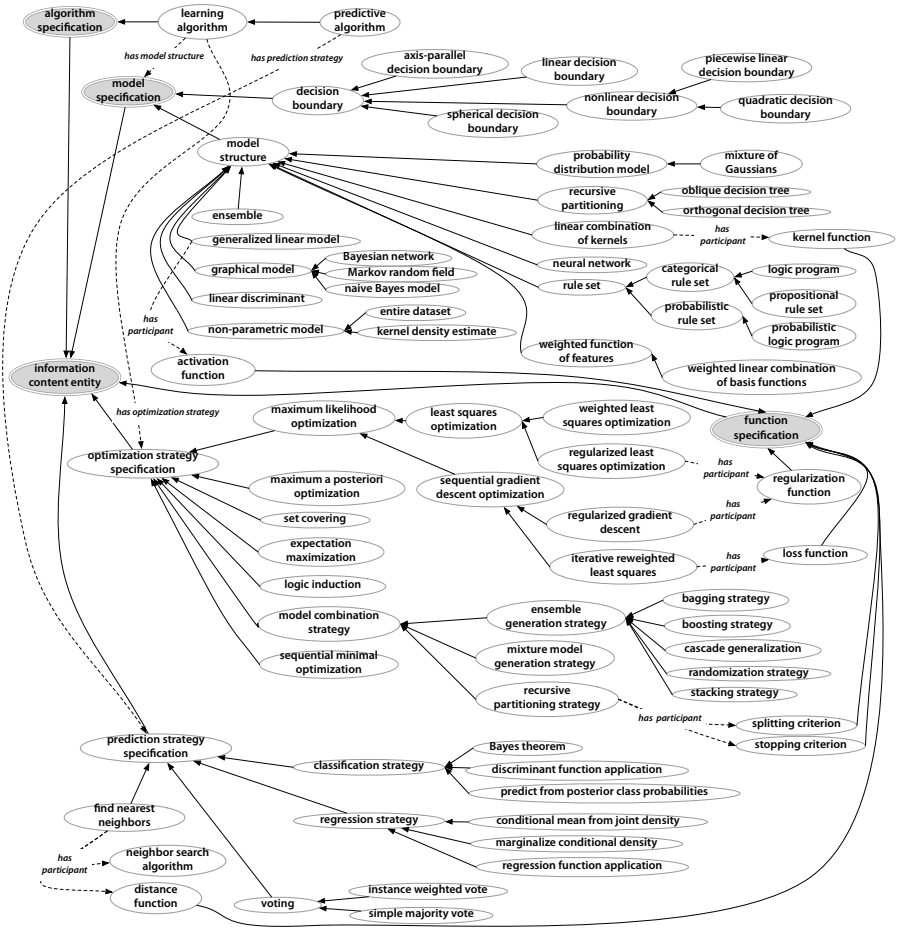


Figure 5.14: Internal learning mechanisms in the Exposé ontology.

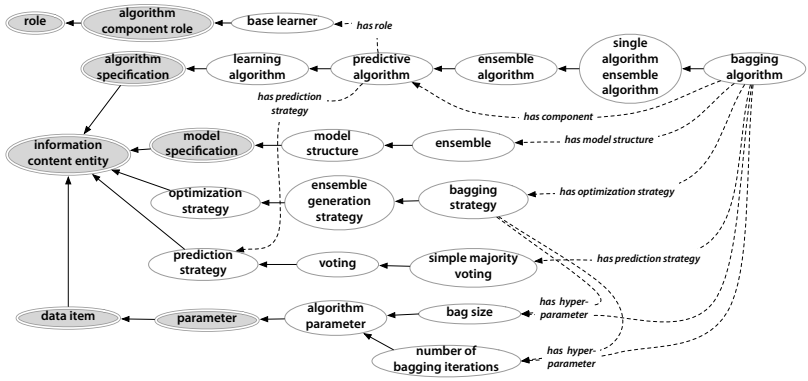


Figure 5.15: Algorithm specification in the Exposé ontology.

5.3.8 Algorithms: an example

Figure 5.15 shows the representation of the bagging algorithm (Breiman 2001), previously discussed in Section 2.2.1.3, in terms of its general classification, components, parameters, model structure, optimization strategy and prediction strategy. We can only show the direct ancestors here. Thus, the bagging algorithm is an *ensemble algorithm* which uses a single base-learner. It has one component: a predictive algorithm which fulfills the role of *base learner*. Its model structure is simply an ensemble of other models, and its optimization strategy is an *ensemble generation strategy*, which tries to improve the ensemble by adding new, preferably very different models. It does this by bagging: sample the training data to get new training sets (bags) of a certain size, build a new model using the base learner and repeat this a number of times. This establishes two parameters as shown. To predict the outcome of new examples, it performs a simple majority vote over all models in the ensemble. Just to clarify, while this model shows that the parameters actually belong to the bagging procedure, they are defined as parameters of the general algorithm as well. By describing these internal mechanisms it becomes possible to define exactly how two similar algorithms differ from each other, and investigate the effects this may have.

5.4 Conclusions

In this chapter, we have established an ontology for experimentation in machine learning: a domain model that covers a large portion of the involved concepts and the relationships between those concepts, thus providing a core vocabulary on which we can build common, extensible representation languages to share

thousands of experiments with the world, as well as database models which allow queries on any aspect of the experimental setup, or on any aspect or property of the involved learning algorithms and datasets.

While focussing on predictive accuracy and propositional datasets, the resulting ontology is quite extensive. It treats experiments as workflows to allow the definition of various types of experiments and includes the possibly extensive data processing workflows that precede them. It also covers the experiment context in which learning algorithm evaluations are performed, stating which were the experimental variables, what hypotheses were tested and what was concluded from the results. Next, we covered the various learner evaluation measures that are used in current research, as well as the techniques used to estimate the performance of predictive learning algorithms on unseen data. We also covered the structure and properties of datasets to link them to the performance of learning algorithms, and last but not least, we described the properties and structure of learning algorithms themselves, including their parameters, internal components such as kernels, base-learners and distance functions, and their internal learning mechanisms such as the representational model and optimization strategy. All this is described in sufficient detail to reproduce the described experiments exactly and to ask questions about many aspects of the underlying components.

The end result is a very rich and well-structured vocabulary which, as we shall see in the next chapter, allows to describe supervised classification experiments in all their aspects. Still, it is also limited in many respects. First of all, the ontology should be extended towards other tasks, such as unsupervised learning. Furthermore, we did not investigate in depth how to handle other types of experiments: simulation experiments, time series analysis and mining data streams will probably require many extensions. Still, since an experiment is presented as a workflow, it should be possible to design new workflows that describe these experiments, and to translate them into database models allowing the queries most useful in those domains. Regarding reinforcement learning, lessons can be learned from the RL logbook project.[5]. Probably the greatest caveat is that we do not yet store models in a way that allows us to write queries about their structure. Here, many lessons may be learned from inductive databases to remedy this (Fromont et al. 2007).

Yet, it provides a firm starting point for extensions in all these directions. It allows researchers from various subfields of machine learning to add their own additional concepts needed to describe their experiments, after which these changes can be translated to new or updated markup languages and database schemas for those subfields. It is also linked to wider ontologies, such as OntoDM and EXPO, whose additional concepts facilitate the extension of this ontology in breadth, while other ontologies, such as DMOP, allow to extend it further in depth.

[5] http://logbook.rl-community.org/

In rivers, the water that you touch is the last of what has passed and the first of that which comes; so with present time.

Leonardo Da Vinci

Chapter Six

The ExpML Markup Language

Having developed a formal vocabulary for describing machine learning experiments in the previous chapter, we can now start to use it to automatically stream machine learning experiments from the software agents running the experiments to local or online repositories where they can be analyzed and reused by anyone interested in the results. In this chapter, we will introduce an XML-based, formal representation language for machine learning experiments, dubbed ExpML, based on the Exposé ontology. As we shall see, this language is a quite straightforward translation from ontological concepts to XML elements and attributes, and from ontological relationships to XML syntax.

The automatic exchange and organization on machine learning experiments enabled by such languages improves greatly over the current methodology in machine learning, which is to only publish averaged results in papers. Indeed, Soldatova and King (2006) argue that "at the beginning of the 21st century, the formalization of scientific knowledge is no longer just a philosophical desirable, but a technological necessity." More powerful experimentation tools and computer clusters output ever large amounts of experiment data, which allows us to perform ever more valuable and accurate research, but also increases the need for empirical results to be filtered and organized automatically.

Moreover, it engenders a much more dynamic, useful and convenient way to publish empirical investigations (Nielsen 2008):

> ... the journal system is perhaps the most open system for the

> transmission of knowledge that could be built with 17th century media. [...] This system has changed surprisingly little in the last 300 years. The Internet offers us the first major opportunity to improve this collective long-term memory, and to create a collective short-term working memory, a conversational commons for the rapid collaborative development of ideas.

Indeed, by sending our machine learning experiments onto the network and into tools that automatically organize them, we build up a large collective memory of empirical observations that can be tapped into by researchers all over the world to speed up research and collaborate dynamically on certain topics. Sharing experimental results makes them tremendously more valuable than when they just sit on someone's hard drive: they can fuel further research, by many researchers, for years to come.

As shown in Figure 4.4, we wish to realize this in practice by extending data mining software agents so they can import and export ExpML descriptions of experiments. A number of data mining tools do already offer some features to facilitate the logging and sharing of experiments, but unfortunately, each tool uses its own format to describe the experiments, which only covers experiments run with the tool in question. This is sufficient as a personal, although typically unstructured log of experimentation, but does not allow the free exchange of empirical information pursued here. To the best of our knowledge, the only standardized language is the Predictive Model Markup Language (PMML)[1], which allows to exchange predictive models, but not detailed experimental setups nor evaluations.

The remainder of this chapter is structured as follows. Since we want to make sure that all shared experiments are reproducible, we shall first propose some guidelines about the required minimal information about a machine learning experiment in Section 6.1, somewhat similar to the MIAME guidelines mentioned in Section 4.3.2.1. In Section 6.2, we discuss how we can translate our ontological domain model to an XML language, and illustrate this process and the ensuing ExpML language using a real example. Section 6.3 concludes.

6.1 Minimal Information about an ML Experiment

In this section, we propose some simple guidelines to ensure that shared experiments are reproducible. We consecutively discuss how the learning algorithms, the datasets, and the evaluation procedures should be described to achieve this goal.

[1] See `http://www.dmg.org/pmml-v3-2.html`

6.1.1 Algorithms and Functions

When describing new *algorithm implementations*, one should ideally store a complete symbolic description of the implementation of an algorithm, and experiment databases could in fact be designed to store, for instance, the entire source code. However, as a minimal requirement, formal descriptions should provide the name and version of each algorithm, together with a pointer to source code or an executable, so that the algorithm can be retrieved at any time. Alternatively, since url's do change, it might be interesting to give each algorithm a unique identifier, similar to the identifiers given to scientific publications. Some identification of the environment (e.g. the required operating system and libraries) and possibly some guidelines on how to setup and run the algorithm complete this description.

Optionally, a description of the algorithm's parameters can also be added to describe, for instance, their function in the algorithm and which value ranges are sensible. Moreover, we can also annotate the algorithm further with generally known or calculated properties of the algorithm (see Section 2.5). It should also be possible to define new algorithm properties at any time, preferably accompanied by a detailed description of the exact procedure or formula used to calculate them.

When describing experiments run with these algorithms, the values of all parameter settings should always be included. For randomized algorithms, we must also store the seed for the random generator they use as another parameter. As such, an algorithm application is always a deterministic function. In the cases where learning algorithms are dependent on other algorithms or functions, such as base-learners, kernels or search functions, these should also be described in the same way as we store algorithm applications, including any parameter settings those methods may have. The exact composition of the main algorithm should also be made clear.

6.1.2 Datasets

Similarly, to describe datasets, one can store name, version and a pointer to a representation of the actual dataset. The latter could be an online text file (possibly in multiple formats) that the algorithm implementations can read, but it could also be a dataset generator together with its parameters (including the generator's random seed) or a detailed description of the data processing workflow that produced it, including its structure, the data processing applications involved (again including their parameter setting) and a pointer to the input dataset. If storage space is not an issue, one could also store the dataset itself in the database. Optionally, characterizations of the dataset (see Section 2.4) can be attached to its description, and new data characterizations should provide the procedure or formula used to calculate them.

6.1.3 Evaluation Procedure

To correctly interpret (and repeat) the outcome of an experiment, we need to describe exactly how the algorithm is run and evaluated as well. For instance, in case we use a cross-validation procedure to estimate the predictive performance of the algorithm on unseen data, this implies storing (a seed to generate) the exact folds. In case a piece of code is used to perform this procedure, we could again store it the way we did with algorithms, i.e. as a deterministic function. Also the exact functions used to compute performance estimates (e.g. RMSE, predictive accuracy,...) should be described. To make the experiments more reusable in various domains, it is advisable to compute a variety of frequently used metrics, or to store the information from which they can be derived. In the case of classifiers, the latter implies storing the full contingency table (see Section 5.3.4).

Finally, to be able to verify the speed of an algorithm, i.e. how long it takes to build a model, we also need to store a detailed description of the used machine to be able to compare experiment runtimes. This is only an issue for comparing runtimes, as most performance metrics are independent of the machine used. One approach, currently used in several data mining challenges, is to report the results of a benchmark process assessing the machine's relative speed.

6.1.4 Models and Contexts

Even if an experiment is fully repeatable, there are other bits of information we still might want to store. Indeed, another important outcome of the experiment is the model generated by the algorithm. For instance, one could store specific properties of these models, such as the time to learn the model, its size, and model-specific properties (e.g. tree depth) for further analysis. If storage space allows this, also a full representation of the model could be stored for later visualization or analysis, e.g. using existing model description languages like PMML, although it might be cheaper to rebuild the model if it is needed. For predictive models, it is also useful to store the individual (probabilistic) predictions for each example in the dataset as it allows the computation of additional (or new) evaluation criteria without rerunning all experiments.

Moreover, as discussed in Section 5.3.3, an experiment is usually part of a range of experiments created to test a certain hypothesis. Therefore, it would be highly useful to describe exactly how this range of experiments was generated. For instance, one could have run a fixed algorithm on a range of datasets, or varied one of its parameter's values using a normal distribution around the default value. A detailed description of the exact experiment setup makes sure that the complete range of experiments can be repeated at any time.

6.1.5 Replicability versus Generalizability

Drummond (2009) remarks that *replicability*, the ability to obtain exactly the same result under exactly the same conditions, should be discerned from *'reproducibility'*, the ability to obtain the same result even if some conditions change. "Reproducibility requires changes; replicability avoids them." He claims that replicating an experiment brings no new knowledge, and is harder to achieve (requiring much more detail to be reported) than reproducing it, which at least teaches us that a result is general enough to also hold under different conditions. While this is a valid argument when we wish to build upon the *conclusions* of other researchers, in which case the ability to obtain the same result even if some conditions change should really be called 'generalizability', it is not valid when we wish to build upon their exact experimental observations. For instance, 'the earth revolves around the sun on an elliptical path' is a conclusion that should be reproducible even with somewhat different measurements, but when we want to investigate the exact shape or tilt of the ellipse not reported in previous studies, we need the exact astronomical observations.
Reusing prior observations for further analysis is only possible if they are described in sufficient detail to interpret them unambiguously, so that they can be clearly related to new observations. The goal here is not to verify the outcome per se, but rather to explore the results further. Also for browsing experimental data stored in a database, having the specific conditions is extremely valuable as a way to navigate the space of all possible experiments. While the queries we write can be very general, the underlying data should be exact in order to assure that the correct results are being returned.

6.2 From Ontology to Markup Language

Now we know which information we should share, we need to find a way to express it in an unambiguous, machine-interpretable way. Using the Exposé ontology as our core vocabulary offers us a principled approach to build such a language: associate ontological *concepts* with XML *elements*, and express ontological *relationships* to XML *syntax*. This translation process is especially useful because it allows ontological extensions (e.g. to new machine learning tasks) to be translated directly to updated ExpML definitions. These definitions (e.g. in XML Schema Definition format) can then be used by researchers to express their experiments in XML. Software interfaces, see Chapter 8, can be build to automatically export the experiments of a certain data mining tool to ExpML files, or upload them to a repository.
Only when ontological extensions are needed to support new kinds of experiments does one have to translate the extensions into an updated XML definition, and adapt software interfaces to include the changes. The choice for XML as our modeling language is quite straightforward: it is well-known and

ubiquitous in software agents (e.g. web services) and many tools exist to create, manipulate or verify XML data.

6.2.1 Operators and processes

First, we need to define which aspects of machine learning experiments we wish to share. We can divide these in two groups:

- Definitions of new experimental operators, such as new algorithms, datasets, evaluation functions and kernels, including known or measured properties of these operators, such as dataset or algorithm characterizations. These correspond to *implementations*, *datasets* and *qualities* in our ontology.
- The experimental setups and results, corresponding to *planned processes* in our ontology: *experiment workflows*, *data processing workflows*, and all their outputs, such as *model evaluation results*.

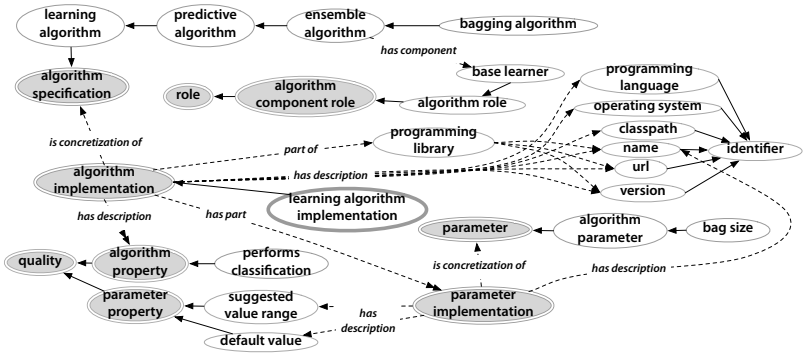
Because ontological relationships are more expressive than XML syntax, different relationships between these concepts need to be translated quite differently. Rather than producing a list of guidelines, we start from an example experiment which we wish to share, and discuss how we can translate the different relationships as they occur.
Our example is a real experiment (experiment 445080 in our experiment database) which we want to express in ExpML: we will first describe a new dataset and algorithm implementation, and then we perform an experiment in which the dataset is preprocessed with a feature selection technique, and subsequently used to evaluate the newly added algorithm implementation.

6.2.2 New Operators

6.2.2.1 Algorithm implementations

We start by describing a new algorithm implementation, i.e. WEKA's implementation of the bagging algorithm. Figure 6.1 shows the ExpML description, together with the corresponding Exposé concepts above. First, we take the core concept, *learning algorithm implementation*, and convert it into a similarly named XML element. We used a shorthand, *learner implementation*, since there is no ambiguity.
While this concept itself has no specific relationships with other concepts, it inherits many relationships from its parent, *algorithm implementation*:

Has-part and has-participant For these relationships, the target simply becomes a nested XML subelement of the source. For instance, in Figure 6.1, an *algorithm implementation* can contain multiple *parameter implementations*. They occur in the XML code as nested elements of `learner implementation`.




```
<expml>
  <learner_implementation name="weka.Bagging" version="1.31.2"
      url="http://www.cs.waikato.ac.nz/..." language="java" os="any"
      libname="weka" libversion="3.4.8" instance_of="bagging_algorithm"
      classpath="weka.classifiers.meta.Bagging">
    <parameter_impl name="P" instance_of="bag_size">
      <property name="default_value">100</property>
      <property name="suggested_value_range">20,40,60,70,80,90,100</property>
    </parameter_impl>
    <parameter_impl name="I" instance_of="number_of_iterations">
      <property name="default_value">10</property>
      <property name="suggested_value_range">10,20,40,80,160</property>
    </parameter_impl>
    <parameter_impl name="S" instance_of="random_seed">
      <property name="default_value">1</property>
    </parameter_impl>
    <learner_implementation role="base-learner"/>
    <property name="performs_classification">true</property>
    <property name="performs_regression">true</property>
    <property name="handles_missing_values">true</property>
    <property name="handles_numeric_attributes">true</property>
    ...
  </learner_implementation>
```

Figure 6.1: An algorithm implementation, in Exposé and ExpML.

We shorten the names of implementation and application elements to 'impl' and 'appl'.

Has-description We differentiate between two types here. In the first type, the target concept is a subclass of *identifier*, e.g. *name* in Figure 6.1, in which case the target becomes an attribute of the parent element: see the attributes `name`, `version`, `url` and so on in the XML code. Moreover, if the identifier is necessary for reproducibility (see previous section), it becomes a *required* attribute in the XML schema definition so that ExpML files can be automatically verified for repeatability.

In the second type, the target is a subclass of *quality*, in which case it becomes an XML subelement called `property` with name-value attributes, as shown at

the bottom of Figure 6.1. Note that *parameter implementation* also has *has-description* relationships, likewise resulting in XML attributes and `property` subelements.

Is-concretization-of This relationship points to the general, implementation-independent description of algorithms, functions and parameters, which is referenced in an `instance of` attribute. Here, for instance, our algorithm implementation is an instance of a *bagging algorithm*, shown at the very top of Figure 6.1. The same goes for parameter implementations and their more general, unambiguous names, e.g. *bag size*.

Has-component If a concept is an instance of another concept, it also inherits its relationships. Since our implementation is an instance of an ensemble algorithm, it inherits a *has-component* relationship, a special case of the *has-participant* relationship, pointing to a certain *role* to be fulfilled. The element expected to fulfill that role, in this case another `learner implementation`, is added as an subelement in the XML code. The role to be fulfilled, *base-learner* in this case, is specified in a `role` attribute.

Part-of Finally, *part-of* relationships, here indicating that the algorithm implementation is part of a programming library, are translated as attributes: the libname and libversion attributes in the XML code.

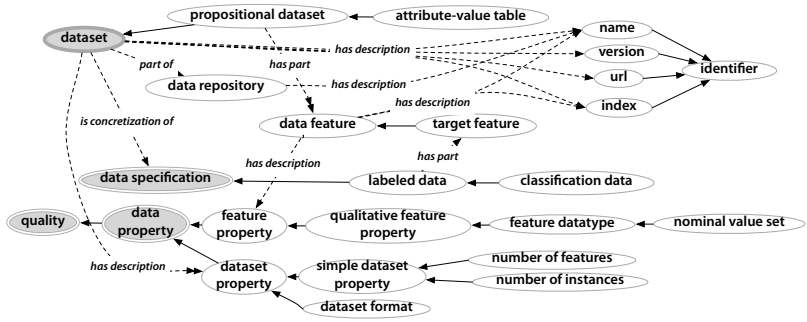
6.2.2.2 Dataset

Figure 6.2 shows the definition of a new dataset. The translation to XML is analogous to the previous example, with the exception that we don't wish to define new XML elements for every type of dataset, such as *attribute-value table* as is the case here. We therefore use a `type` attribute to specify the kind of dataset we are dealing with. Note that the dataset has features because it is propositional, and a target feature because the data is labeled.

6.2.3 Experiment Workflows

Figure 6.3 shows, from bottom to top, a small experiment workflow, the expression of the first half of this workflow in ExpML and the corresponding parts of the ontology. We start with the *data processing workflow* in the ontology, which has datasets as input and output. A new type of relationship appears:

Has-specified-input The input of an element becomes an attribute, `input data`, of the source element. Here, we first include our previously defined dataset (assuming the dataset is known, a shorter reference suffices), and state it as the input of the `data processing workflow`. To allow a flexible workflow



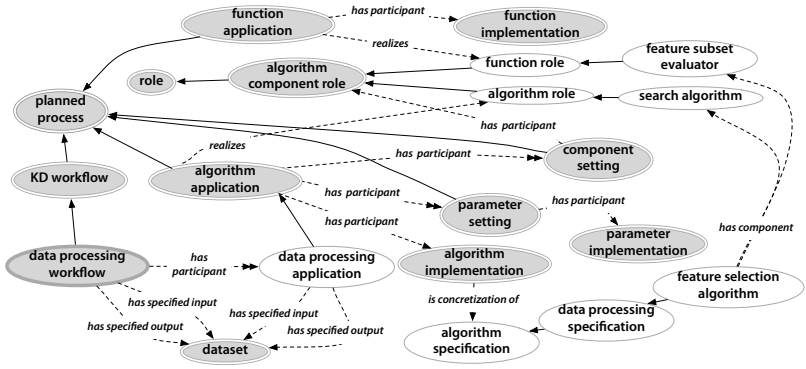

```
<dataset name="kr-vs-kp" version="1" repository="uci" type="attribute-value"
     instance_of="classification_data" url="http://...">
  <target_feature name="class" index="-1"/>
  <feature name="bkblk" index="0">
    <property name="datatype">{t,f}</property>
  </feature>
  ...
  <feature name="class" index="37">
    <property name="datatype">{won,nowin}</property>
    <property name="entropy">0.998576</property>
  </feature>
  <property name="format">ARFF</property>
  <property name="number_of_features">37</property>
  <property name="number_of_instances">3196</property>
  <property name="noise-signal_ratio">30.3908</property>
  <property name="default_accuracy">0.522215</property>
  <property name="landmarker_1NN">0.672426</property>
  ...
</dataset>
```

Figure 6.2: A dataset definition, in Exposé and ExpML.

definition, inputs and outputs are given an id as one of their attributes. As such, if a dataset with `id='d1'` serves as an input, the receiving element will have an attribute `input data='d1'`, or `input data='d1,d2'` if it has more than one input. **has-specified-output** works in the same direction. The output will get an attribute `output of='o1'`, in which o1 is the id of the element creating the output. All this is illustrated in Figure 6.3. An element can also have multiple outputs (appear in the `output of` attribute of multiple elements). As such, arbitrary complex workflows can be defined.

Our data processing workflow (Figure 6.3) contains a single participant: a data processing application, i.e. a feature selection algorithm. It also requires an input, which will be the same dataset as before, and has a participant: an `algorithm implementation`. It can also have multiple *parameter settings* and *component settings*, which will become XML subelements.

In this case, there are two components (see the *has-component* relationships of the *feature selection algorithm* in the ontology): a feature subset evaluation function and a search algorithm. Each *component setting* has a participant




```
<dataset id="d1" name="kr-vs-kp" url="http://expdb.cs.kuleuven.be/...">
  <target_feature name="class" index="-1" />
</dataset>
<data_processing_workflow id="op1" input_data="d1">
  <data_processing_appl id="op2" input_data="d1">
    <data_processing_impl name="AttributeSelection" version="1.7"
       libname="weka"/>
    <component_setting>
      <function_appl role="feature_subset_evaluator">
         <function_impl name="CfsSubsetEval" version="1.26" />
      </function_appl>
    </component_setting>
    <component_setting>
      <search_algorithm_appl role="search_algorithm">
         <search_algorithm_impl name="BestFirst" version="1.28" />
         <parameter_setting name="D" value="1"/>
         <parameter_setting name="N" value="5"/>
      </search_algorithm_appl>
    </component_setting>
  </data_processing_appl>
</data_processing_workflow>
<dataset id="d2" name="kr-vs-kp-AttrSelection-CfsSubsetEval-BestFirst-D1-N5"
         url="http://expdb.cs.kuleuven.be/..." output_of="op1,op2">
  <target_feature name="class" index="-1"/>
</dataset>
```

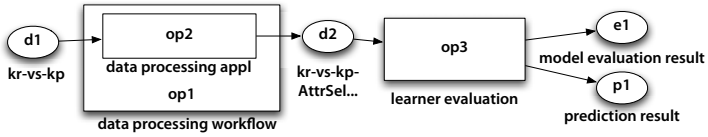


Figure 6.3: An experimental workflow (data preprocessing) in Exposé and ExpML

assumed to fulfill each of these roles. In the ontology, a *realizes* relationship indicates which processes can fulfill them. The first is a *function application*, hence the inclusion of a `function appl` element in the XML code of `component setting`, with a `role` attribute signaling the role it is fulfilling. It also has a participant: a `function impl`.
The second is a `search algorithm appl` that, next to the implementation, also has some parameter settings, as indicated in the XML code. Note that we make a small exception here: normally, a `parameter impl` subelement should be included in the `parameter setting`. Still, as an algorithm will only use its own parameter implementations, we chose for a more compact and more readable representation, in which simply the name is added as an attribute.
The resulting dataset, dubbed 'd2' is then described in the last lines of XML code in Figure 6.3, stating that it is the output of the data processing application 'op2' and of the entire data processing workflow 'op1'.

6.2.4 Experiment Evaluation and Results

As shown in the second part of the workflow at the bottom of Figure 6.3, this datasets serves as the input for a learner evaluation, which will in turn produce a model evaluation result and a prediction result.
The ExpML description is shown in Figure 6.4, and the corresponding ontological concepts in Figure 6.5. It consists of a learning algorithm application complete with parameter and component settings (in this case including a base learner application with its own parameter settings), a performance estimation application (in this case 10-fold cross-validation) and a list of evaluation functions to assess the produced models, pointing to their precise implementation. This level of detail is important to assess whether the results of this evaluation can be confidently reused. Indeed, there are many pitfalls associated with the statistical evaluation of algorithms (Salzberg 1999; Bradford and Brodley 2001; Demsar 2006; Dietterich 1998). By stating exactly which techniques were used, it will be easy to query for trustworthy results. For instance, when comparing algorithms with cross-validation, it is important that the same folds are used for all algorithms. By stating the cross-validation implementation and the random seeds used in this experiment, it is possible to check if this is the case, and also to rebuild those exact same folds for further experimentation. Although not shown here, it is also possible to define each fold as a dataset (with a download url), and to state these datasets in the performance estimation application description as components fulfilling a 'cross-validation fold' role.
The output of the experiment is shown next, consisting of all evaluation results (also stating the machine used in order to interpret cpu time) and all predictions, including the probabilities for each class. Storing predictions is especially useful if we want to apply new evaluation metrics afterwards without rerunning all prior experiments. We do not provide a format to describe models, as there already exist such formats (e.g. PMML).

```
  <learner_evaluation id="op3" input_data="d2" series="exp1"
        experiment_id="445080">
    <learner_appl id="op4">
      <learner_impl name="Bagging" version="1.31.2.2" libname="weka"/>
      <parameter_setting name="P" value="100"/>
      <parameter_setting name="I" value="10"/>
      <parameter_setting name="S" value="1"/>
      <component_setting>
        <learner_appl role="base-learner">
          <learner_impl name="REPTree" version="1.19.2.2" libname="weka"/>
            <parameter name="M" value="2"/>
            <parameter name="V" value="0.0010"/>
            <parameter name="N" value="3"/>
            <parameter name="S" value="1"/>
            <parameter name="L" value="-1"/>
        </learner_appl>
      </component_setting>
    </learner_appl>
    <performance_estimation_appl id="op5">
      <performance_estimation_impl name="Evaluation.crossValidateModel".../>
      <parameter_setting name="numfolds" value="10"/>
      <parameter_setting name="random" value="1"/>
    </performance_estimation_appl>
    <model_evaluation_function_appl name="predictive_accuracy">
      <model_evaluation_function_impl name="Evaluation.pctCorrect".../>
    </model_evaluation_function_appl>
    ...
  </learner_evaluation>
  <model_evaluation_result  id="e1" output_of="op3">
    <machine>vic_ni-09-10</machine>
    <evaluation name="build_cputime" value="2.25"/>
    <evaluation name="build_memory" value="36922896"/
    <evaluation name="mean_absolute_error" value="0.01712292"/>
    <evaluation name="root_mean_squared_error" value="0.08567175"/>
    <evaluation name="relative_absolute_error" value="0.03431352"/>
    <evaluation name="root_relative_squared_error" value="0.17151280"/>
    <evaluation name="predictive_accuracy" value="0.99123905"/>
    <evaluation name="kappa" value="0.98243855"/>
    <evaluation name="confusion_matrix" value="[[won,nowin],[1660,19],[9,1508]]"
        />
    <evaluation name="precision_array" value="[[won,nowin],[0.9887,0.9941]]"/>
    <evaluation name="recall_array" value="[[won,nowin],[0.9946,0.9876]]"/>
    <evaluation name="fmeasure_array" value="[[won,nowin],[0.9916,0.9908]]"/>
    <evaluation name="precision_mean" value="0.991376"/>
    <evaluation name="recall_mean" value="0.991082"/>
    <evaluation name="fmeasure_mean" value="0.99122"/>
    ...
  </model_evaluation_result>
  <prediction_result id="p1" output_of="op3">
    <prediction instance="0000" value="won">
      <probability outcome="won" value="0.98589249"/>
      <probability outcome="nowin" value="0.0141075011"/>
    </prediction>
    ...
    <prediction instance="3195" value="nowin">
      <probability outcome="won" value="0.0101243029"/>
      <probability outcome="nowin" value="0.98987569"/>
    </prediction>
  </prediction_result>
</expml>
```

Figure 6.4: An experimental workflow (setup and results) in ExpML

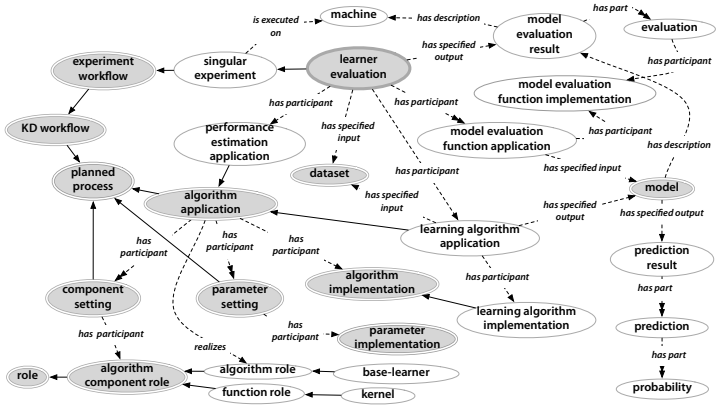


Figure 6.5: An experimental workflow (setup and results) in Exposé

```
<composite_experiment id="exp1">
  <author_details>...</author_details>
  <bibliographic_reference>...</bibliographic_reference>
  <task>classification</task>
  <experimental_design name="one-factor-at-a-time">
    <variable type="controlled" name="P">20, 30, 40 ... 100</variable>
    <variable type="controlled" name="I">10, 20, 40 ... 160</variable>
    <variable type="controlled" name="base_learner">All non-ensemble
        classification algorithms with default parameters.</variable>
    <variable type="uncontrolled">machine</variable>
   </experimental_design>
  <goal>...</goal>
  <hypothesis>...</hypothesis>
  <conclusion>...</conclusion>
</composite_experiment>
```

Figure 6.6: Context of an empirical study.

6.2.5 Experiment Context

Finally, next to individual results, the context in which scientific investigations are conducted is also of great use to correctly interpret their outcome. An example description is shown in Figure 6.6 and the corresponding part of the ontology can be found in Figure 5.7. Note that is has an id which is referenced in the `learner evaluation` in Figure 6.4. We don't nest all *singular experiments* in one element since this would inhibit parallel execution of learner evaluations. First of all, it covers the experimental setup including a specific experimental design (automated or involving humans) and the considered (un)controlled and (in)dependent variables. Second, bibliographical information is stored to cross-reference experiments and publications, so that, given a publication, we can easily locate the associated experiments. On a final note, it is also worthwhile to state the goals, hypotheses and conclusions of a study, if only to look up

whether a certain topic has been investigated before. This is especially useful for small-scale studies or studies with negative results, which are nearly impossible to publish in most sciences, but can be freely published as ExpML descriptions and stored in public experiment databases. Submitting them to an experiment database provides a way of saying "I tried this because I thought it was interesting or made perfect sense, but it didn't work". The conclusions could provide an explanation why the approach did not work, so that later researchers do not venture down the same blind alleys.

6.3 Conclusions

Returning to the the conceptual framework of experiment databases depicted in Figure 4.4, it is clear that their successful realization hinges on practical formal description languages for machine learning experiments. They are key in allowing experiments to be freely shared and reused within the machine learning community, and in enabling data mining tools to automatically stream experiments over the network to other software agents or central experiment repositories. In Section 4.4.2, we demanded that such a language be very flexible: it should capture the basic structure of machine learning experiments, yet allow each element (e.g. algorithms, kernels, datasets, evaluation metrics,...) to be described further to define the exact setup and cover task-specific properties. In this chapter, we have hurdled this challenge by casting the ontological vocabulary developed in the previous chapter into a formal, XML-based language dubbed ExpML. By consistently translating the involved concepts and their relationships to XML elements and syntax, we obtain a flexible language that describes experiments as workflows and defines the exact, arbitrarily complex composition of all involved operators. Moreover, any ontological extensions towards other types of experiments can be translated in the same fashion, thus allowing ExpML to adapt quickly to other subfields or changing needs.

We have illustrated this translation process and the flexibility of the ensuing language by using a real experiment as it was run, described and stored in our experiment database. Moreover, a set of guidelines was presented about the minimal amount of information that should be included to make sure the experiments are reproducible. In our XML definition, these are listed as required attributes. As such, any submitted ExpML files can be automatically verified for completeness. A formal definition of the language in the XML Schema Document (XSD) format is available online (**http://expdb.cs.kuleuven.be**).

Next to making sure that experiments are reproducible, 'publishing' them in ExpML adds real value to experimental results: they become part of a large collective memory of empirical observations that can be tapped into by researchers all over the world to speed up research and collaborate dynamically. Exactly how all these ExpML descriptions can be organized for quick and efficient analysis will be shown in the next chapter.

> The men of experiment are like the ant, they only collect and use; the reasoners resemble spiders, who make cobwebs out of their own substance. But the bee takes a middle course: it gathers its material from the flowers of the garden and of the field, but transforms and digests it by a power of its own.
>
> *Francis Bacon*

Chapter Seven

Anatomy of an Experiment Database

In the previous chapter, we have introduced ExpML, a formal description language that allows data mining tools to automatically send all performed experiments onto the network in a unified fashion. With the Internet now teeming with data mining experiments, the final step is to gather and organize all this empirical and theoretical information in searchable *experiment databases*. In doing so, each singular experiment becomes a point in the space of all possible experiments, immediately linked to all known theoretical information about its components. We can then explore this space to locate, rearrange, and reuse experiments of interest in many future studies. The ensuing benefits for machine learning research were already discussed in detail in Chapter 4, so we will not repeat them here.

In this chapter, we discuss how to design such experiment databases in standard relational database systems, and introduce a pilot implementation. Since every successful database design starts with a good conceptual model of the data that is to be stored, we will again start from the Exposé ontology, described in Chapter 5, to define a detailed database model for classification experiments that allows to write queries about the many aspects of learning behavior that are covered by that ontology. The range of questions that can be answered in this fashion, in just one or perhaps a few queries, will be amply illustrated in Chapter 9. Moreover, as with ExpML, this coupling will allow future refine-

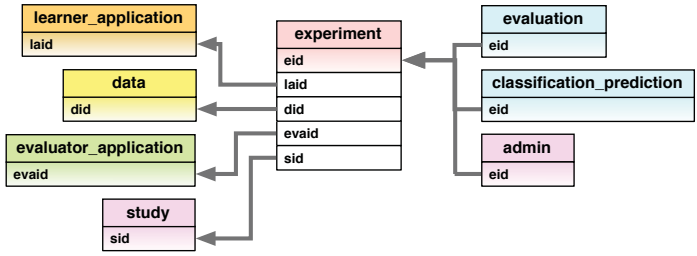


Figure 7.1: The experiment table in the experiment database.

ments and extensions of the ontological domain model (e.g. to other tasks) to be translated more easily into refinements of the database, or to develop new databases optimized for other types of experiments.
In the remainder of this chapter, we will first provide some concise guidelines on how to translate ontologies into relational database models and apply them to our Exposé ontology to design a pilot experiment database for classification in Section 7.1 . In Section 7.2, we fill this database with algorithms, datasets, and a myriad of classification experiments. Section 7.3 concludes.

7.1 From Ontology to Database Model

Databases are typically designed based on a conceptual entity-relationship model that models *entities*, *attributes* and *relationships* which are then converted into tables, columns and relations between tables. While ontological domain models are much more expressive, and are more suitable for the conceptual modeling of scientific domains in high detail, unfortunately, there is no straightforward way to convert them into relational database models. Still, there do exists some general guidelines (El-Ghalayini et al. 2007), which we shall apply here, resulting in a database model for our pilot experiment database. We will subsequently cover the parts of the database used for storing experiments, learning algorithms, datasets (and data processing workflows), evaluation techniques, outputs (evaluations and predictions) and finally the context and execution of experiments.

7.1.1 Experiments

The database model is necessarily dependent on the kind of experiment that needs to be stored (although multiple types can be stored in the same database). In our pilot experiment database, we focus on *learner evaluations*, yielding a single table simply called `experiment`. Figure 7.1 shows that, as in the corre-

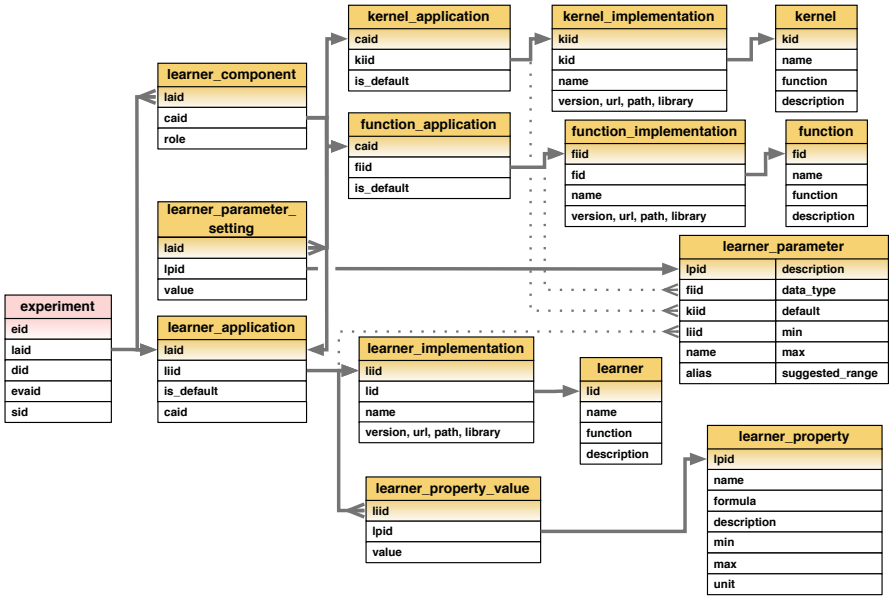


Figure 7.2: Learning algorithms in the experiment database.

sponding part of Exposé shown in Figure 5.5, such an experiment has the participants *learner application*,*performance estimation application* (shortened to *evaluator application*) and *model evaluation functions*, the latter of which is better linked to the *evaluation* table (see table `evaluation metric` in Figure 7.5). An experiment also has an input dataset (simply called `data`), and two outputs: *model evaluation function applications* (here: `evaluations`) and *models*, from which we store the *prediction result* (here: `classification predictions`). It is in itself part of a *composite experiment*, called `study`. A final table contains some administrative information about the experiment execution (e.g. the machine on which it ran).
As such, every experiment (and its outcome) becomes a point in the space spanned by all learner applications, datasets and evaluator applications, which can be explored simply by adding or dropping constraints about these dimensions.

7.1.2 Learning Algorithms

With the basic structure set, we can start to translate more specific parts of the ontology, starting with the composition of learning algorithm applications, shown in Figure 5.12. We translate all the relationships of the concept *algorithm application* (learning algorithm application's parent class), the result of which

is shown in Figure 7.2. While many other aspects, such as database performance, influence our design decisions, we mostly used the following guidelines (El-Ghalayini et al. 2007):

- Classes can be mapped to entities (tables), under some constraints: the table should also contain some meaningful attributes or relations. Moreover, one has to define a useful level of abstraction. Since it would be counterproductive to have separate tables for each type of algorithm defined in the ontology (indeed, algorithm comparisons would become very laborious), we only withheld tables at the level of learning algorithms (application, implementation, and general specification), as shown in Figure 7.2. The names of specific types of learning algorithms will however still occur in the database as *rows* in a table (see table `learner` in Figure 7.2) and used in queries. Likewise, many of the more specific concepts will become rows in a table that lists specific elements of that type, for instance, evaluation functions (see table `evaluation metric` in Figure 7.5) and dataset properties (table `data property` in Figure 7.3). Other concepts are eschewed for being too general: we don't have a table for the top-level *quality* concept, but we do for *algorithm property* (table 'learner_property' in Figure 7.2).
- *Intrinsic property relationships*, meaning relationships with concepts that belong *only* to a certain class, are mapped to attributes (columns). As such, every *has-description* relationship to *identifiers* becomes an attribute in a database table. In Figure 7.2, *learner_implementation* gets the attributes name, version, url, and so forth. Other relationships of this kind, like has-model-structure also become attributes.
- *Emergent property relationships*, with concepts not belonging to a certain class (e.g. the *number of instances* property of datasets) become relations, or *linking tables* in the case of many-to-one relationships. For instance, the *has-description* relation between *learning algorithm implementation* and *algorithm property* becomes the `learner property value` linking table between `learner implementation` and `learner property` in Figure 7.2. The forked arrow tail indicates that there can be many `learner property value` assignments for each `learner implementation`.
- Any *has-part* or *has-participant* relationship in the ontology becomes a relation between database tables. In fact, most relationships in the database model are of this type. Moreover, *has-specified-input* and *has-specified-output* are also specific kinds of *has-participant* relationships. In Figure 7.2, a `learner application` can have multiple `learner parameter setting` participants, and a `learner parameter setting` has a `learner parameter` participant.
- Other ontological relationships simply have to be translated according to their meaning. For instance, the *is-concretization-of* relationship between a *learning algorithm implementation* and the general *learning algorithm*

also becomes a simple relationship in the database model linking algorithm implementations with their more general definitions also shown in Figure 7.2).

Reviewing the database model in Figure 7.2, we see it shows that learning algorithms are composite, parameterized objects, and that *learner application* stores the exact configuration used in an experiment. It can have multiple *learner parameter settings*, each linked to a specific *learner parameter* storing all known theoretical information about its use, such as default values and suggested value ranges. We separate between *parameter implementations* and general *parameters* through the `alias` attribute which states the general name of the parameter. The `is default` attribute flags the application with all default parameter settings. A learner application can also be composed out of `learner components`, which play a certain `role`, and can either be a learner application, kernel application or function application, each of which can in turn have its own parameter settings (and further component settings in the case of a learner). Although a kernel is actually also a function, we provide a separate table to store kernel-specific information and make queries more interpretable. All this completely defines the learning algorithm configuration: queries can select the values of all parameters and components in a learning algorithm and query constraints allow to *zoom in* on results for specific settings.
Experiment results are also linked to theoretical information about the algorithm attached to the `learner implementation`, which also links to the general `learner` and has a list of `learner properties` (e.g. its bias-variance profile or noise resilience). As said before, the one-to-many relationship calls for two tables here: one for storing the properties (e.g. their name and formula) and one for storing the value of each property.

7.1.3 Datasets and Data Processing

Figure 7.3 shows how datasets and data processing workflows are stored, following the parts of the ontology shown in Figure 6.2 (structure and properties) and Figure 6.3 (data processing). `Data` represents any dataset, original or preprocessed by (the output of) a `preprocessing step`. The latter is a linking table necessary because multiple datasets can be required as input for a `preprocessor application`. A (data processing) `workflow` is a named collection of preprocessing steps. Since preprocessors are algorithms, they also have an implementation and can also use a series of parameter settings. We can thus store the exact workflows used to generate a certain dataset, and query for datasets preprocessed in a specific way.
Theoretical information about the datasets is also attached, and thus linked to all experiments using them. First, a dataset contains multiple data features (some of which can be targets), described by their name and data type, and both entire datasets and data features can have multiple theoretical data properties.

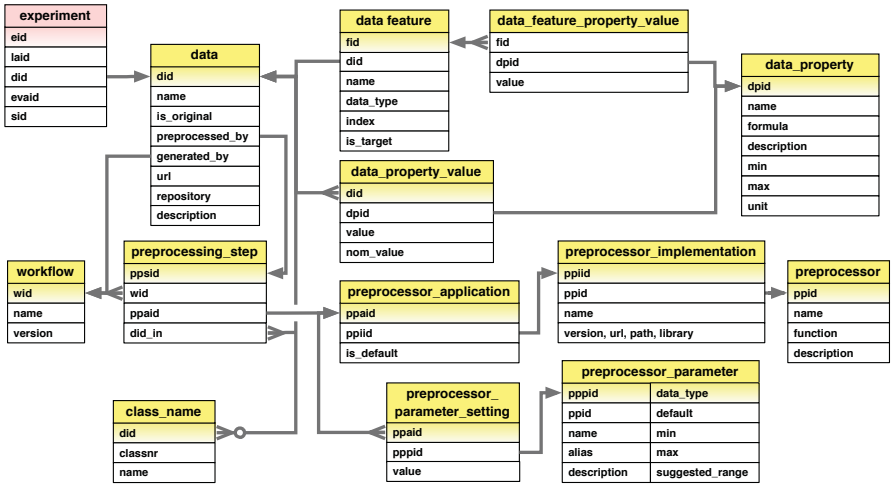


Figure 7.3: Datasets and data processing in the experiment database.

7.1.4 Evaluation Techniques

Performance estimation applications, shortened to `evaluator applications`, such as 10-fold cross-validation, are stored very similarly to learning algorithms, as shown in Figure 7.4. Like all algorithms, they can have a range of parameter settings and contain an implementation which, in turn, concretizes a more general `evaluator`.

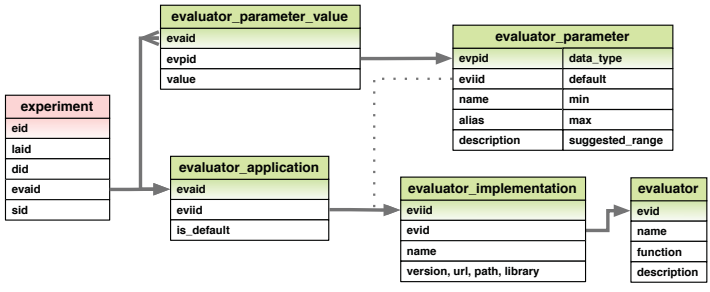


Figure 7.4: Performance estimation techniques in the experiment database.

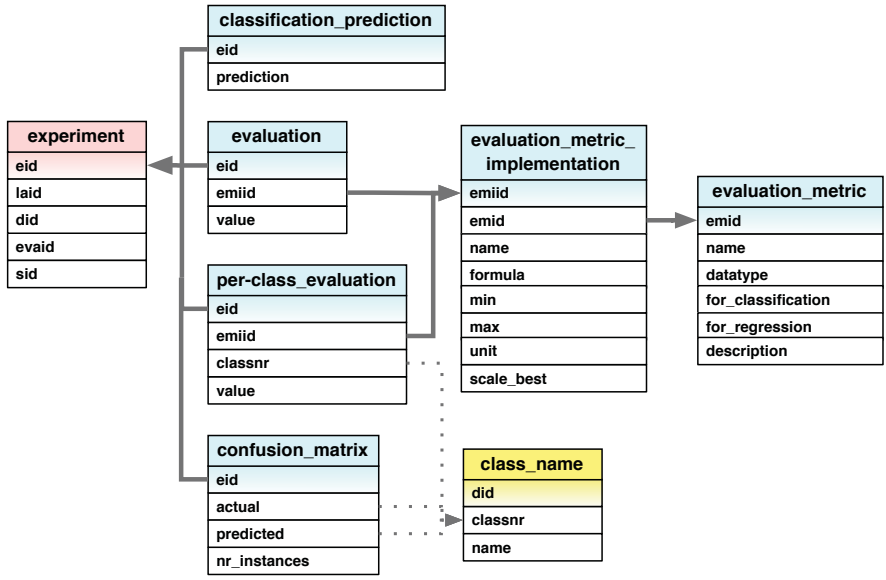


Figure 7.5: Experiment outputs in the experiment database.

7.1.5 Outputs: Evaluations and Predictions

Figure 7.5 shows the outputs of a learner evaluation experiment. First, we store all (probabilistic) classification predictions, but because of their sheer volume, we save them as a compressed string. This means they cannot be directly queried, but they can be expanded in the software interface (see Chapter 8). Next, we have three tables for three kinds of evaluations. The *model evaluation function application* concept in Figure 5.5 is represented in three tables depending on the structure of the result. The first is the standard `evaluation`, in which case the result consists of a single value. It is linked to the `evaluation metric implementation` function that computed it, which is in turn linked to the general `evaluation metric` (e.g. predictive accuracy). The second kind is an array of per-class evaluations of binary evaluation metrics, each time assigning one class to be positive and the remaining ones to be negative, as discussed in Section 5.3.4. The third kind is the complete confusion matrix, also previously discussed in Section 5.3.4.

7.1.6 Context and Execution

Finally, Figure 7.6 shows a table containing information about the execution of the experiment as well as the *study* (*composite experiment*) to which a submitted learner evaluation belonged. The corresponding part of the ontology

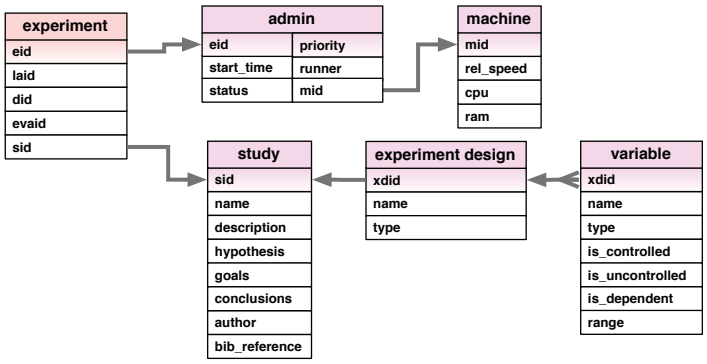


Figure 7.6: Experimental context and execution data.

is shown in Figure 5.7. A study contains the context in which a group of experiments was executed, including textual descriptions of the hypotheses, goals and conclusions of the study. It can also contain a specific `experiment design` and the list of variables and their range of values that were varied in the design. One can also search for submitted experiments associated with published papers or a certain author.

7.2 Populating the Database

As we want to use this database to gain insight into the behavior of machine learning algorithms under various conditions, we first need to populate it with very diverse experiments.

First, we need to add the necessary implementations and datasets. We selected 54 well-known classification algorithms from the WEKA platform (Hall et al. 2009) and inserted them together with detailed descriptions of all their parameters (e.g. sensible value ranges). We also added a range of simple algorithm properties (e.g. the representation model used). Next, we added a cross-validation procedure and two bias-variance decomposition implementations (Kohavi and Wolpert 1996), also from WEKA, as well as 45 implementations of evaluation measures. Concerning datasets, 149 commonly used datasets were taken from the UCI repository and inserted together with 56 data characteristics, calculated for each dataset. Finally, two data preprocessor implementations were added: feature selection with Correlation-based Feature Subset Selection (Hall 1998), and a subsampling procedure.

To generate a sample of classification experiments that covers a wide range of conditions, while also allowing to test the performance of some algorithms under very specific conditions, three series of experiments were performed, in

which a number of algorithms were explored more thoroughly than others:

- A first series of experiments simply varied the chosen algorithm, running them with default parameter settings on all datasets.
- To study parameter effects, a second series varied the parameter settings, with at least 20 different values, of a smaller selection of popular algorithms: SMO (a support vector machine (SVM) trainer), MultilayerPerceptron, J48 (C4.5), 1R, RandomForest, Bagging and Boosting. Next to parameter settings, component settings were are also varied: two different kernels were used in the SVM, with their own range of parameter settings, and all non-ensemble learners were used as base-learners for ensemble learners. In the case of multiple varied parameters, we used a one-factor-at-a-time design: each parameter is varied in turn while keeping all other parameters at default.
- Finally, a third series of experiments used a random sampling design to uniformly cover the entire parameter space (with at least 1000 settings) of an even smaller selection of algorithms: J48, Bagging and 1R.

All parameter settings were run on all datasets, and for all randomized algorithms, each experiment was repeated 20 times with different random seeds. More detailed information on exactly which parameter settings were used can be found in the stored experiment context descriptions.
As for the evaluation procedure, all experiments were evaluated with 10-fold cross-validation, using the same folds on each dataset, using all 45 evaluation metrics. A large portion was additionally evaluated with a bias-variance analysis.
In total, the experiment database currently contains over 650,000 experiments and is in itself a contribution to the machine learning community. It can be publicly accessed online using various interfaces, as shall be shown in the next chapter.

7.3 Conclusions

In this chapter, we have investigated how machine learning experiments can be intelligently organized in searchable experiment databases. Starting from the Exposé ontology, we derived a detailed database model for classification experiments that allows us to write queries about the many aspects of learning behavior that are covered by the ontology. In this database model, each singular experiment becomes a point in the space of all possible experiments, immediately linked to all known theoretical information about its components. By querying it, we can quickly locate and rearrange thousands of experiments, and reuse them in many future studies.
More specifically, the database is designed in such a way that we can:

- explore the space spanned by all learner applications, evaluator applications, datasets and the evaluation outcome simply by adding or dropping constraints about these dimensions in queries.
- write queries to select the values of specific parameters and internal components of complex learning algorithm configurations. Conversely, by adding constraints on these internal components we can *zoom in* on experimental results obtained with very specific algorithm configurations.
- store the exact workflows used to generate a specific dataset, and query for datasets preprocessed in a specific way.
- link all empirical results generated by experiments to stored theoretical information about the learning algorithms and datasets (or single data features).
- search for submitted experiments associated with published papers or a certain author.

Finally, since we want to use this database to gain insight into the behavior of machine learning algorithms under various conditions, we populated it with a myriad of very diverse experiments on many algorithms and many datasets.

Conclusions Part I

In this part of the thesis, we have set out to organize the world's machine learning information, or at least a portion of it. Through a *community-based approach to machine learning experimentation*, we aim to create a collaborative workspace in which researchers can freely exchange experimental results. In a first step to this ideal, we offer tools that allow the massive streams of experiments that are being executed to evaluate learning algorithms to be automatically shared and gathered into open repositories, *experiment databases*, that automatically organize all the meta-data so it becomes easily accessible to all, and can be investigated in depth simply by writing queries.
We can summarize the benefits of this approach as follows:

Reproducibility The database stores all details of the experimental setup, thus attaining the scientific goal of truly reproducible research.

Reference All experiments, including algorithms and datasets, are automatically organized in one resource, creating a useful 'map' of all known approaches, their properties, and results on how well they fared on previous problems. This also includes *negative results*, which usually do not get published in the literature.

Visibility New learning algorithms will pop up in relevant queries.

Reuse It saves time and energy, as previous experiments can be readily reused.

Generalizability It enables larger and more generalizable studies. Studies covering many algorithms, parameter settings and datasets are hugely expensive to run, but become much more feasible if a large portion of the necessary experiments are available online.

Integration The formalized descriptions of experiments also allow the integration of such databases in data mining tools, for instance, to automatically log and share every experiment in a study or to reuse past experiments to speed up the analysis of new problems.

In **Chapter 4**, we started off by outlining the need for greater reproducibility, generalizability and interpretability in machine learning research. To learn from previous practical applications of experiment repositories, we looked to so-called e-Sciences, where they have, in some form or another, been applied successfully for some time now, particularly in bioinformatics, astrophysics and high-energy physics. From this, we learned that they all seem to have evolved towards online, public infrastructures for experiment exchange, using more or less the same three components: ontologies, formal experiment description languages, and searchable repositories.

Additional considerations specific to machine learning led to a conceptual framework for collaborative experimentation in machine learning.
In **Chapter 5**, we have established **Exposé**: an ontology for experimentation in machine learning. It covers a large portion of the involved concepts and their relationships, thus providing a core vocabulary on which we can build common, extensible representation languages and database models.
While focussing on supervised classification and propositional datasets, the resulting ontology is quite extensive. It treats experiments as workflows to allow the definition of various types of experiments and includes the possibly extensive data processing workflows that precede them. It also covers the experiment context in which individual learning algorithm evaluations are performed, the various learner evaluation measures and performance estimation techniques that are used in current research, the structure and properties of datasets and of learning algorithms. The latter include their parameters, internal components such as kernels, base-learners and distance functions, and their internal learning mechanisms.
In **Chapter 6**, we have cast the ontological vocabulary developed in the previous chapter into a formal, XML-based language, dubbed **ExpML**. By consistently translating the involved concepts and their relationships to XML elements and syntax, we obtain a flexible language that describes experiments as workflows and defines the exact composition of all involved operators. Moreover, any ontological extensions towards other machine learning tasks and other types of experiments can be translated in the same fashion, thus allowing ExpML to adapt quickly to other subfields or changing needs.
We have illustrated this translation process and the flexibility of the ensuing language by using a real experiment as it was run, described and stored in our experiment database. Moreover, a set of guidelines was presented about the minimal amount of information that should be included to make sure the experiments are reproducible. As such, any submitted ExpML files can be automatically verified for completeness.
In **Chapter 7**, we have investigated how machine learning experiments can be intelligently organized in searchable experiment databases. Starting from the Exposé ontology, we derived a detailed database model for classification experiments that allows to write queries about the many aspect of learning behavior that are covered by the ontology. In this database model, each singular experiment becomes a point in the space of all possible experiments, immediately linked to all known theoretical information about its components. By querying it, we can quickly locate and rearrange thousands of experiments, and reuse them in many future studies.
Finally, since we want to use this database to gain insight in the behavior of machine learning algorithms under various conditions, we populated it with a myriad of very diverse experiments on many algorithms and many datasets.

Part II

Learning From the Past

Outline Part II

While in the previous part of this thesis, we described how to organize large amounts of empirical machine learning results in experiment databases, this part is concerned with using these experiment databases to learn from past experiments and gain new insights.
First, Chapter 8 describes how to access the experiment database. Two separate interfaces will be defined:

- A software interface that can be used by data mining tools to interact with locally set up experiment databases as well as online, global experiment repositories. We will also use one popular data mining toolbox, WEKA, and extend it to import ExpML experiment setups, run them, and return the results.
- An intuitive query interface that allows users to easily compose database queries and visualize the returned results. An online web interface and a desktop explorer tool are shown, and the use of an intuitive graphical point-and-click interface is demonstrated.

Next, Chapter 9 will use the experiment database described in Chapter 7 and the query tools in the previous chapter to evaluate how easily the results of previously stored experiments can be exploited for the discovery of new insights into a wide range of meta-learning questions, as well as to verify a number of recent studies.
More specifically, we distinguish between three types of studies, increasingly making use of the available meta-level descriptions, and offering increasingly generalizable results:

1. Model-level analysis. These studies evaluate the produced models through a range of performance measures, but consider only individual datasets and algorithms. They identify HOW a specific algorithm performs, either on average or under specific conditions.
2. Data-level analysis. These studies investigate how known or measured data properties, not individual datasets, affect the performance of algorithms. They identify WHEN (on which kinds of data) an algorithm can be expected to behave in a certain way.
3. Method-level analysis. These studies don't look at individual algorithms, but take general properties of algorithms (e.g. their bias-variance profile) into account to identify WHY an algorithm behaves in a certain way.

Simplicity is the ultimate sophistication.

Leonardo da Vinci

Chapter Eight

Interfaces: Hiding the Complexity

Returning to our conceptual framework for collaborative experimentation in machine learning in Figure 4.4, we find that we have now covered three of the five boxed components: the ontology (Exposé), experiment markup language (ExpML) and experiment database (ExpDB). Two components remain: an *API* to build uniform experiment instances out of all necessary details (and express them in ExpML), and a *query interface* that allows users to easily compose database queries and thoroughly analyze the returned data.

Indeed, while the fine-grainedness of the experiment database model (see Chapter 7) allows very flexible querying, it also means that the resulting SQL queries tend to become rather complex when different aspects of machine learning experiments (e.g. algorithms, datasets and evaluations) need to be linked together, simply because many tables need to be joined.

However, we can also use the inherent structure of the model to our advantage, and in this chapter, we will use it to design intuitive interfaces that hide this complexity from the user.

First, Section 8.1 discusses the API and how it was used to run and store all the experiments currently available in the experiment database. Next, we discuss the available database interfaces: an online web interface and a desktop explorer tool, in Section 8.2. Finally, Section 8.3 illustrates the use of a graphical query tool that allows to compose queries through an intuitive *point-and-click* interface. Section 8.4 concludes.

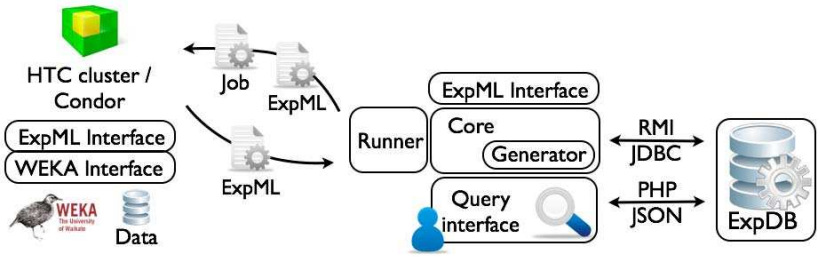


Figure 8.1: Software Components

8.1 Software Components

Figure 8.1 shows the various software components involved in using experiment databases to run and analyze machine learning experiments. Perhaps the most important component is the *ExpML Interface*, which allows other software agents to build uniform experiments as described in this thesis. The current version provides a Java API to build such experiment instances programmatically, and to import/export them from/to ExpML files or ExpDBs. The details required by the API include all information necessary to reproduce the experiment and organize its results. We won't discuss any implementation details here: JavaDocs and other documentation is provided on the database website, and the source code contains a series of working examples to illustrate its use.

The next component, the core of our system, offers various ways to manipulate experiment instances. First, it contains an experiment generator which creates new experiments according to a few implemented experiment design methods. Second, it interfaces with the MySQL implementation of our experiment database, connecting remotely to it using RMI (Remote Method Invocation): it stores new experiments and extracts any necessary information. Third is the experiment runner, which connects to high throughput computing (HTC) systems to quickly run many experiments. It sends an ExpML description of every experiment setup (without the output evaluations and predictions) together with a job description needed for the HTC job scheduling systems.

To execute the experiments, we wrote a wrapper around the WEKA toolbox, that imports the experiment in ExpML, runs it using the WEKA platform, and exports the finished experiment again in ExpML. A full integration into WEKA is also planned as future work. Finally, all finished experiments are returned to the core system and stored in the database.

The last component is the query interface, which connects to the database through a PHP server and that uses JSON (JavaScript Object Notation, a simple data-interchange format), to describe the returned results.

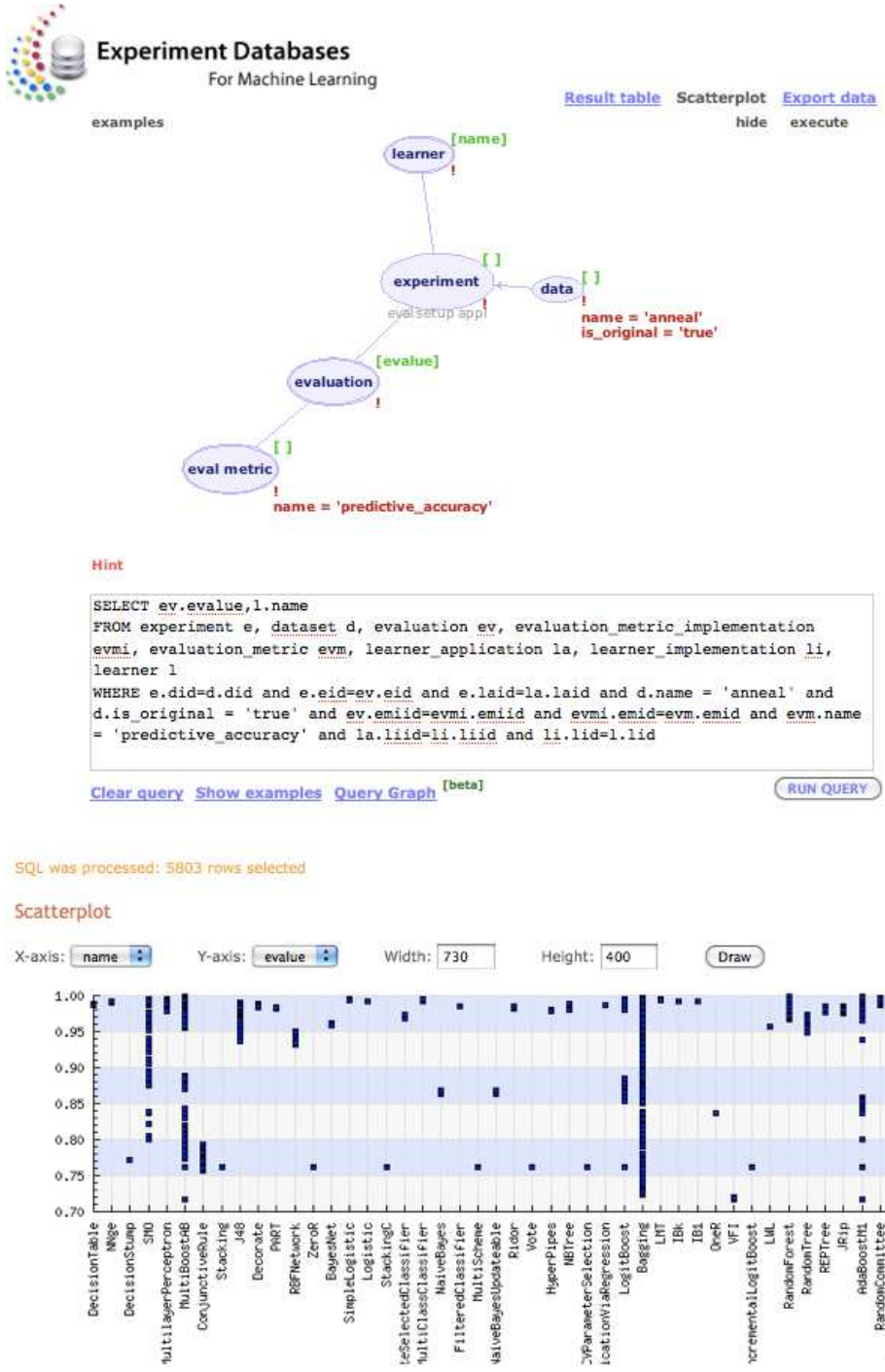


Figure 8.2: The ExpDB Web Interface

Up-to-date documentation and information about this framework can be found online at **http://expdb.cs.kuleuven.be**. This includes all source code, database schemas, OWL-DL descriptions of Exposé and XSD descriptions of ExpML, all licensed under GPLv2 or higher.[1] All this allows to quickly set up new experiment databases, and even to adapt them to specific needs.

8.2 Interfaces

The database can be accessed through two query interfaces: an online interface on the homepage itself and a desktop application. Both allow to launch queries written in SQL, or composed using the graphical query tool discussed in Section 8.3. For both methods, many examples are available in the interfaces, including the ones used in the next chapter.

8.2.1 An online interface

The online query interface, again available on **http://expdb.cs.kuleuven.be**, is shown in Figure 8.2. The top part shows the query tool which will be discussed in Section 8.3. After a query has been composed, the corresponding SQL code is automatically generated and shown in the box below. The query in question shows all predictive accuracy evaluations of all algorithms on dataset 'anneal'. While the SQL query may seem complex, it simply selects the name of the learner and the value of the evaluation and adds three constraints: that the dataset should be 'anneal', that it should be original (not preprocessed), and that the evaluation metric should be 'predictive accuracy'. The remainder of the query simply joins the necessary table together. The bottom part of Figure 8.2 shows the results. The web interface can show the results as a table or as a 2D-scatterplot, shown here. Since the query did not specify which parameter settings should be used (and did not aggregate the results in any way), one can clearly see the variation caused by the parameter settings. We will analyze such results in depth in the next chapter. Besides the result table and the scatterplot, a third option is to output the data in a certain format (e.g. csv) so it can be analyzed by other tools.

8.2.2 A desktop application

The second interface, shown in Figure 8.3, is a desktop explorer tool with additional functionality. First, the SQL query is updated in real time as the query graph is altered, and it allows users to save and load graph queries to/from a file. Second, it has additional visualizations: colorized 2D and 3D scatterplots, so that experiments (points) can be colored based on algorithm,

[1] Part of desktop query interface uses code of the KNIME project (www.knime.org), and is licensed accordingly.

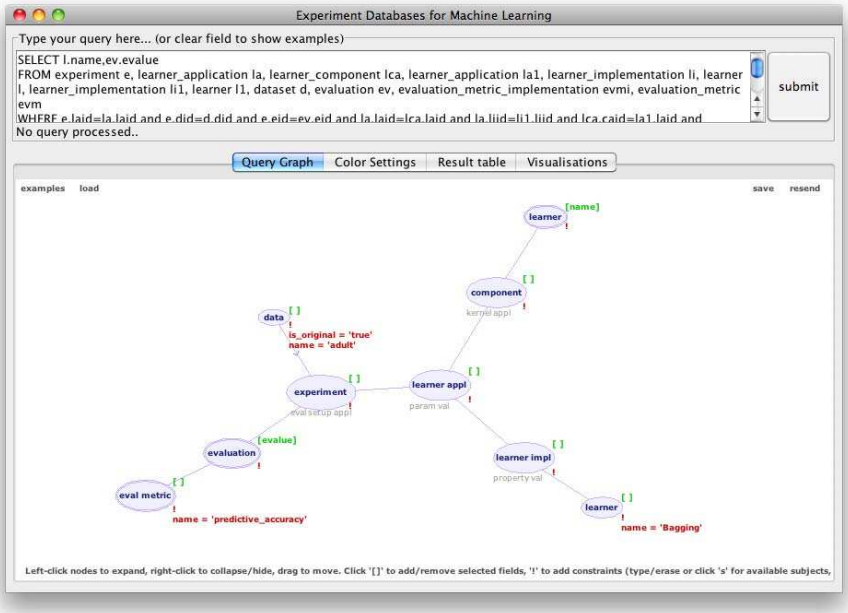


Figure 8.3: The ExpDB Explorer Tool

dataset, or any other feature, as well as Self Organizing Maps (SOMs) (Kohonen 1998): a clustering method that builds a two-dimensional map of the clusters. SOMs can, for instance, be used to discover clusters of datasets that are similar to each other based on their meta-features (Smith et al. 2002).

8.3 The graphical query tool

When one is unfamiliar with the structure of the experiment database, the best way to query it is the graphical query tool. The best way to understand how it works is simply to use it, although we'll provide some guidance here as well. As shown in Figure 8.4, it starts out with a simple graph showing the components of an experiment. It is thus similar to Figure 7.1, but with a few simplifications. First, nodes in double ellipses are 'stacked', which means that there are other, more specific nodes beneath it. For instance, in the case of *learner* there is a *learner implementation* and *learner application* node underneath that do link directly to *experiment*. They thus hide the details that may not be needed at first, and allow to expand the query exactly as much as is needed to compose the query. Second, 'output' is an artificial node that hides the many possible outputs so as not to overload the graph.

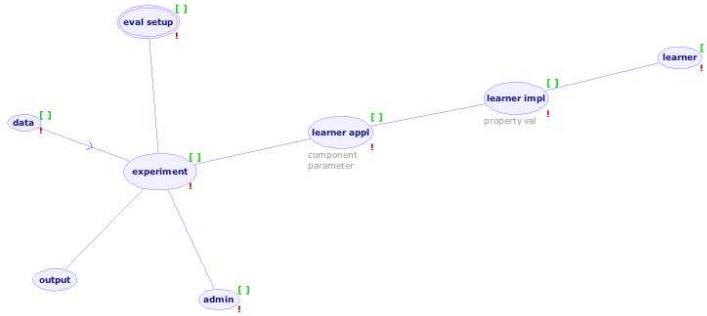
Figure 8.4: Collapsed query graph

8.3.1 Expanding the graph

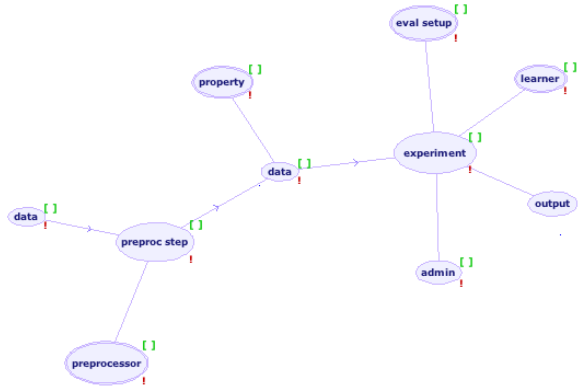
Clicking on a node expands all its children, or when it is stacked, the stack is laid open. Conversely, right-clicking a node collapses it. A simple physics engine lays out the new graph automatically, although nodes can be manually dragged as well. Figure 8.5(a) shows what happens if *learner* is clicked: the *learner implementation* and *learner application* are shown, and below them, there are *hints*: a list of children that may be of interest. In the case of the *learner implementation*, it indicates that learning algorithm *properties* are connected to the implementation, and in the case of the *learner application*, it shows that it can be expanded to show component and parameter settings.

Figure 8.5(b) shows the expansion of the *data* node. Next to a node storing *data properties*, it also allows to specify data workflows: the original dataset becomes the output of a *preprocessing step*, which takes another dataset as input. Input and output relations are shown by arrows on the edges. The preprocessing step links to a *preprocessor*, which can be further expanded to reveal the implementation and parameter settings. Click *preprocessor step* again to bring up a second, third,... input dataset.

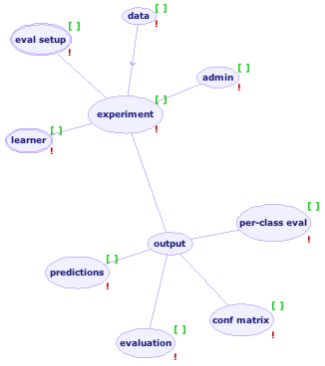
Figure 8.5(c) shows the expansion of the *output* node: it reveals four types of output: evaluations, per-class evaluations, confusion matrices and predictions, as previously shown in Figure 7.5. If one type of output is selected, the remaining ones and the output node disappear. Clicking *evaluation* yields the graph shown in Figure 8.5(d), revealing an *evaluation metric* that allows to choose a certain evaluation measure. Clicking *evaluation* a second time brings the output node back again.



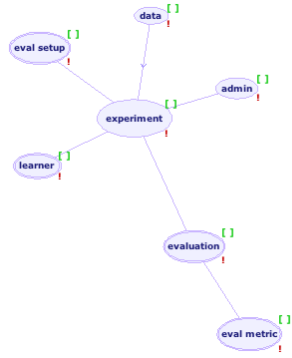
(a) Learning algorithm expansion



(b) Data processing expansion

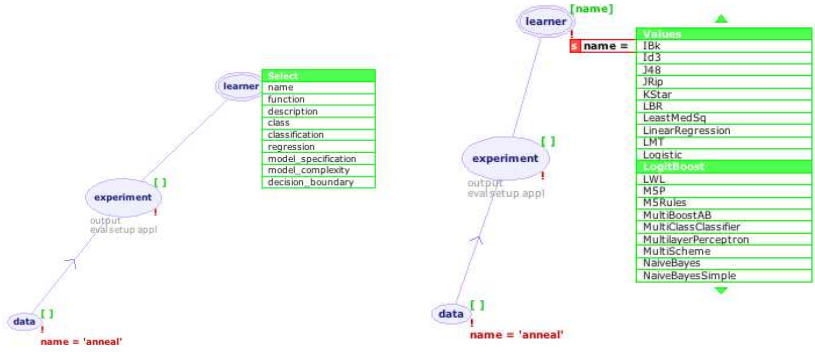


(c) Output expansion



(d) Evaluation selection

Figure 8.5: Expanding the query graph



(a) Selecting attributes (b) Composing constraints

Figure 8.6: Selecting attributes and composing constraints

8.3.2 Selecting attributes and composing constraints

Figure 8.4 shows that green brackets and red exclamation marks appear next to each node. The green brackets are used to select attributes. As shown in Figure 8.6(a), clicking the brackets brings up a list of all attributes (a quick query is sent to the database to retrieve it). Clicking an attribute adds it to the list of attributes selected by the query, as shown in **[name]** in Figure 8.6(b). Selecting the same attribute twice removes it again.

To compose constraints, a click on the red exclamation mark (Figure 8.6(b)) brings up a red-bordered field in which the constraint can be typed. Still, as most constraints restrict the values of attributes, the constraint box is preceded with an 's' and succeeded with a 'v' button. Clicking the 's' button again brings up a list of attributes to select from, and clicking 'v' brings up a complete, scrollable list of attribute values that appear in the database (except for floating point-typed attributes). In Figure 8.6(b), we selected 'name', and thus a complete list of all learner names available in the dataset is returned. It even checks other constraints: for instance, if a certain learning algorithm was selected, only parameters belonging to that algorithm will be returned when composing a parameter constraint on that algorithm.

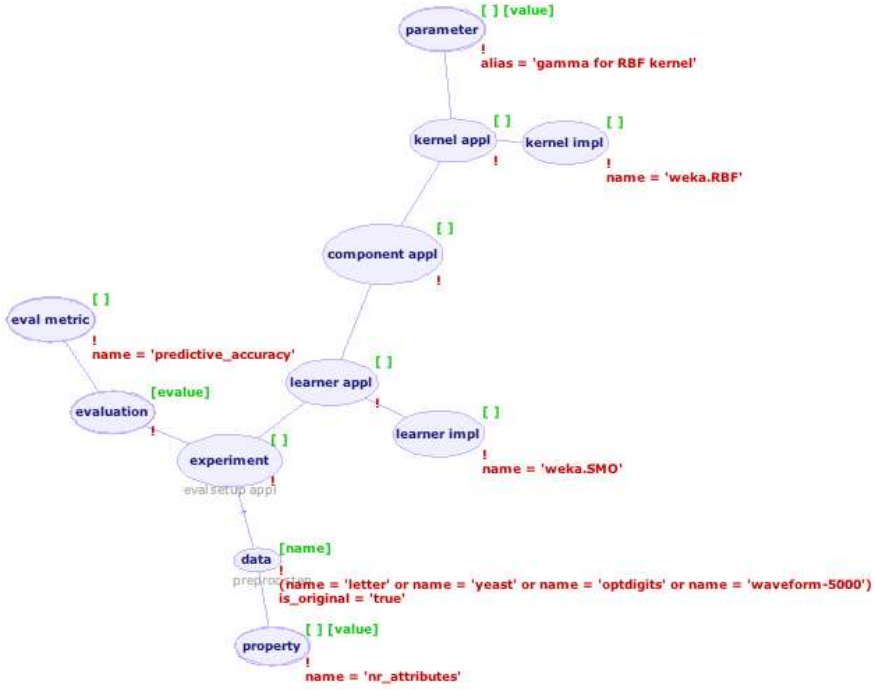
Selecting a value completes the constraint, as shown by the *name='anneal'* constraint on the bottom of Figure 8.6(b). Clicking the 'v' button again allows to select a second, third,... value, as shown at the *data* node in Figure 8.7(a). Clicking the exclamation point again allows to declare multiple constraints. As such, a wide range of queries can be intuitively composed by a few clicks in the query tool.

8.3.3 An example query

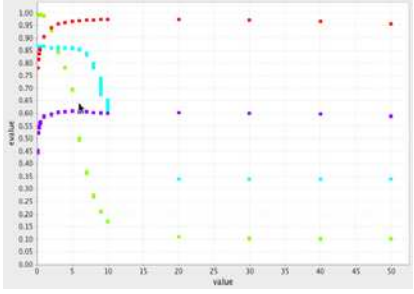
A more advanced query composed in this fashion is shown in Figure 8.7(a). It selects the value of the parameter aliased 'gamma for RBF kernel', belonging to the 'weka.RBF' kernel implementation, and used as a component in the application of the learner implementation 'weka.SMO', an SVM implementation. Next, the value of the measured predictive accuracy is selected, as well as the name of the dataset, which can be any of the stated four, but all non-preprocessed. Finally, the number of attributes in each of these datasets is requested. The double brackets with *property* (and *parameter*) indicate that the linking table (which only has a 'value' attribute) is hidden.
As such, this query shows the performance effect of the 'gamma' parameter setting of the RBF kernel on a number of datasets. When run in the desktop interface, it can be visualized either as a 2D scatterplot, showing the parameter effect on each dataset in a different color (Figure 8.7(b)), or as a 3D scatterplot in which the number of attributes is the third dimension. The interpretation of these results will be resumed in Section 9.1.2.
While this is one example of the use of learning algorithm components in queries, the query in Figure 8.3 is a second: it compares the performances of all possible base-learners in a Bagging ensemble on the 'adult' dataset.
Note that this graphical querying tool still has its limitations: for instance, it currently can't perform outer joins or include aggregation functions, but it does help to write the basic (sub)queries, and the SQL code it generates can always be refined manually afterwards.

8.4 Conclusions

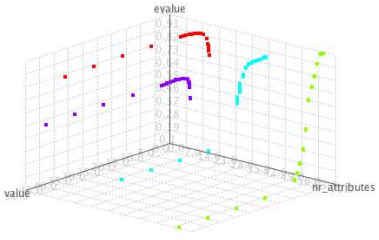
In this chapter, we have covered the remaining two components of our framework for collaborative experimentation in machine learning.
The first is a software interface that can be used by data mining tools to programmatically export experiments to ExpML files, or to immediately store them in an experiment database. We have used this interface to write a wrapper for the WEKA toolbox that accepts ExpML descriptions of experimental setups (without the outputs), runs the experiment and returns the finished experiment again in ExpML. Moreover, this has enabled us to easily parallelize the execution of experiments on high performance computing systems.
The second component is an intuitive query interface that allows users to easily compose database queries and thoroughly analyze the returned results. Two such interfaces are available: an online web interface and a desktop explorer tool. Both contain visualization tools to analyze the retrieved data, and both offer an intuitive graphical point-and-click interface that allows to compose queries simply by selecting the appropriate attributes and assembling the right constraints.



(a) A more advanced query



(b) 2D scatterplot



(c) 3D scatterplot

Figure 8.7: Visualizations in the query interface

Somewhere, something incredible is waiting to be known.

Carl Sagan

Chapter Nine

Exploring learning behavior

In this chapter, we use the experiment database described in Chapter 7 and the query tools in the previous chapter to evaluate how easily the results of previously stored experiments can be exploited for the discovery of new insights into a wide range of meta-learning questions, as well as to verify a number of recent studies. In doing this, we aim to take advantage of the theoretical information stored with the experiments to gain deeper insights.
More specifically, we distinguish between three types of studies, increasingly making use of the available meta-level descriptions, and offering increasingly generalizable results (a similar distinction is identified by Van Someren (2001)):

1. Model-level analysis. These studies evaluate the produced models through a range of performance measures, but consider only individual datasets and algorithms. They identify HOW a specific algorithm performs, either on average or under specific conditions.

2. Data-level analysis. These studies investigate how known or measured data properties, not individual datasets, affect the performance of algorithms. They identify WHEN (on which kinds of data) an algorithm can be expected to behave in a certain way.

3. Method-level analysis. These studies don't look at individual algorithms, but take general properties of algorithms (e.g. their bias-variance profile) into account to identify WHY an algorithm behaves in a certain way.

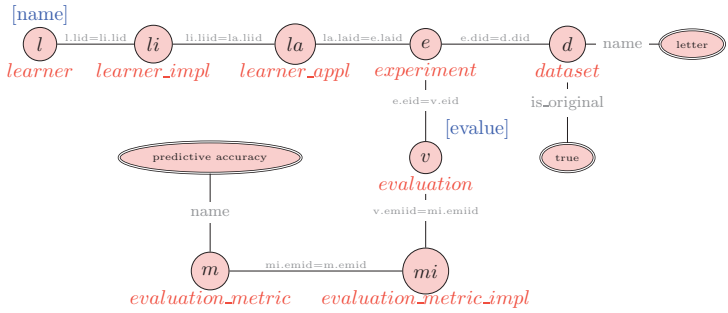


Figure 9.1: A graph representation of our query.

9.1 Model-level analysis

In the first type of study, we are interested in how individual algorithms perform on specific datasets. This type of study is typically used to benchmark, compare or rank algorithms, but also to investigate how specific parameter settings affect performance.

9.1.1 Comparing Algorithms

To compare the performance of all algorithms on one specific dataset, we have to select the name of the algorithm used and, for instance, the predictive accuracy recorded in all experiments on a certain dataset. This can be translated to SQL as shown graphically in Figure 9.1: we join the necessary tables and select (in brackets) the learner name and the value of an evaluation, adding the constraints (in ovals) that the evaluation metric should be predictive accuracy and that the dataset's name should be, for instance, 'letter'. `is_original='true'` indicates that we only want the non-preprocessed version of the dataset.
These query graphs are equivalent to the ones used in the graphical query interface in Chapter 8. Here, however, we redraw them in a more conventional way that has the added benefit that they are more easily readable.
Running the query returns all known experiment results, which are scatter-plotted in Figure 9.2, ordered by performance. This immediately provides a complete overview of how each algorithm performed. Since the generality of the results is constrained only by the constraints written in the query, the results on sub-optimal parameter settings are shown as well (at least for those algorithms whose parameters were varied), clearly indicating the performance variance they create. As expected, ensemble and kernel methods are very dependent on the selection of the correct base-learner or kernel, respectively.
We can however extend the query to ask for more details about these algorithms. Figure 9.3 shows how to append the name of the kernel and the name

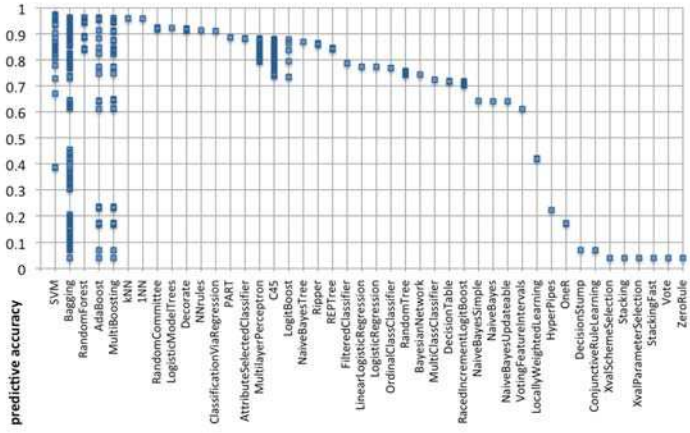


Figure 9.2: Performance of all algorithms on dataset 'letter'.

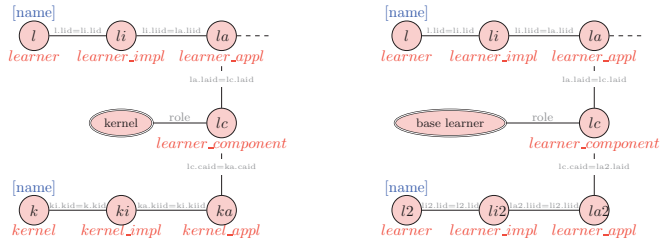


Figure 9.3: A partial graph representation of the extended query, showing how to select kernels (left) and the base-learners of an ensemble method (right). The rest of the query is the same as in Figure 9.1.

of the base-learner in algorithms that have such components, yielding the results shown in Figure9.4. This provides a very detailed overview of learning performance on this dataset. For instance, when looking at SVMs, it is clear that especially the RBF-kernel is of great use here, which is not unexpected given that RBF kernels are popular in letter recognition problems. However, there is still much variation in the performance of the RBF-based SVM's, so it might be interesting to investigate this in more detail. Also, while most algorithms vary smoothly as their parameters are altered, there are large jumps in the performances of SVMs and RandomForests, which are, in all likelihood, caused by parameters that heavily affect their performance.

Moreover, when looking at the effects of bagging and boosting, it is clear that some base-learners are much stronger than others. For instance, it appears that bagging and boosting have almost no effect on logistic regression and naive Bayes. In fact, bagged logistic regression has been shown to perform

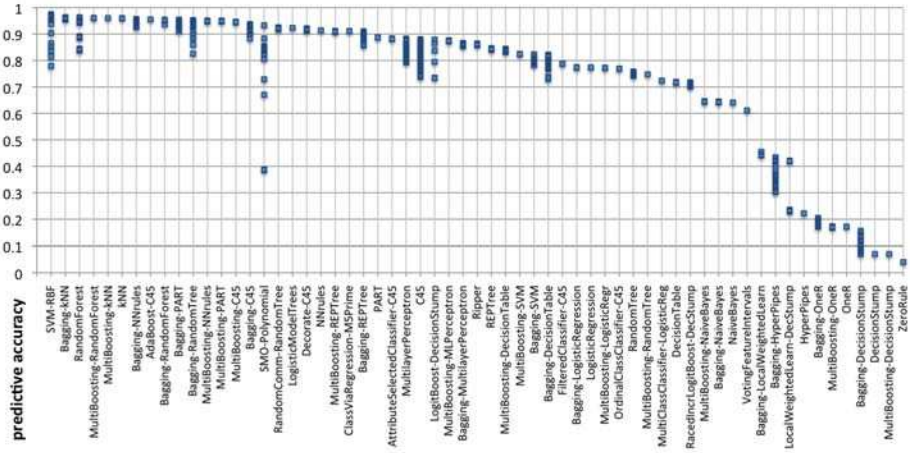


Figure 9.4: Performance of all algorithms on dataset 'letter', including base-learners and kernels. Some similar (and similarly performing) algorithms were omitted to allow a clear presentation

poorly (Perlich et al. 2003), and naive Bayes, which generally produces very little variance error (see Section 9.3) is unlikely to benefit from bagging, a variance-reduction method. Conversely, bagging random trees seems to be hugely profitable, but this does not hold for boosting. A possible explanation for this is the fact that random trees are very prone to variance error, which is primarily improved by bagging but much less by boosting (Bauer and Kohavi 1999). Another explanation is that boosting might have stopped early. Some learners, including the random tree learner, can yield a model with zero training error, causing boosting to stop after one iteration. A new query showed that on half of the datasets, the boosted random trees indeed yield exactly the same performance as the single random tree. This shows that boosting is best not used with base learners that can achieve zero training error, which includes random trees, random forests and SVMs with radial basis function or high-degree polynomial kernels.

Finally, it also seems more rewarding to fine-tune random forests, multi-layer perceptrons and SVMs than to bag or boost their default setting, while both bagging and boosting C4.5[1] does yield large improvements. Still, this is only one dataset, we will examine whether these observations still hold over many datasets in Section 9.1.3.

The remainder of this chapter is dedicated to investigating each of these aspects in more detail. Given the large number of stored experiments, each query returns very general results, likely to highlight things we were not expecting,

[1]This is actually J48, WEKA's implementation of the C4.5 algorithm.

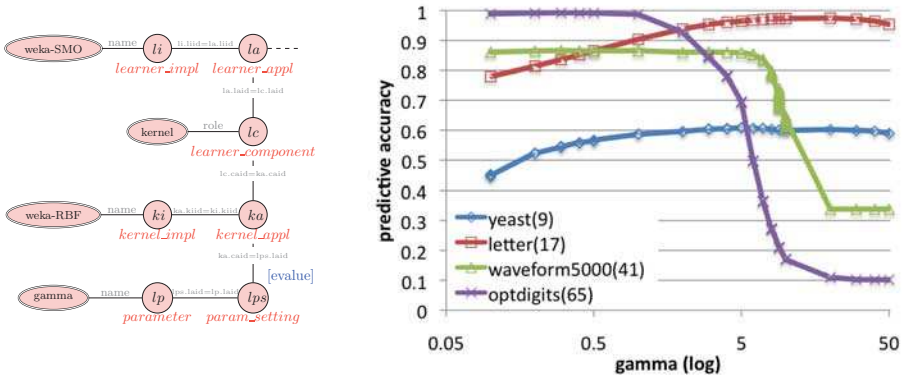


Figure 9.5: The effect of parameter gamma of the RBF-kernel in SVMs on a number of different datasets, with their number of attributes shown in brackets, and the accompanying query graph.

providing interesting cases for further study. In that respect, ExpDBs are a great tool for exploring learning behavior.

9.1.2 Investigating Parameter Effects

First, we examine the effect of the parameters of the RBF kernel. Based on the previous query (Figure 9.3 (left)), we can *zoom in* on the SVM's results by adding a constraint that the learner implementation should be weka.SMO, see Figure 9.5 (left), and add a new dimension by additionally asking for the value of the parameter we are interested in, e.g. 'gamma' of the 'RBF' kernel. While we are doing that, we can just as easily ask for the effect of this parameter on a number of other datasets as well, yielding Figure 9.5 (right).
When comparing the effect of gamma to the variation in RBF-kernel performance on the 'letter' dataset in the previous plot, we see that all the variation is explained by the effect of this parameter. We also see that its effect on other datasets is markedly different: on some datasets, performance increases until reaching a maximum and then slowly declines, while on other datasets, performance decreases very slowly up to a point, after which it quickly drops down to default accuracy. Moreover, this behavior seems to correlate with the number of attributes in each dataset (shown in brackets), which we will investigate further in Section 9.2.1.

9.1.3 General Comparisons

Previous queries investigated the performance of algorithms under rather specific conditions. Yet, by simply dropping the constraints on the datasets used,

the query will return the results over a large number of different problems. Furthermore, to compare algorithms over a range of performance metrics, instead of only considering predictive accuracy, we can use a normalization technique used by Caruana and Niculescu-Mizil (2006): normalize all performance metrics between the baseline performance and the best observed performance over all algorithms on each dataset. Using the aggregation functions of SQL, we can do this normalization on the fly, as part of the query. We won't print any more queries here, but they can be found in full on the database website.

One can now perform a very general comparison of supervised learning algorithms. We select all UCI datasets and all algorithms whose parameters were varied (see Section 7.2) and, though only as a point of comparison, logistic regression, nearest neighbors (kNN), naive Bayes and RandomTree with their default parameter settings. As for the performance metrics, we used predictive accuracy, F-measure, precision and recall, the last three of which were averaged over all classes. We then queried for the maximal (normalized) performance of each algorithm for each metric on each dataset, averaged each of these scores over all datasets, and finally ranked all classifiers by the average of predictive accuracy, precision and recall.[2] The result of this query is shown in Figure 9.6. Taking care not to overload the figure, we compacted groups of similar and similarly performing algorithms, indicated with an asterix (*). The overall best performing algorithms are mostly bagged and boosted ensembles. Especially bagged and boosted trees (including C4.5, PART, Ripper, NaiveBayesTree, REPTree and similar tree-based learners) perform very well, in agreement with the results reported by Caruana and Niculescu-Mizil (2006). Another shared conclusion is that boosting full trees performs dramatically better than boosting stumps (see Boosting-DStump) or boosting random trees.

However, one notable difference is that C4.5 performs slightly better than RandomForests and MultiLayerPerceptrons, though only for predictive accuracy, not for any of the other measures. A possible explanation lies in the fact that performance is averaged over both binary and multi-class datasets: since some algorithms perform much better on binary than on multi-class datasets, we can expect some differences. Still, it is easy to investigate these effects: we add a constraint that restricts our results to binary datasets and also ask for the evaluations of another metric used in Caruana and Niculescu-Mizil (2006) (root mean squared error (RMSE)), yielding Figure 9.7.

On the 35 binary datasets, MultiLayerPerceptrons and RandomForest do outperform C4.5 on all metrics, while bagged and boosted trees are still at the top of the list. On the other hand, boosted stumps perform much better on binary datasets.

[2]Since all algorithms were evaluated over all of the datasets (with 10-fold cross-validation), we could not optimize their parameters on a separate calibration set for this comparison. To limit the effect of overfitting, we only included a limited set of parameter settings, all of which were fairly close to the default setting. Nevertheless, these results should be interpreted with caution as they might be optimistic.

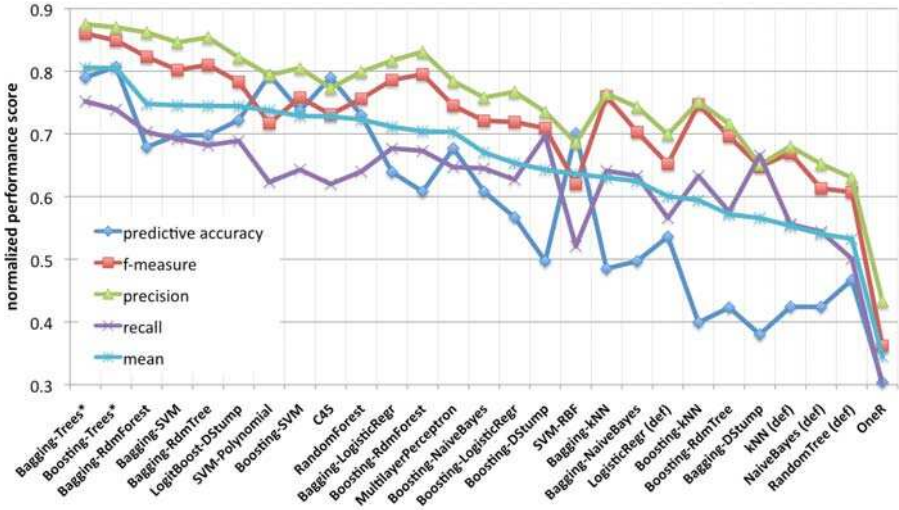


Figure 9.6: Ranking of algorithms over all datasets and over different performance metrics. Parameter settings are not fully optimized.

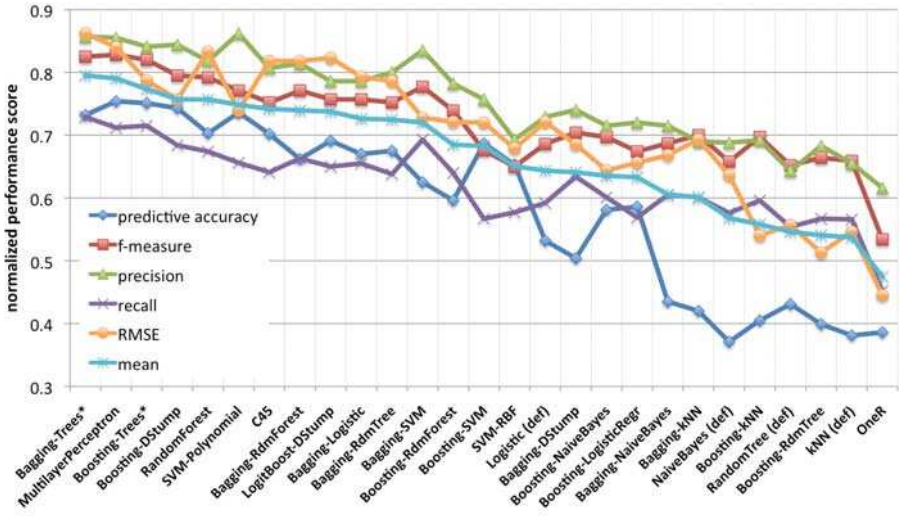


Figure 9.7: Ranking of algorithms over all binary datasets and over different performance metrics. Parameter settings are not fully optimized.

Furthermore, since this study contains many more algorithms, we can make a number of additional observations. In Figure 9.6, for instance, the bagged versions of most strong learners (SVM, C4.5, RandomForest, etc.) seem to improve primarily on precision and recall, while the original base-learners (with optimized parameters) perform better on predictive accuracy. Apparently, tuning the parameters of these strong learners has a much larger effect on accuracy than on the other metrics, for which it is better to employ bagging than parameter tuning, at least on multi-class datasets. On binary datasets, the relative performances of most algorithms seem relatively unaffected by the choice of metric, except perhaps for RMSE.

We can also check whether our observations from Section 9.1.1 still hold over multiple datasets and evaluation metrics. First of all, averaged over all datasets, the polynomial kernel seems to perform better than the RBF kernel in SVM's. Also contrary to what was observed earlier, bagging and boosting does generally improve the performance of logistic regression, at least when compared to its default setting. The other observations do hold: boosting (high-bias) naive Bayes classifiers is much better than bagging them, bagging (high-variance) random trees is dramatically better than boosting them, and while boosting trees is generally beneficial, boosting SVM's and RandomForests is not. This is further evidence that **boosting stops early on these algorithms**, while pruning mechanisms in tree learners avoid overfitting and thus allow boosting to perform many iterations.

Remarks Note that while this is a very comprehensive comparison of learning algorithms, we used rather general experiments: one parameter was varied while the other ones were kept at their default value (see Section 7.2). As these experiments were not performed to optimize the algorithm's parameter settings on each dataset, these rankings are not to be interpreted as 'the only true rankings'. Here, we aimed to show that such comparisons can be easily made by querying. Additional experiments should performed if one wishes to draw definite conclusions about relative learning performance.

Also, while we printed the general names of the learning algorithms to make them easily recognizable, remember that these are, in fact, all WEKA implementations. A simple query modification could ask for the implementation name instead, so that several implementations of a certain algorithm could be compared. However, to get a meaningful result, many more experiments should be run with algorithm implementations from many different toolboxes.

Ideally, the differences between these implementations should be expressed clearly, so that we can query for the effects of different implementation choices. The part of the ontology shown in Figure 5.14 is a good step in this direction, but further extensions will be needed to perform in-depth comparisons of different implementations of the same algorithms.

Finally, each such comparison is still only a snapshot in time. However, as new

algorithms, datasets and experiments are added to the database, one can at any time rerun the query and immediately see how things have evolved. These remarks are also valid in the next subsection.

9.1.4 Ranking Algorithms

In some applications, one prefers to have a ranking of learning approaches, preferably using a statistical significance test. This can also be written as a query in most databases. For instance, to investigate whether some algorithms consistently rank high over various problems, we can query for their average rank (using each algorithm's optimal observed performance) over a large number of datasets. Figure 9.8 shows the result of a single query over all UCI datasets, in which we selected 18 algorithms to limit the amount of statistical error generated by using 'only' 87 datasets. To check which algorithms perform significantly different, we used the Friedman test, as discussed in Demsar (2006). The right axis shows the average rank divided by the *critical difference*, meaning that two algorithms perform significantly different if the average ranks of two algorithms differ by at least one unit on that scale. The critical difference was calculated using the Nemenyi test with p=0.1, 18 algorithms and 87 datasets.

This immediately shows that indeed, some algorithms rank much higher on average than others over a large selection of UCI datasets. Boosting and bagging, if used with the correct base-learners, perform significantly better than SVMs, and SVMs in turn perform better than C4.5. We cannot yet say that SVM are also better than the MultilayerPerceptron or RandomForests: more datasets (or fewer algorithms in the comparison) are needed to reduce the critical difference. Note that the average rank of bagging and boosting is close to two, suggesting that a (theoretical) meta-algorithm that reliably chooses between the two approaches and the underlying base-learner would yield a very high rank. Indeed, rerunning the query while joining bagging and boosting yields an average rank of 1.7, down from 2.5.

Of course, to be fair, we should again differentiate between different base-learners and kernels. We can drill down through the previous results by adjusting the query, additionally asking for the base-learners and kernels involved, yielding Figure 9.9. Bagged Naive Bayes trees seem to come in first, but the difference is not significant compared to that of SVMs with a polynomial kernel (although it is compared to the RBF kernel). Also note that, just as in Section 9.1.1, bagging and boosting PART and NBTrees seem to yield big performance boosts, while boosting random trees proves particularly ineffective.

In any such comparison, it is important to keep the No Free Lunch theorem (Wolpert 2001) in mind: if all possible data distributions are equally likely, "... for any algorithm, any elevated performance over one class of problems is exactly paid for in performance over another class". Even if method A is better than method B across a variety of datasets, such as the UCI datasets

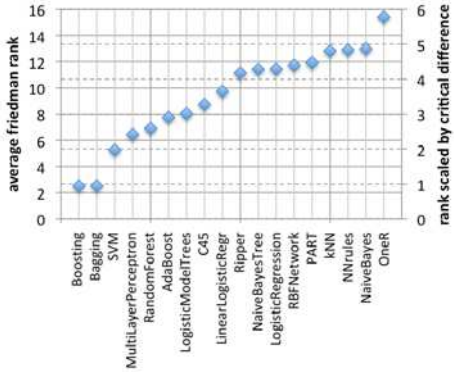


Figure 9.8: Average rank, general algorithms (non-optimized).

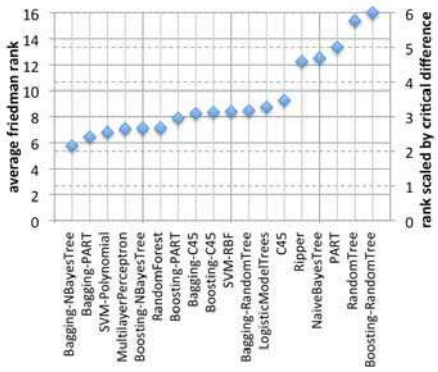


Figure 9.9: Average rank, specific algorithm setups (non-optimized).

in Figure 9.9, this could be attributed to certain properties of those datasets, and results may be very different over a group of somehow different datasets. An interesting avenue of research would be to repeat these queries on various collections of datasets, with different properties or belonging to specific tasks, to investigate such dependencies.

9.2 Data-Level analysis

While the queries in the previous section allow a detailed analysis of learning performance, they give no indication of exactly *when* (on which kind of datasets) a certain behavior is to be expected. In order to obtain results that generalize over different datasets, we need to look at the properties of individual datasets, and investigate how they affect learning performance.

9.2.1 Data Property Effects

In a first such study, we examine whether the 'performance jumps' that we noticed with the Random Forest algorithm in Figure 9.4 are linked to dataset size. Querying for the effect of the number of trees in the forest on all datasets, ordered from small to large yields Figure 9.10.
This shows that predictive accuracy increases with the number of trees (indicated with different labels), usually leveling off between 33 and 101 trees. One dataset in the middle of the figure is a notable exception: obtaining less than 50% accuracy with a single tree on a binary problem, **it actually performs worse as more trees are included**, as more of them will vote for the wrong class. Also, as dataset size increases, the accuracies for a given forest size vary

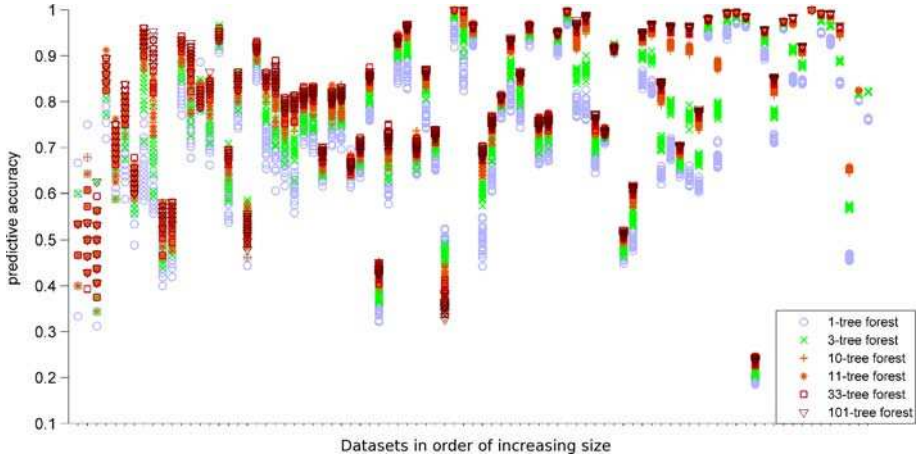


Figure 9.10: The effect of data size and the number of trees on random forests. The actual dataset names are omitted since they are too many to print legibly.

less as trees become more stable on large datasets, eventually causing clear performance jumps on very large datasets. However, for very small datasets, the benefit of using more trees is overpowered by the randomness in the trees. All this illustrates that even simple queries can give a detailed picture of an algorithm's behavior, showing interactions between parameters and data properties.

Let's take a closer look at the atypical dataset that caused RandomForest to perform worse with more trees. It is called `monks-problems-2_test`, and represents a binary classification problem with 6 attributes (each having 2 up to 4 nominal values). The task is to separate those instances for which exactly two of the attributes are assigned their first values. It is somewhat similar to parity problems, in the sense that higher-order interactions between the attributes (by which we mean that the value of an attribute depends on the values of several other attributes) make it hard to learn for most learning algorithms.

A query similar to the very first one yields Figure 9.11. We see that indeed, most algorithms do not surpass the default accuracy of 67% for this dataset (at least not with default parameters). There are however some notable exceptions. Some algorithms score very badly. Random Tree, for instance, does indeed hardly pass the 50% mark. Random Committee also uses Random Tree as a base-learner, and thus suffers from the same problem as Random Forests. Some algorithms reach perfect accuracy under certain parameter settings: MultilayerPerceptron when having 3 or more hidden nodes (all experiments used only one hidden layer) and SVMs when using a polynomial kernel of order 2 or higher (default accuracy is never surpassed when using first order polynomial or RBF kernels). These requirements are not surprising for dealing with a near-

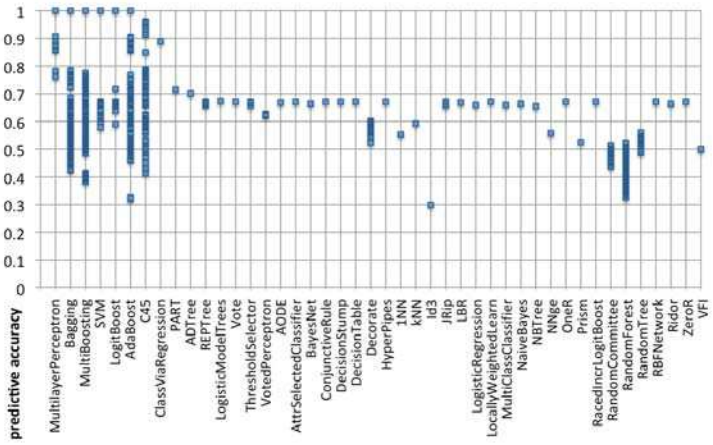
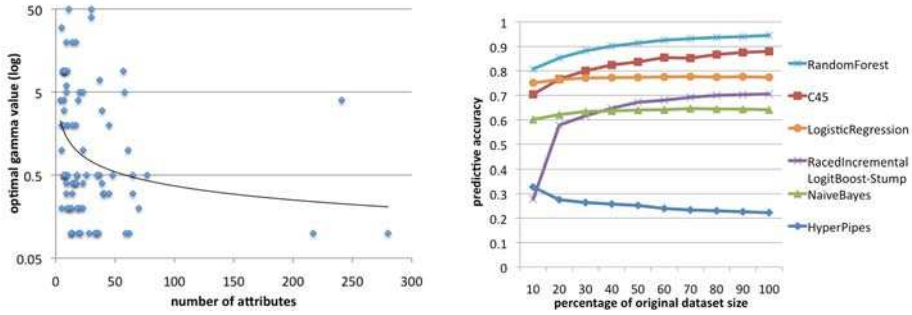


Figure 9.11: Performance comparison of all algorithms on the monks-problems-2_test dataset.

parity problem. The remaining perfect-scoring algorithms are ensembles, and only perform perfectly when using the MultilayerPerceptron as a base learner. The performance of C4.5 is harder to explain. We will revisit this question in Section 9.2.3.2.

A second effect we can investigate is whether the optimal value for the gamma-parameter of the RBF-kernel is indeed linked to the number of attributes in the dataset. After querying for the relationship between the gamma-value corresponding with the optimal performance and the number of attributes in the dataset used, we get Figure 9.12(a).

Although the number of attributes and the optimal gamma-value are not directly correlated, it looks as though high optimal gamma values predominantly occur on datasets with a small number of attributes, also indicated by the fitted curve. A possible explanation for this lies in the fact that this SVM implementation normalizes all attributes into the interval [0,1]. Therefore, the maximal squared distance between two examples, $\sum (a_i - b_i)^2$ for every attribute i, is equal to the number of attributes. Since the RBF-Kernel computes $e^{(-\gamma * \sum (a_i - b_i)^2)}$, the kernel value will go to zero very quickly for large gamma-values and a large number of attributes, making the non-zero neighborhood around a support vector very small. Consequently, the SVM will overfit these support vectors, resulting in low accuracies. This suggests that the RBF kernel should take the number of attributes into account to make the default gamma value more suitable across a range of datasets. It also illustrates how **experiment databases can assist algorithm development**.



(a) (b)

Figure 9.12: **(a)** The effect of the number of attributes on the optimal gamma-value. **(b)** Learning curves on the Letter-dataset.

9.2.2 Preprocessing Effects

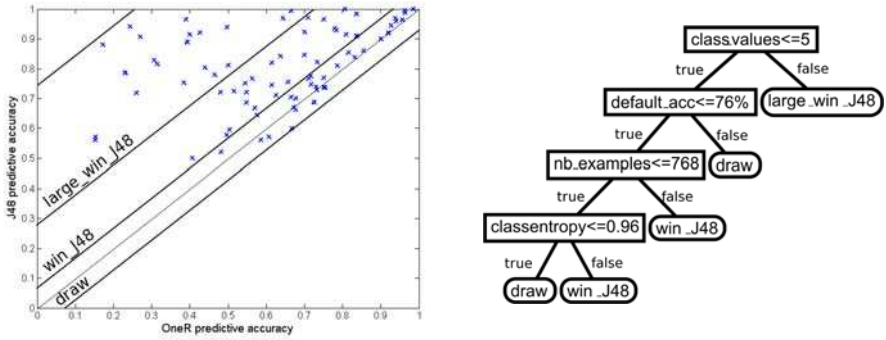
Since the database can also store preprocessing methods, we can investigate their effect on the performance of learning algorithms. For instance, to investigate if the results in Figure 9.2 are also valid on smaller samples of the 'letter' dataset, we can query for the results on downsampled versions of the dataset, yielding a learning curve for each algorithm, as shown in Figure 9.12(b). It is clear that the ranking of algorithms also depends on the size of the sample. While logistic regression is initially stronger than C4.5, the latter prevails when given more data: **the learning curves cross**! This confirms earlier analysis by Perlich et al. (2003). Also note that RandomForest performs consistently better, that RacedIncrementalLogitBoost crosses two other curves, and that the performance of the HyperPipes algorithm actually worsens given more data, which suggest it was "lucky" on the smaller samples.

9.2.3 Mining for Patterns in Learning Behavior

As shown in Figure 4.4, another way to tap into the stored information is to use data mining techniques to automatically model the effects of many different data properties on an algorithm's performance.

9.2.3.1 Modeling Data Property Effects

For instance, when looking at Figure 9.6, we see that OneR performs much worse than the other algorithms. Still, some earlier studies, most notably one by Holte (1993), found very little performance differences between OneR and the more complex C4.5. To study this discrepancy in more detail, we can query for the default performance of OneR and J48 (a C4.5 implementation) on all UCI datasets, and plot them against each other, as shown in Figure 9.13(a). This



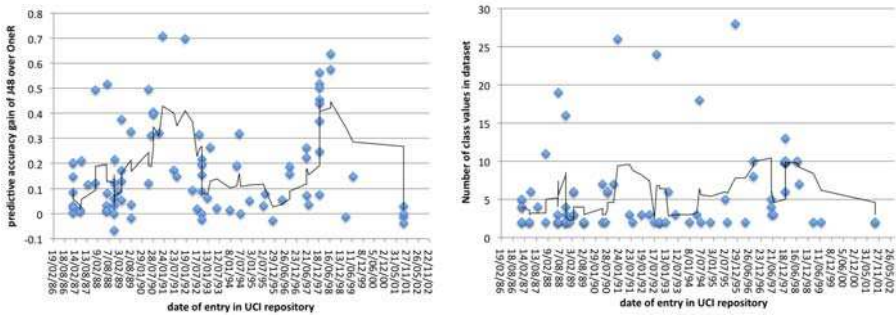
(a) (b)

Figure 9.13: **(a)** J48's performance against OneR's for all datasets, discretized into 3 classes. **(b)** A meta-decision tree predicting algorithm superiority based on data characteristics.

shows that on many datasets, the performances are indeed similar (crossing near the diagonal), while on others, C4.5 is the clear winner. We also see that J48's performance never drops below 50%, which makes it much more useful as a base-learner in ensemble-methods than OneR, which also can be deduced from Figure9.6.

To model the circumstances under which J48 performs better than OneR, we first discretize these results into three classes as shown in Figure 9.13(a): "draw", "win_J48" (4% to 20% gain), and "large_win_J48" (20% to 70% gain). We then extend the query by asking for all stored characteristics of the datasets used, and train a meta-decision tree on the returned data, predicting whether the algorithms will draw, or how large J48's advantage will be (see Figure 9.13(b)). From this **surprisingly small** tree, we learn that C4.5 has a clear advantage on datasets with many class values and, to a lesser extent, on large datasets with high class entropies. This can be explained by the fact that OneR bases its prediction only on the most predictive attribute. When the class attribute has more values than the attribute selected by OneR, performance will necessarily suffer.

Interestingly, if we add the constraint that only datasets published at the time of these earlier studies can be used, the dominance of C4.5 is much less pronounced. Querying for the date at which each dataset was published (in the UCI repository) gives us some insight into the effects of dataset availability. Figure 9.14(a) displays the average gain of C4.5 over OneR over time. This confirms that on the datasets used in (Holte 1993), all from the period 1988-1989, C4.5 only yields small improvements, while it greatly outperforms OneR on later datasets. Furthermore, Figure 9.14(b) plots the number of classes in each dataset over time, showing that most datasets of that period featured only



(a) (b)

Figure 9.14: **(a)** Gain of C4.5 over OneR over time and moving average. **(b)** Number of classes in UCI datasets over time and moving average.

a few classes. These figures also suggests that one of the conclusions in Holte (1993), i.e. that most UCI datasets were too simple, had a clear effect, since datasets added later seem to be more complex on average.

9.2.3.2 Modeling Parameter Effects

In Section 9.2.1 we investigated the effect of algorithm parameters on the `monks-problems-2_test` dataset. As can be seen from Figure 9.11, a lot of results for C4.5 score below the default accuracy, some lie above this default accuracy and a few reach accuracies higher than 85%. In order to gain insight into the relationship between the parameter settings of C4.5 and its accuracy on this dataset, we queried for all parameter settings of C4.5 on this dataset. Then, we used C4.5 to generate a (meta-)decision tree that predicts in which interval (below default accuracy, above default accuracy or higher than 85%) the accuracy lies of a tree built with a certain combination of parameters. The resulting tree, which obtained 97.3% accuracy, is shown in Figure 9.15.
The parameter settings used in the meta-tree are `binary_splits` (whether binary splits are used in the tree), `min_inst` (the minimal number of instances in a leaf of the tree), `use_rep` (whether reduced error pruning is used), `pruning` (whether pruning is used) and `conf_thresh` (the confidence threshold for regular pruning). As can be derived from the tree in Figure 9.15, C4.5 is able to build the best models if binary splits are used, if the minimal number of instances per leaf is smaller than 3 and if reduced error pruning is not used as the pruning technique. Knowing the concept of `monks-problems-2_test`, it is indeed clear that binary splits are a better option, as only the first value of each of the attributes is related to the class value. Moreover, more examples (and thus more splits) remain in each branch, allowing better generalization. Concerning the minimal number of instances in the leaves it is indeed necessary for a (near-)parity problem like this, to grow the tree (nearly) up to the level

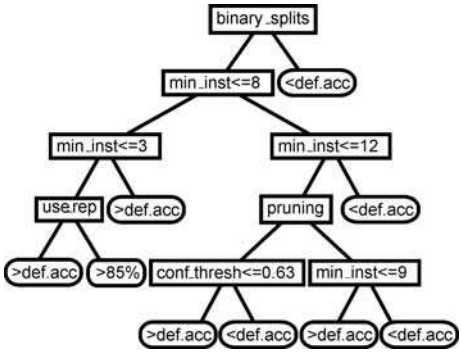


Figure 9.15: A meta-tree learned on a meta-dataset concerning predictive accuracies of trees learned on the `monks-problems-2_test` dataset.

of individual instances. The fact that reduced error pruning seems to reduce the performance over normal pruning could be explained by the validation set that is used to do the pruning, leaving fewer examples for training.

9.3 Method-Level Analysis

While the results in the previous section are clearly more generalizable towards the datasets used, they only consider individual algorithms and do not generalize over different techniques. Hence, we need to include the stored algorithm properties in our queries as well.

9.3.1 Bias-Variance Profiles

One very interesting algorithm property is its bias-variance profile. Since the database contains a large number of bias-variance decomposition experiments, we can give a **realistic numerical assessment** of how capable each algorithm is in reducing bias and variance error. Figure 9.16 shows, for each algorithm, the proportion of the total error that can be attributed to bias error, using default parameter settings and averaged over all datasets. The algorithms are ordered from large bias (low variance), to low bias (high variance). NaiveBayes is, as expected, one of the algorithms whose error consists primarily of bias error, while RandomTree has very good bias management, but generates more variance error. When looking at the ensemble methods, it clearly shows that bagging is a variance-reduction method, as it causes REPTree to shift significantly to the left, just as RandomForest sweeps RandomTree to the left. Conversely, boosting reduces bias, shifting DecisionStump to the right in AdaBoost and LogitBoost (additive logistic regression).

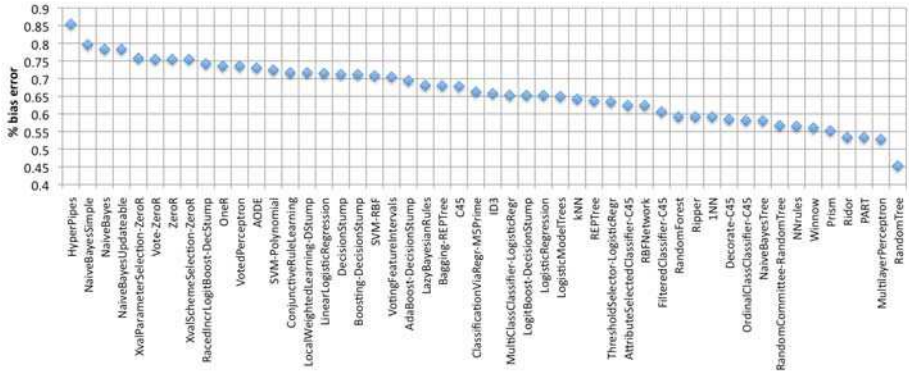


Figure 9.16: The average percentage of bias-related error for each algorithm averaged over all datasets.

9.3.2 Bias-Variance Effects

As a final study, we investigate the claim by Brain and Webb (2002) that on large datasets, the bias-component of the error becomes the most important factor, and that we should use algorithms with good bias management to tackle them. To verify this, we look for a connection between the dataset size and the proportion of bias error in the total error of a number of algorithms, using the previous figure to select algorithms with very different bias-variance profiles. Averaging the bias-variance results over datasets of similar size for each algorithm produces the result shown in Figure 9.17. It shows that bias error is of varying significance on small datasets, but **steadily increases in importance on larger datasets**, for all algorithms. This validates the previous study on a larger set of datasets. In this case (on UCI datasets), bias becomes the most important factor on datasets larger than 50000 examples, no matter which algorithm is used. As such, it is indeed advisable to look to algorithms with good bias management when dealing with large datasets.

9.4 Conclusions

Much can be learned by looking at past learning experiments, and the creation of experiment databases provides an effective way of tapping into this information, often yielding surprising new insights or generating interesting research questions. We summarize these findings in three types of studies, increasingly making use of the available meta-level descriptions, and offering increasingly generalizable results.

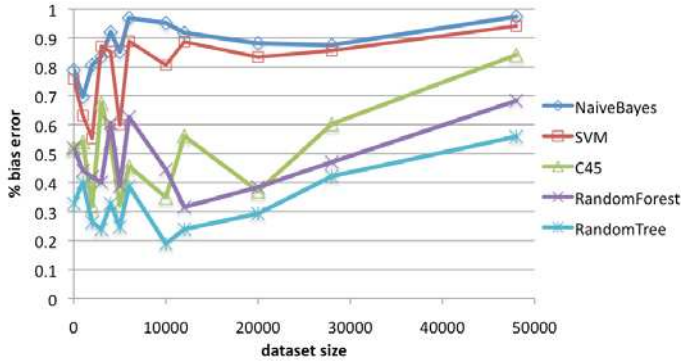


Figure 9.17: The average percentage of bias-related error in algorithms as a function of dataset size.

Model-level analysis. We first perform a very general ranking of learning algorithms and verify some earlier studies. We also make interesting observations concerning ensemble methods: the benefit of bagging over parameter optimization varies with the evaluation metric used, bagging primarily reduces variance error while boosting primarily reduces bias error, and boosting even seems to be useless for certain algorithms. Finally, we show that statistical significance tests can be included in queries to build algorithm rankings on the fly.
Data-level analysis. First, we show how the number of trees in a random forest interacts with the size of the dataset, and found that there exist situations where performance actually decreases as more trees are added. We also find that some SVM implementations tend to overfit data with large numbers of attributes and suggest algorithm improvements, thus illustrating how experiment databases can help algorithm design. By including data preprocessing steps, we also draw learning curves for several algorithms and show that they can cross. Finally, we build a surprisingly simple meta-decision tree modeling the circumstances under which one algorithm outperforms another and explain the result in terms of their internal learning mechanisms.
Method-level analysis. Using large amounts of bias-variance decomposition results, we create bias-variance profiles for all involved algorithms, and provide further evidence that low-bias algorithms are especially useful on large datasets. Still, we have probably just seen the tip of the iceberg, and many more types of queries could be written to delve deeper into the available experimental results. As such, we are confident that many more interesting results can be discovered by learning from past experiments. In the words of Albert Einstein:

> *Learn from yesterday, live for today, hope for tomorrow.*
> *The important thing is not to stop questioning.*

Conclusions Part II

Whereas Part I of this thesis hurdled the challenge of designing experiment databases that can collect and organize large amounts of empirical machine learning results from many researchers, Part II has demonstrated how such databases can be queried to gain new insights into learning algorithm behavior.

Interfaces First, in Chapter 8, two separate ways were identified to access the experiment database.
The first is a software interface that can be used by data mining tools to programmatically export experiments to ExpML files, or to immediately store them in an experiment database. We have used this interface to write a wrapper for the WEKA toolbox that accepts ExpML descriptions of experimental setups (without the outputs), runs the experiment and returns the finished experiment again in ExpML.
The second component is an intuitive query interface that allows users to easily compose database queries and thoroughly analyze the returned results. Two such interfaces are available: an online web interface and a desktop explorer tool. Both contain visualization tools to analyze the retrieved data, and both offer an intuitive graphical point-and-click interface that allows to compose queries simply by selecting the appropriate attributes and assembling the right constraints.

Exploring learning behavior Next, in Chapter 9, we have used these query tools and the experiment database designed and populated with experiments in Chapter 7 to evaluate how easily the results of previously stored experiments can be exploited for the discovery of new insights into a wide range of meta-learning questions. We summarize our findings in three types of studies, increasingly making use of the available meta-level descriptions, and offering increasingly generalizable results:
Model-level analysis. We performed a very general ranking of learning algorithms including all the algorithms and datasets stored, and multiple evaluation measures. We verified an earlier study by Caruana and Niculescu-Mizil (2006) and also made some additional observations. First, bagged versions of most strong learners (SVM, C45, RandomForest, etc.) seem to improve primarily on precision and recall, while the original base-learners (with optimized parameters) perform better on predictive accuracy. We also find that boosting (high-bias) naive Bayes classifiers is much better than bagging them, bagging

(high-variance) random trees is dramatically better than boosting them, and while boosting trees is generally beneficial, boosting SVM's and RandomForests is not. This is further evidence that boosting stops early on these algorithms. Next, we showed that advanced statistical significance tests Demsar (2006) can be employed in queries to build rankings of learning algorithms, and showed that over all UCI datasets, some algorithms are indeed significantly stronger than other ones.

Data-level analysis. In a first study of this type, we showed how the number of trees in a random forest interacts with the size of the dataset, and found one dataset, a near-parity problem, on which performance actually decreased as more trees were added to the ensemble. Next, we built a meta-learning tree based on C4.5 evaluations on that dataset, showing exactly how these parameters interact with the near-parity properties of the dataset.

We also found that WEKA's SVM implementation tends to overfit its support vectors for large gamma values on datasets with large numbers of attributes, which suggests that the RBF kernel should take the number of attributes into account. This illustrates how experiment databases can assist algorithm development.

When including data preprocessing steps, we found that the learning curves of several algorithms cross.

Finally, we built another meta-decision tree modeling the circumstances under which J48 outperforms the OneR algorithm (previously shown to perform similarly on many datasets (Holte 1993)), and explained the result in terms of their internal workings. Restricting the results to datasets available at previous time periods explained why previous studies obtained different results.

Method-level analysis. Using large amounts of bias-variance decomposition results, we created a bias-variance profile for a large number of algorithms, showing the average proportion of its total error that can be attributed to bias error. It also shows that bagging is a variance-reduction method, while boosting reduces bias.

A final study of the effect of the dataset size on the proportion of bias error in the total error of a number of algorithms, showed that it is indeed advisable to look to algorithms with good bias management when dealing with large datasets (Brain and Webb 2002).

All this illustrates that much can be learned by looking at past learning experiments, and that building experiment databases and querying them provides an effective way of tapping into this information, often yielding surprising new insights or generating interesting research questions.

Finale

A teacher is a person who never says anything once.

Howard Nemerov

Chapter Ten

Summary and Future Work

10.1 Summary

Research in machine learning and data mining can be speeded up tremendously by moving empirical research results "out of people's heads and labs, onto the network and into tools that help us structure and filter the information" (Nielsen 2008). The massive streams of experiments that are being executed to benchmark new algorithms, test hypotheses or model new datasets have many more uses beyond their original intent, but are often discarded or their details are lost over time. In this thesis, we developed a framework to automatically export experiments to *experiment databases*, databases specifically designed to collect all the details on large numbers of past experiments, performed by many different researchers, and to compose queries about almost any aspect of the behavior of learning algorithms. They can be set up for personal use, to share results within a lab, or to build community-wide repositories.
This research is thus specifically aimed at facilitating and speeding up machine learning research itself:

- Many questions about data mining algorithms (e.g. the effect of a parameter on a particular performance criterion) can be answered *on the fly* - in a single query to the database - using the combined results of many prior studies. In the current methodology, this requires the manual setup of new experiments, a very time consuming and laborious procedure.

- Large-scale empirical studies providing generally valid insights are sped up tremendously by experiment reuse. While most algorithms are relatively fast, testing them under many circumstances (e.g. many different datasets and many different parameter settings) is prohibitively expensive when one has to start from scratch. The current status quo is that most studies limit themselves to a relatively small selection of datasets and parameter variations, making it hard to interpret how generalizable the findings are. It has been argued that this creates a false sense of progress (Hand 2006).
- Experiment repositories also serve as a venue to store all the details that make the experiments reproducible, which is crucial to verify prior results and to build off them. Unfortunately, data mining experiments are currently not being documented with the rigor found in noncomputing experimental sciences, leading to a lot of critique (Pedersen 2008). In fact, reproducibility is listed as one of the foremost concerns in a recent panel of the SIAM International Conference on Data Mining (Hirsh 2008).

First, we have covered the practice and state-of-the-art in meta-learning research and its applications, providing a foundation for the work in the subsequent parts. In particular, we defined a framework for meta-learning investigations, and we also used this to highlight the different types of meta-data that should be stored in order to gain insight into the behavior of learning algorithms. Next, we provide a survey of the various architectures that have been developed, or simply proposed, to build KD support systems. We also propose a new KD support architecture that combines the best aspects of earlier systems, and that solves many of their limitations by taking a community-based approach built around experiment databases that allow many people using the same KD techniques to collaborate and benefit from each other's meta-data.

Next, we have set out to organize the world's machine learning information, or at least a portion of it. In a first step to this ideal, we offer tools that allow the massive streams of experiments that are being executed to evaluate learning algorithms to be automatically shared and gathered into open experiment databases that intelligently organize all the meta-data so it can be investigated in depth. To learn from previous practical applications of experiment repositories, we looked to so-called e-Sciences, where they have, in some form or another, been applied successfully for some time now, particularly in bioinformatics, astrophysics and high-energy physics. From this, we learned that they all seem to have evolved towards online, public infrastructures for experiment exchange, using more or less the same three components.

The Exposé ontology. We have established **Exposé**: an ontology for experimentation in machine learning. It covers a large portion of the involved concepts and their relationships, thus providing a core vocabulary on which we can build common, extensible representation languages and database models. While focussing on supervised classification and propositional datasets,

the resulting ontology is quite extensive. It treats experiments as workflows to allow the definition of various types of experiments and includes the possibly extensive data processing workflows that precede them. It also covers the experiment context in which individual learning algorithm evaluations are performed, the various learner evaluation measures and performance estimation techniques that are used in current research, the structure and properties of datasets and of learning algorithms. The latter include their parameters, internal components such as kernels, base-learners and distance functions, and their internal learning mechanisms.

The ExpML Markup Language. Next, we cast this ontological vocabulary into a formal, XML-based language, dubbed **ExpML**. By consistently translating the involved concepts and their relationships to XML elements and syntax, we obtain a flexible language that describes experiments as workflows and defines the exact composition of all involved operators. Moreover, any ontological extensions towards other machine learning tasks and other types of experiments can be translated in the same fashion, thus allowing ExpML to adapt quickly to other subfields or changing needs. We have illustrated this translation process and the flexibility of the ensuing language by using a real experiment as it was run, described and stored in our experiment database. Moreover, a set of guidelines was presented about the minimal amount of information that should be included to make sure the experiments are reproducible. As such, any submitted ExpML files can be automatically verified for completeness.

Experiment Databases Then, we have investigated how machine learning experiments can be intelligently organized in searchable experiment databases. Starting from the Exposé ontology, we derived a detailed database model for classification experiments that allows to write queries about the many aspect of learning behavior that are covered by the ontology. In this database model, each singular experiment becomes a point in the space of all possible experiments, immediately linked to all known theoretical information about its components. By querying it, we can quickly locate and rearrange thousands of experiments, and reuse them in many future studies. Also, since we want to use this database to gain insight into the behavior of machine learning algorithms under various conditions, we populated it with a myriad of very diverse experiments on many algorithms and many datasets.

Finally, we demonstrate how such databases can be queried to gain new insights into learning algorithm behavior.

Query Interfaces We provide two interfaces that hide the inherent complexities of experiment databases. The first is a software interface that can be used by data mining tools to programmatically export experiments to ExpML files, or to immediately store them in an experiment database. We have used this interface to write a wrapper for the WEKA toolbox that accepts ExpML descriptions of experimental setups (without the outputs), runs the experiment and returns the finished experiment again in ExpML. The second component is an intuitive query interface that allows users to easily compose database queries

and thoroughly analyze the returned results. Two such interfaces are available: an online web interface and a desktop explorer tool. Both contain visualization tools to analyze the retrieved data, and both offer an intuitive graphical point-and-click interface that allows to compose queries simply by selecting the appropriate attributes and assembling the right constraints.

Exploring learning behavior Finally, we used these query tools and the experiment database to evaluate how easily the results of previously stored experiments can be exploited for the discovery of new insights into a wide range of meta-learning questions. We summarize these findings in three types of studies, increasingly making use of the available meta-level descriptions, and offering increasingly generalizable results.

Model-level analysis. We first perform a very general ranking of learning algorithms and verify some earlier studies. We also make interesting observations concerning ensemble methods: the benefit of bagging over parameter optimization varies with the evaluation metric used, bagging primarily reduces variance error while boosting primarily reduces bias error, and boosting even seems to be useless for certain algorithms. Finally, we show that statistical significance tests can be included in queries to build algorithm rankings on the fly.

Data-level analysis. First, we show how the number of trees in a random forest interacts with the size of the dataset, and found that there exist situations where performance actually decreases as more trees are added. We also find that some SVM implementations tend to overfit data with large numbers of attributes and suggest algorithm improvements, thus illustrating how experiment databases can help algorithm design. By including data preprocessing steps, we also draw learning curves for several algorithms and show that they can cross. Finally, we build a surprisingly simple meta-decision tree modeling the circumstances under which one algorithm outperforms another and explain the result in terms of their internal learning mechanisms.

Method-level analysis. Using large amounts of bias-variance decomposition results, we create bias-variance profiles for all involved algorithms, and provide further evidence that low-bias algorithms are especially useful on large datasets.

10.2 Future Work

10.2.1 A Portal for Collaborative Experimentation

While the feasibility and practical usefulness of experiment databases has been demonstrated in this thesis, we believe that the potential of this approach can be leveraged tremendously, and made available to the entire machine learning community, by extending it along a number of dimensions, each of which entails new scientific challenges.

Ontological extensions Our ontology can still be extended significantly, most importantly to other data mining tasks, such as regression, clustering, graph

mining, relational learning, data streams and time-series analysis. Moreover, the current model is mainly aimed at covering the syntax of describing experiments, and the ontology should be extended to model semantic aspects of data mining as well. For instance, many data mining algorithms are variants of the same basic approach, or share certain properties with other algorithms. These relationships should be investigated and expressed explicitly to allow a much deeper analysis of performance results. Many of these semantic relationships could be discovered by querying the portal itself: newly observed trends can lead to new ontological descriptions, which in turn allow more powerful queries, thus creating a positive feedback loop. An example of this can be found in the bias-variance decompositions in Chapter 9: when averaged over many datasets, these can in turn be used to characterize known algorithms.

Querying models Models are currently not stored in a way that allows them to be included in queries. To solve this, we aim to employ techniques from *inductive databases* (Fromont et al. 2007), databases in which models are stored next to the data from which they were extracted.

Auto-population Another benefit of inductive databases is that they can execute algorithms based on a given query (e.g., when no results yet exist under the stated constraints). Currently, an experiment database is passive and will only return previously stored results. However, by interpreting a users query, it could actively start the execution of new experiments, and show the new results to the user as they become available. We want to investigate how to interpret these queries and use *active learning* principles to select the most useful experiments given the ones that are already stored. Results from this research will also be very useful to further research on inductive databases.

Quality Control When everybody can freely submit buggy algorithms, or bad (perhaps even fraudulent) ExpML descriptions, the quality of an experiment database will diminish. Therefore, we should think about ways to verify the quality of submitted results. One solution, used in several repositories in bioinformatics, is to attach a trustworthiness value to the source of certain results. Experiments submitted from a trusted tool may be labeled very trustworthy, while custom submissions might get a lower value until the results are verified. In the case that a portal can run its own experiments, it could rerun all submitted experiments and verify the results.

10.2.2 Large-scale meta-learning

The second avenue of future research is to use this portal as a platform for meta-learning studies. Data mining methods will then be run on this portal to discover insightful patterns in the performance of data mining algorithms on various problems. The sheer amount of experiments stored in such portals and the way in which they are organized enables meta-learning studies that were not possible before.

Meta-models A first approach is to automatically build interpretable models

that relate the performance of data mining algorithms to properties of the data on which they were run, as shown in Chapter 9. We plan to expand this research and look for many more insightful patterns in data mining performance. Using the portal and its semantic descriptions, such discoveries are likely to be made much faster than was possible before.

Algorithm recommendation Past experiments are also very useful to provide practical advice for new data mining problems. Many proposals exist for recommender systems, but to the best of our knowledge, none were built to process this amount of experiments with this amount of detail (e.g. parameter settings, data preprocessing workflows). This brings new opportunities to provide more targeted advice.

Appendix

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 $[\rho_1 \dots \rho_{10}]$ 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.

Alpaydin, E. (1999). Combined 5 2 cv f test for comparing supervised classification learning algorithms. *Neural computation 11*(8), 1885–1892.

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

Ashburner, M., C. A. Ball, J. A. Blake, D. Botstein, H. Butler, J. M. Cherry, A. P. Davis, K. Dolinski, S. S. Dwight, J. T. Eppig, M. A. Harris, D. P. Hill, L. Issel-Tarver, A. Kasarskis, S. Lewis, J. C. Matese, J. E. Richardson, M. Ringwald, G. M. Rubin, and G. Sherlock (2000). Gene ontology: tool for the unification of biology. *Nature Genetics 25*, 25–29.

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

Baader, F., I. Horrocks, and U. Sattler (2005). Description logics as ontology languages for the semantic web. *Lecture Notes in Artificial Intelligence 2605*, 228–248.

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

Ball, C., A. Brazma, H. Causton, and S. Chervitz (2004). Submission of microarray data to public repositories. *PLoS Biology 2*(9), e317.

Bauer, E. and R. Kohavi (1999). An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning 36*(1-2), 105–139.

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.

Bernstein, A. and M. Daenzer (2007). The next system: Towards true dynamic adaptations of semantic web service compositions. *Lecture Notes in Computer Science 4519*, 739–748.

Bernstein, A., F. Provost, and S. Hill (2005). Toward intelligent assistance for a data mining process: an ontology-based approach for cost-sensitive classification. *IEEE Transactions on Knowledge and Data Engineering 17*(4), 503–518.

Beygelzimer, A. and S. Dasgupta (2009). Importance weighted active learning. *Proceedings of the 26th International Conference on Machine Learning*, 49–56.

Bishop, C. (2006). *Pattern Recognition and Machine Learning*. Springer.

Blockeel, H. (2006). Experiment databases: A novel methodology for experimental research. *Lecture Notes in Computer Science 3933*, 72–85.

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*.

Bonnarel, F., P. Fernique, D. Egret, F. Genova, F. Ochsenbein, M. Wenger, and J. G. Bartlett (2000). The aladin interactive sky atlas - a reference tool for identification of astronomical sources. *Astronomy and Astrophysics Supplement Series 143*, 33–40.

Bouckaert, R. and E. Frank (2004). Evaluating the replicability of significance tests for comparing learning algorithms. *Lecture Notes in Computer Science 3056*, 3–12.

Bradford, J. and C. Brodley (2001). The effect of instance-space partition on significance. *Machine Learning 42*, 269–286.

Brain, D. and G. Webb (2002). The need for low bias algorithms in classification learning from large data sets. *PKDD '02: Proceedings of the 6th European Conference on Principles of Data Mining and Knowledge Discovery*, 62—73.

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.

Brazma, A., P. Hingamp, J. Quackenbush, G. Sherlock, P. Spellman, C. Stoeckert, J. Aach, W. Ansorge, C. A. Ball, H. C. Causton, T. Gaasterland, P. Glenisson, F. C. Holstege, I. F. Kim, V. Markowitz, J. C. Matese, H. Parkinson, A. Robinson, U. Sarkans, S. Schulze-Kremer, J. Stewart, R. Taylor, and J. V. M. Vingron (2001). Minimum information about a microarray experiment. *Nature Genetics 29*, 365 – 371.

Brazma, A., H. Parkinson, U. Sarkans, M. Shojatalab, J. Vilo, N. Abeygunawardena, E. Holloway, M. Kapushesky, P. Kemmeren, G. G. Lara, A. Oezcimen, P. Rocca-Serra, and S.-A. Sansone (2003). Arrayexpress–a public repository for microarray gene expression data at the ebi. *Nucleic Acids Research 31*(1), 68–71.

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

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

Brown, D., R. Vogt, B. Beck, and J. Pruet (2007). High energy nuclear database: a testbed for nuclear data information technology. *International Conference on Nuclear Data for Science and Technology*, Article 250.

Calero, C., F. Ruiz, and M. Piattini (2006). Ontologies for software engineering and software technology. *Springer*.

Cannataro, M. and C. Comito (2003). A data mining ontology for grid programming. *First International Workshop on Semantics in Peer-to-Peer and Grid Computing at WWW 2003*, 113–134.

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

Caruana, R. and A. Niculescu-Mizil (2006). An empirical comparison of supervised learning algorithms. *Proceedings of the 23rd International Conference on Machine Learning (ICML'06)*, 161–168.

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.

Chandrasekaran, B. and J. Josephson (1999). What are ontologies, and why do we need them? *IEEE Intelligent systems 14*(1), 20–26.

Chapman, P., J. Clinton, R. Kerber, T. Khabaza, T. Reinartz, C. Shearer, and R. Wirth (1999). Crisp-dm 1.0. a step-by-step data mining guide [http://www.crisp-dm.org]. *Online document*.

Charest, M. and S. Delisle (2006). Ontology-guided intelligent data mining assistance: Combining declarative and procedural knowledge. *Proceedings of the 10th IASTED International Conference on Artificial Intelligence and Soft Computing*, 9–14.

Charest, M., S. Delisle, O. Cervantes, and Y. Shen (2006). Intelligent data mining assistance via cbr and ontologies. *Proceedings of the 17th International Conference on Database and Expert Systems Applications (DEXA'06)*.

Charest, M., S. Delisle, O. Cervantes, and Y. Shen (2008). Bridging the gap between data mining and decision support: A case-based reasoning and ontology approach. *Intelligent Data Analysis 12*, 1–26.

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.

Cohn, D. (1996). Neural network exploration using optimal experiment design. *Advances in Neural Information Processing Systems 6*, 679–686.

Cohn, D., Z. Ghahramani, and M. Jordan (1996). Active learning with statistical models. *Journal of Artificial Intelligence Research 4*, 129–145.

Craw, S., D. Sleeman, N. Graner, and M. Rissakis (1992). Consultant: Providing advice for the machine learning toolbox. *Research and Development in Expert Systems IX: Proceedings of Expert Systems '92*, 5–23.

Davis, J. and M. Goadrich (2006). The relationship between precision-recall and roc curves. *Proceedings of the 23rd International Conference on Machine Learning (ICML'06)*, 233–240.

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

Dean, M., D. Connolly, F. van Harmelen, J. Hendler, I. Horrocks, D. orah L McGuinness, P. F. Patel-Schneider, and L. A. Stein (2003). Web ontology language (owl) reference version 1.0. *W3C Working Draft. Available at http://www.w3.org/TR/2003/WD-owl-ref-20030331*.

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

Demšar, J., B. Zupan, G. Leban, and T. Curk (2004). Orange: From experimental machine learning to interactive data mining. *White Paper (www.ailab.si/orange)*.

Derriere, S., A. Preite-Martinez, and A. Richard (2006). Ucds and ontologies. *ASP Conference Series 351*, 449.

Diamantini, C., D. Potena, and E. Storti (2009). Kddonto: An ontology for discovery and composition of kdd algorithms. *Proceedings of the Third Generation Data Mining Workshop at the 2009 European Conference on Machine Learning (ECML 2009)*.

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

Dietterich, T. (1998). Approximate statistical tests for comparing supervised classification learning algorithms. *Neural computation 10*(7), 1895–1923.

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.*

Drummond, C. (2009). Replicability is not reproducibility: Nor is it good science. *Proceedings of the Evaluation Methods for Machine Learning Workshop at the 26th ICML.*

Drummond, C. and R. Holte (2000). Explicitly representing expected cost: An alternative to roc representation. *Proceedings of the sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 198–207.

Dzeroski, S. (2007). Towards a general framework for data mining. *Lecture Notes in Computer Science 4747*, 259–300.

El-Ghalayini, H., M. Odeh, and R. McClatchey (2007). Engineering conceptual data models from domain ontologies: A critical evaluation. *International Journal of Information Technology and Web Engineering 2*(1).

Engels, R. (1996). Planning tasks for knowledge discovery in databases; performing task-oriented user-guidance. *Proceedings of the 2nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'96)*, 170–175.

Everitt, B. (1992). The analysis of contingency tables. second edition. *Chapman and Hall, Londen.*

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.*

Fawcett, T. (2009). Ml 2d pattern gallery, http://home.comcast.net/~tom.fawcett/public_html/ml-gallery/pages/index.html. *Online document.*

Fayyad, U., G. Piatetsky-Shapiro, and P. Smyth (1996). From data mining to knowledge discovery in databases. *AI magazine 17*(3), 37–54.

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

Fikes, R. and N. Nilsson (1971). Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence 2*, 189–208.

Foster, I. (2005). Service-oriented science. *Science 308*(5723), 814.

Foundalis, H. (2006). Phaeaco: A cognitive architecture inspired by bongard's problems. *PhD Thesis. Department of Computer Science. Indiana University*.

Frawley, W. (1989). The role of simulation in machine learning research. *ANSS '89: Proceedings of the 22nd annual symposium on Simulation*, 119–127.

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

Friedman, M. (1940). A comparison of alternative tests of significance for the problem of m rankings. *The Annals of Mathematical Statistics 11*(1), 86–92.

Fromont, E., H. Blockeel, and J. Struyf (2007). Integrating decision tree learning into inductive databases. *Lecture Notes in Computer Science 4747*, 81–96.

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.

Giraud-Carrier, C. (2008). Metalearning-a tutorial. *Tutorial at the 2008 International Conference on Machine Learning and Applications (ICMLA'08)*.

Giraud-Carrier, C. and F. Provost (2005). Toward a justification of meta-learning: Is the no free lunch theorem a show-stopper? *Proceedings of the ICML-2005 Workshop on Meta-learning*, 9–16.

Goble, C., O. Corcho, P. Alper, and D. D. Roure (2006). e-science and the semantic web: A symbiotic relationship. *Lecture Notes in Computer Science 4265*, 1–12.

Grabczewski, K. and N. Jankowski (2007). Versatile and efficient meta-learning architecture: Knowledge representation and and management in computational intelligence. *IEEE Symposium on Computational Intelligence and Data Mining*, 51–58.

Hall, M. (1998). Correlation-based feature selection for machine learning. *Ph.D dissertation Hamilton, NZ: Waikato University, Department of Computer Science*.

Hall, M., E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. Witten (2009). The weka data mining software: An update. *SIGKDD Explorations 11*(1), 10–18.

Hand, D. (2006). Classifier technology and the illusion of progress. *Statistical Science*.

Hand, D. (2009). Measuring classifier performance: a coherent alternative to the area under the roc curve. *Machine Learning 77*(1), 103–123.

Hartley, R. (1994). Ml-list. subject: Conservation law. *LCG Discussion Thread, Machine Learning List, Vol. 6, Nos. 22-27 (online at: ftp://ftp.ics.uci.edu/pub/ml-list/V6)*.

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

Hidalgo, M., E. Menasalvas, and S. Eibe (2009). Definition of a metadata schema for describing data preparation tasks. *Proceedings of the ECML/PKDD09 Workshop on 3rd generation Data Mining (SoKD-09)*, 64–75.

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.

Hilario, M., A. Kalousis, P. Nguyen, and A. Woznica (2009). A data mining ontology for algorithm selection and meta-mining. *Proceedings of the ECML/PKDD09 Workshop on 3rd generation Data Mining (SoKD-09)*, 76–87.

Hirsh, H. (2008). Data mining research: Current status and future opportunities. *Statistical Analysis and Data Mining 1*(2), 104–107.

Holte, R. (1993). Very simple classification rules perform well on most commonly used datasets. *Machine Learning 11*, 63–91.

Homann, J. and B. Nebel (2001). The ff planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research 14*, 253–302.

Horridge, M., H. Knublauch, A. Rector, R. Stevens, and C. Wroe (2009). A practical guide to building owl ontologies using protege 4 and co-ode tools. *The University of Manchester*.

Horrocks, I., P. Patel-Schneider, and H. Boley (2004). Swrl: A semantic web rule language combining owl and ruleml. *W3C Member submission, http://www.w3.org/Submissions/SWRL/*.

Hoste, V. and W. Daelemans (2005). Comparing learning approaches to coreference resolution. there is more to it than bias. *Proceedings of the Workshop on Meta-Learning (ICML-2005)*, 20–27.

Hume, D. (1740). A treatise of human nature: Texts. *(Edited by D.F. Norton and M.J. Norton, Oxford University Press, 2000)*, 1173.

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.

Jain, A., M. Murty, and P. Flynn (1999). Data clustering: a review. *ACM computing surveys (CSUR) 31*(3), 264–323.

Janssen, F. and J. Fürnkranz (2007). On meta-learning rule learning heuristics. *Proceedings of the 7th IEEE International 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., A. Bernstein, and M. Hilario (2008). Meta-learning with kernels and similarity functions for planning of data mining workflows. *ICML/-COLT/UAI 2008 Planning to Learn Workshop (PlanLearn)*, 23–28.

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.

Karapiperis, S. and D. Apostolou (2006). Consensus building in collaborative ontology engineering processes. *Journal of Universal Knowledge Management 1*(3), 199–216.

Kaufman, K. (1997). Inlen: a methodology and integrated system for knowledge discovery in databases. *PhD Thesis. School of Information Technology and Engineering, George Mason University*.

Kaufman, K. and R. Michalski (1998). Discovery planning: Multistrategy learning in data mining. *Proceedings of the Fourth International Workshop on Multistrategy Learning*, 14–20.

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 S. Kasetty (2003). On the need for time series data mining benchmarks: A survey and empirical demonstration. *Data Mining and Knowledge Discovery 7*(4), 349–371.

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.

Kietz, J., F. Serban, A. Bernstein, and S. Fischer (2009). Towards cooperative planning of data mining workflows. *Proceedings of the Third Generation Data Mining Workshop at the 2009 European Conference on Machine Learning (ECML 2009)*, 1–12.

King, R., J. Rowland, S. Oliver, M. Young, W. Aubrey, E. Byrne, M. Liakata, M. Markham, P. Pir, L. Soldatova, A. Sparkes, K. Whelan, and A. Clare (2009). The automation of science. *Science 324(3)*(5923), 85–89.

Klusch, M., A. Gerber, and M. Schmidt (2005). Semantic web service composition planning with owls-xplan. *Proceedings of the First International AAAI Fall Symposium on Agents and the Semantic Web*.

Kodratoff, Y., D. Sleeman, M. Uszynski, K. Causse, and S. Craw (1992). Building a machine learning toolbox. *Enhancing the knowledge engineering process: contributions from ESPRIT*, 81–108.

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.

Kohonen, T. (1998). The self-organizing map. *Neurocomputing 21*(1-3).

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.

Kuehl, R. (1999). Design of experiments: statistical principles of research design and analysis. *Duxbury Press*.

LaLoudouana, D. and M. Tarare (2003). Data set selection. *Journal of Machine Learning Gossip 1*, 11–19.

Le-Khac, N., M. Kechadi, and J. Carthy (2006). Admire framework: Distributed data mining on data grid platforms. *Proceedings of the 1st International Conference on Software and Data Technologies 2*, 67–72.

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.

Liu, Z., A. Ranganathan, and A. Riabov (2007). A planning approach for message-oriented semantic web service composition. *Proceedings of the National Conference on AI 5*(2), 1389–1394.

Manolescu, I., L. Afanasiev, A. Arion, J. Dittrich, S. Manegold, N. Polyzotis, K. Schnaitter, P. Senellart, and S. Zoupanos (2008). The repeatability experiment of sigmod 2008. *ACM SIGMOD Record 37*(1).

Markatou, M., H. Tian, S. Biswas, and G. Hripcsak (2006). Analysis of variance of cross-validation estimators of the generalization error. *Journal of Machine Learning Research 6*, 1127–1168.

METAL (2001). Metal: A meta-learning assistant for providing user support in machine learning and data mining. *ESPRIT Framework IV LRT Reactive Project Nr. 26.357*.

Michalski, R., L. Kerschberg, and K. Kaufman (1992). Mining for knowledge in databases: The inlen architecture, initial implementation and first results. *Journal of Intelligent Information Systems 1*(1), 85–113.

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

Mitchell, T. (1980). The need for biases in learning generalizations. *Readings in machine learning*.

Mitchell, T. (2006). The discipline of machine learning. *Machine Learning Department technical report CMU-ML-06-108*.

MLT (1993). Machine learning toolbox. *Esprit Framework II Research Project Nr. 2154*.

Morik, K. and M. Scholz (2004). The miningmart approach to knowledge discovery in databases. *Intelligent Technologies for Information Analysis*, 47–65.

Nadeau, C. and Y. Bengio (2003). Inference for the generalization error. *Machine Learning 52*(3), 239–281.

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*.

Nielsen, M. (2008). The future of science: Building a better collective memory. *APS Physics 17*(10).

Noy, N. and D. McGuinness (2002). Ontology development 101: A guide to creating your first ontology. *Stanford University*.

Ochsenbein, F., R. Williams, C. Davenhall, D. Durand, P. Fernique, R. Hanisch, D. Giaretta, T. McGlynn, A. Szalay, and A. Wicenec (2004). Votable: tabular data for the virtual observatory. *Toward an International Virtual Observatory. Springer*, 118–123.

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

Panov, P., S. Dzeroski, and L. Soldatova (2008). Ontodm: An ontology of data mining. *Proceedings of the 2008 IEEE International Conference on Data MIning Workshops*, 752–760.

Panov, P., L. Soldatova, and S. Dzeroski (2009). Towards an ontology of data mining investigations. *Lecture Notes in Artificial Intelligence 5808*, 257–271.

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

Pedersen, T. (2008). Empiricism is not a matter of faith. *Computational Linguistics 34*, 465–470.

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.

Perlich, C., F. Provost, and J. Simonoff (2003). Tree induction vs. logistic regression: A learning-curve analysis. *The Journal of Machine Learning Research 4*, 211–255.

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.

Podpecan, V., M. Jursic, M. Zakova, and N. Lavrac (2009). Towards a service-oriented knowledge discovery platform. *Proceedings of the SoKD-09 International Workshop on Third Generation Data Mining at ECML PKDD 2009*, 25–38.

Provost, F., T. Fawcett, and R. Kohavi (1998). The case against accuracy estimation for comparing induction algorithms. *Proceedings of the Fifteenth International Conference on Machine Learning*, 445–453.

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)*.

Roure, D. D., C. Goble, and R. Stevens (2009). The design and realisation of the myexperiment virtual research environment for social sharing of workflows. *Future Generation Computer Systems 25*, 561–567.

Rowe, A., D. Kalaitzopoulos, and M. Osmond (2003). The discovery net system for high throughput bioinformatics. *Bioinformatics 19*, 225–231.

Ryan, T. P. (2007). Modern experimental design. *Wiley Interscience*.

Sacerdoti, E. (1974). Planning in a hierarchy of abstraction spaces. *Artificial intelligence 5*(2), 115–135.

Sacks, J., W. Welch, T. Mitchell, and H. Wynn (1989). Design and analysis of computer experiments. *Statistical science 4*(4), 409–435.

Salzberg, S. (1999). On comparing classifiers: A critique of current research and methods. *Data mining and knowledge discovery 1*, 1–12.

Schaaff, A. (2007). Data in astronomy: From the pipeline to the virtual observatory. *Lecture Notes in Computer Science 4832*, 52–62.

Schaffer, C. (1994). A conservation law for generalization performance. *Proceedings of the International Conference on Machine Learning*, 259–265.

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.

Sirin, E. and B. Parsia (2007). Sparql-dl: Sparql query for owl-dl. *Third International Workshop on OWL Experiences and Directions (OWLED 2007)*.

Sirin, E., B. Parsia, D. Wu, J. Hendler, and D. Nau (2004). Htn planning for web service composition using shop2. *Journal of Web Semantics 1*(4), 377–396.

Sleeman, D., M. Rissakis, S. Craw, N. Graner, and S. Sharma (1995). Consultant-2: Pre-and post-processing of machine learning applications. *International journal of human-computer studies 43*(1), 43–63.

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.

Snedecor, G. W. and W. G. Cochran (1989). Statistical methods. eighth edition. *Iowa State University Press.*.

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.

Soldatova, L., A. Clare, A. Sparkes, and R. King (2006). An ontology for a robot scientist. *Bioinformatics 33*(14), 464–471.

Soldatova, L. and R. King (2006). An ontology of scientific experiments. *Journal of the Royal Society Interface 3*(11), 795–803.

Sonnenburg, S., M. Braun, C. Ong, S. Bengio, L. Bottou, G. Holmes, Y. LeCun, K. Muller, F. Pereira, C. E. Rasmussen, G. Ratsch, B. Scholkopf, A. Smola, P. Vincent, J. Weston, and R. Williamson (2007). The need for open source software in machine learning. *Journal of Machine Learning Research 8*, 2443–2466.

Spellman, P., M. Miller, J. Stewart, C. Troup, S. Chervitz, D. Bernhart, G. Sherlock, C. Ball, M. Lepage, M. Swiatek, W. Marks, J. Goncalves, S. Markel, D. Iordan, M. Shojatalab, A. Pizarro, J. White, R. Hubley, E. Deutsch, M. Senger, B. J. Aronow, A. Robinson, D. Bassett, C. J. S. Jr, and A. Brazma (2002). Design and implementation of microarray gene expression markup language (mage-ml). *Genome Biology 3*(9), RESEARCH0046.

Stajich, J. and H. Lapp (2006). Open source tools and toolkits for bioinformatics: significance, and where are we? *Briefings in Bioinformatics 7*(3), 287–296.

Stoeckert, C., H. Causton, and C. Ball (2002). Microarray databases: standards and ontologies. *nature genetics 32*, 469–473.

Studer, R., V. Benjamins, and D. Fensel (1998). Knowledge engineering: principles and methods. *Data & Knowledge Engineering 25*(2), 161–197.

Szalay, A. and J. Gray (2001). The world-wide telescope. *Science 293*, 2037–2040.

Talia, D., P. Trunfio, and O. Verta (2005). Weka4ws: a wsrf-enabled weka toolkit for distributed data mining on grids. *Lecture Notes in Computer Science 3721*, 309–320.

Taylor, I., M. Shields, I. Wang, and A. Harrison (2007). The triana workflow environment: Architecture and applications. *Workflows for e-Science. Springer.*, 320–339.

Thimbleby, H. (2003). Explaining code for publication. *Software Practice and Experience 33*(10), 975–1001.

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.

Utgoff, P. (1986). Shift of bias for inductive concept learning. *Machine learning: An artificial intelligence approach. Volume II. Morgan Kaufmann.*.

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*.

Vizcaino, J. A., R. Cote, F. Reisinger, J. M. Foster, M. Mueller, J. Rameseder, H. Hermjakob, and L. Martens (2009). A guide to the proteomics identifications database proteomics data repository. *Proteomics 9*(18), 4276–4283.

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.

Webb, G. and P. Conilione (2005). Estimating bias and variance from data. *Pre-publication manuscript (http://www.csse.monash.edu/ webb/-Files/WebbConilione06.pdf)*.

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.

Wilcoxon, F. (1945). Individual comparisons by ranking methods. *Biometrics 1*(6), 80–83.

Wirth, R., C. Shearer, U. Grimmer, T. Reinartz, J. Schlosser, C. Breitner, R. Engels, and G. Lindner (1997). Towards process-oriented tool support for knowledge discovery in databases. *Lecture Notes in Computer Science 1263*, 243–253.

Witten, I. and E. Frank (2005). Data mining: Practical machine learning tools and techniques (2nd edition). *Morgan Kaufmann*.

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

Wolpert, D. (2001). The supervised learning no-free-lunch theorems. *Proceedings of the 6th Online World Conference on Soft Computing*.

Yang, Q. and X. Wu (2006). 10 challenging problems in data mining research. *International Journal of Information Technology and Decicion Making 5*(4), 597–604.

Yasuda, N., Y. Mizumoto, M. Ohishi, W. O. amd T Budavári, V. Haridas, N. Li, T. Malik, A. Szalay, M. Hill, T. Linde, B. Mann, and C. Page (2004). Astronomical data query language: Simple query protocol for the virtual observatory. *ASP Conference Proceedings 314*, 293.

Záková, M., P. Kremen, F. Zelezny, and N. Lavrac (2008). Planning to learn with a knowledge discovery ontology. *Second planning to learn workshop at the joint ICML/COLT/UAI Conference*, 29–34.

Záková, M., V. Podpecan, F. Zelezný, and N. Lavrac (2009). Advancing data mining workflow construction: A framework and cases using the orange toolkit. *Proceedings of the SoKD-09 International Workshop on Third Generation Data Mining at ECML PKDD 2009*, 39–51.

Zhong, N., C. Liu, and S. Ohsuga (2001). Dynamically organizing kdd processes. *International Journal of Pattern Recognition and Artificial Intelligence 15*(3), 451–473.

Zhong, N., Y. Matsui, T. Okuno, and C. Liu (2002). Framework of a multi-agent kdd system. *Lecture Notes in Computer Science 2412*, 337–346.

Zhong, N. and S. Ohsuga (1994). The gls discovery system: its goal, architecture and current results. *Lecture Notes in Computer Science 869*, 233–244.

Publication List

Articles in internationally reviewed journals

- J. Vanschoren, H. Blockeel, B. Pfahringer, and G. Holmes *Experiment Databases: a new way to share, organize and learn from experiments*, Machine Learning, (Resubmitted after strong encouragement)

- J. Vanschoren, H. Blockeel *A platform for collaborative experimentation in machine learning*, Journal of Machine Learning Research (Submitted)

Contributions at international conferences

With rigorous selection

- J. Vanschoren, H. Blockeel *A community-based platform for machine learning experimentation*, Lecture Notes In Computer Science (ECML-2009), 5782, 750-754, 2009 **Winner of best demo award!**

- J. Vanschoren, B. Pfahringer, and G. Holmes *Learning from the past with experiment databases*, Lecture Notes in Artificial Intelligence (PRICAI-2008), 5351, 485-496, 2008

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

- J. Vanschoren, and H. Blockeel *Investigating classifier learning behavior with experiment databases*, Data Analysis, Machine Learning and Applications - Annual Conference of the Gesellschaft für Klassifikation (GfKL-2007), 421-428, 2008

- H. Blockeel[†] and J. Vanschoren[†] *Experiment databases: Towards an improved experimental methodology in machine learning.* Lecture notes in computer science (ECML-2007), 4702, 6-17, 2007

[†]First authors, ordered alphabetically

With less strict selection

- J. Vanschoren, H. Blockeel *Stand on the shoulders of giants: Towards a portal for collaborative experimentation in data mining.* Workshop on Third Generation Data Mining at ECML PKDD 2009, 88-99, 2009
- J. Vanschoren *Experiment databases for machine learning.* NIPS Workshop on Machine Learning Open Source Software at NIPS 2008, 2008
- J. Vanschoren, H. Blockeel, B. Pfahringer, and G. Holmes *Organizing the world's machine learning information*, Workshop on Third Generation Data Mining at ECML PKDD 2008, 2008
- J. Vanschoren, H. Blockeel, B. Pfahringer, and G. Holmes *Experiment databases: Creating a new platform for meta-learning research*, In : Proceedings of the ICML/UAI/COLT Joint Planning to Learn Workshop (PlanLearn-08), pp. 10-15, 2008
- J. Vanschoren, A. Van Assche, C. Vens, and H. Blockeel *Meta-learning from experiment databases: An illustration*, Annual Machine Learning Conference of Belgium and The Netherlands (Benelearn-07), pp. 120-127, 2007
- J. Vanschoren, H. Blockeel *Towards understanding learning behavior*, Annual Machine Learning Conference of Belgium and The Netherlands (Benelearn-06), pp. 89-96, 2006

Book Chapters

- J. Vanschoren *Meta-learning architectures. Collecting, organizing and exploiting meta-knowledge.* In: Meta-learning (N. Jankowski, W. Duch, K. Grabczewski, ed.), Springer. (In press)
- J. Vanschoren, H. Blockeel *Experiment Databases.* In: Inductive Databases and Constraint-Based Data Mining (S. Džeroski, B. Goethals, P. Panov, ed.), Springer. (In press)

Abstracts of presentations

- J. Vanschoren *Towards an ontology for machine learning*, Spring Workshop on Mining and Learning (SML-09), 2009
- J. Vanschoren *Experiment databases: Towards an improved experimental methodology in machine learning*, Symposium on Recent Trends in Data Mining, 2008

- H. Blockeel[†] and J. Vanschoren[†] *Experiment databases: Towards an improved experimental methodology in machine learning*, Proceedings of the 19th Belgian-Dutch Conference on Artificial Intelligence, pp 315-316, 2007

- J. Vanschoren *Investigating learning behavior with experiment databases*, Summer School on Algorithmic Data Analysis, 2007

- J. Vanschoren, and H. Blockeel *Investigating learning behavior with experiment databases*, Former Freiburg, Leuven and Friends Workshop on Machine Learning (FLF-07), 2007

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

- J. Vanschoren, and H. Blockeel *Towards understanding learning behavior*, Freiburg, Leuven and Friends Workshop on Machine Learning (FLF-06), 2006

- J. Vanschoren, and H. Blockeel *Towards understanding learning behavior*, First Research Contact Day of the CIL doctoral school, 2006

[†]First authors, ordered alphabetically

Arenberg Doctoral School of Science, Engineering & Technology
Faculty of Engineering
Department of Computer Science
Research group Declarative Languages and Artificial Intelligence
Celestijnenlaan 200A, B-3001 Leuven