

Cognitive Technologies

Managing Editors: D. M. Gabbay J. Siekmann

Editorial Board: A. Bundy J. G. Carbonell
M. Pinkal H. Uszkoreit M. Veloso W. Wahlster
M. J. Wooldridge

Advisory Board:

Luigia Carlucci Aiello
Franz Baader
Wolfgang Bibel
Leonard Bolc
Craig Boutilier
Ron Brachman
Bruce G. Buchanan
Anthony Cohn
Artur d'Avila Garcez
Luis Fariñas del Cerro
Koichi Furukawa
Georg Gottlob
Patrick J. Hayes
James A. Hendler
Anthony Jameson
Nick Jennings
Aravind K. Joshi
Hans Kamp
Martin Kay
Hiroaki Kitano
Robert Kowalski
Sarit Kraus
Maurizio Lenzerini
Hector Levesque
John Lloyd

Alan Mackworth
Mark Maybury
Tom Mitchell
Johanna D. Moore
Stephen H. Muggleton
Bernhard Nebel
Sharon Oviatt
Luis Pereira
Lu Ruqian
Stuart Russell
Erik Sandewall
Luc Steels
Oliviero Stock
Peter Stone
Gerhard Strube
Katia Sycara
Milind Tambe
Hidehiko Tanaka
Sebastian Thrun
Junichi Tsujii
Kurt VanLehn
Andrei Voronkov
Toby Walsh
Bonnie Webber

Pavel Brazdil · Christophe Giraud-Carrier
Carlos Soares · Ricardo Vilalta

Metalearning

Applications to Data Mining

With 53 Figures and 11 Tables



Springer

Authors:

Prof. Pavel Brazdil
LIAAD
Universidade do Porto
Fac. Economia
Rua de Ceuta 118-6°
4050-190 Porto, Portugal
pbrazdil@liaad.up.pt

Dr. Christophe Giraud-Carrier
Brigham Young University
Department of Computer Science
Provo, UT 84602, USA
cgc@cs.byu.edu

Dr. Carlos Soares
LIAAD
Universidade do Porto
Fac. Economia
Rua de Ceuta 118-6°
4050-190 Porto, Portugal
csoares@fep.up.pt

Dr. Ricardo Vilalta
University of Houston
Department of Computer Science
501 PGH Building
Houston, TX 77204-3010, USA
vilalta@cs.uh.edu

Managing Editors:

Prof. Dov M. Gabbay
Augustus De Morgan Professor of Logic
Department of Computer Science
King's College London
Strand, London WC2R 2LS, UK

Prof. Dr. Jörg Siekmann
Forschungsbereich Deduktions- und
Multiagentensysteme, DFKI
Stuhlsatzenweg 3, Geb. 43
66123 Saarbrücken, Germany

ISBN: 978-3-540-73262-4 e-ISBN: 978-3-540-73263-1
DOI: 10.1007/978-3-540-73263-1

Cognitive Technologies ISSN: 1611-2482

Library of Congress Control Number: 2008937821

ACM Computing Classification (1998): I.2.6, H.2.8

© Springer-Verlag Berlin Heidelberg 2009

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilm or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.

The use of general descriptive names, registered names, trademarks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

Cover design: KünkelLopka, Heidelberg

Printed on acid-free paper

9 8 7 6 5 4 3 2 1

springer.com

Dedication

Pavel

to my wife and lifelong companion, Fátima.

Christophe

to my wife and children.

Carlos

to Manela, Quica and Manel.

Ricardo

to my parents.

Preface

The continuous growth of successful applications in machine learning and data mining has led to an apparent view of real progress towards the understanding of the nature and mechanisms of learning machines. From the automated classification of millions of luminous objects stored in astronomical images, to the complex analysis of long sequences of genes in biomedical data sets, machine learning has positioned itself as an indispensable tool for data analysis, pattern recognition, and scientific discovery. This apparent progress in the search for accurate predictive models relies on the design of learning algorithms exhibiting novel functionalities. The history of machine learning shows a research community devoted to the study and improvement of a highly diverse set of learning algorithms such as nearest neighbors, Bayesian classifiers, decision trees, neural networks, and support vector machines (to name just a few). While the design of new learning algorithms is certainly important in advancing our ability to finding accurate data models, so is the understanding of the relation between data set characteristics and the particular mechanisms embedded in the learning algorithm. Rather than testing multiple algorithms to assess which one would perform satisfactorily on a certain data set, the end user needs guidelines pointing to the best learning strategy for the particular problem at hand. Researchers and practitioners in machine learning have a clear need to answer the following question: what works well where? There is a strong need to characterize both data distributions and learning mechanisms to construct a theory of learning behavior. Moreover, we advocate the development of a new generation of learning algorithms that are capable of profound adaptations in their behavior to the input data. This may include changes to the model language itself.

Despite different interpretations of the term *metalearning*, in this book we pursue the goal of finding principled methods that can make learning algorithms adaptive to the characteristics of the data. This can be achieved in many ways as long as there is some form of feedback relating learning performance with data distributions. Thus one can think of the problem of algorithm selection (or ranking), or algorithm combination, as frameworks

that exploit past performance to guide the selection of a final model. The ultimate end is to design learning algorithms that adapt to the problem at hand, rather than invoking the same fixed mechanisms independent of the nature of the data under analysis. We explain in this book that a unifying theme in metalearning is that of exploiting experience or metaknowledge to achieve flexible learning systems.

These ideas have brought us together to write a book that summarizes the current state of the art in the field of metalearning. The motivation for such a book can be traced back to the METAL project [166], in which the first three authors were active participants. Our first (electronic) meetings regarding this book took place in the second half of 2005 and have continued over the following three years until the second half of 2008. The project proved challenging in many ways, most particularly in unifying our view concerning the scope of the term metalearning. After long discussions we finally agreed on the definition provided in Chapter 1. Equally challenging was to decide on a list of topics that stand as clearly representative of the field. We hope the reader will find our selection appropriate and sufficiently broad to offer adequate coverage. Finally, it is our hope that this book will serve not only to place many of the ideas now dispersed all over the field into a coherent volume, but also to encourage researchers in machine learning to consider the importance of this fascinating area of study.

Book Organization

The current state of diverse ideas in the field of metalearning is not yet mature enough for a textbook based on a solid conceptual and theoretical framework of learning performance. Given this, we have decided to cover the main topics where there seems to be a clear consensus regarding their relevance and legitimate membership in the field. In the following, we briefly describe the contents of the book acknowledging the contribution of each of the authors. In Chapter 1, all of us worked on introducing the main ideas and concepts that we believe are essential to understanding the field of metalearning. Chapters 2–4 have a more practical flavor, illustrating the important problem of selecting and ranking learning algorithms, with a description of several currently operational applications. In Chapter 2, Soares and Brazdil describe a simple meta-learning system that, given a dataset, provides the user with guidance concerning which learning algorithm to use. The issues that are involved in developing such a system are discussed in more detail by Soares in Chapter 3, including a survey of existing approaches to address them. In Chapter 4, Giraud-Carrier describes a number of systems that incorporate some form of automatic user guidance in the data mining process.

Chapters 5–7, on the other hand, have a more conceptual flavor, covering the combination of classifiers, learning from data streams, and knowledge transfer. Chapter 5, authored by Giraud-Carrier, describes the main concepts behind model combination, including classical techniques such as bagging and

boosting, as well as more advanced techniques such as delegating, arbitrating and meta-decision trees. We invited Gama and Castillo to contribute to Chapter 6; the chapter discusses the dynamics of the learning process and general strategies for reasoning about the evolution of the learning process itself. The main characteristics and new constraints on the design of learning algorithms imposed by large volumes of data that evolve over time are described, including embedding change-detection mechanisms in the learning algorithm and the trade-off between the cost of update and the gain in performance. Chapter 7, authored by Vilalta, covers the important topic of knowledge transfer across tasks; the chapter covers topics such as multitask learning, transfer in kernel methods, transfer in parametric Bayesian methods, theoretical models of learning to learn, and new challenges in transfer learning with examples in robotics. Lastly, Chapter 8, authored by Brazdil, discusses the important role of metalearning in the construction of complex systems through the composition of induced subsystems. It is shown how domain-specific metaknowledge can be used to facilitate this task.

Acknowledgements

We wish to express our gratitude to all those who helped in bringing this project to fruition. We are grateful to the University of Porto and Faculty of Economics and also to the Portuguese funding organization FCT for supporting the R&D laboratory LIAAD (Laboratory of Artificial Intelligence and Decision Support) where a significant part of the work associated with this book was carried out. We also acknowledge support from the Portuguese funding organization FCT for the project ALES II – Adaptive LEarning Systems. This work was also partially supported by the US National Science Foundation under grant IIS-0448542.

The motivation for this book came from the involvement of the first three authors in an earlier grant from the European Union (ESPRIT project METAL [166]). We would like to acknowledge financial support from this grant and the contribution of our colleagues: Hilan Bensusan, Helmut Berrer, Sašo Džeroski, Peter Flach, Johannes Fürnkranz, Melanie Hilario, Jörg Keller, Tom Khabaza, Alexandros Kalousis, Petr Kuba, Rui Leite, Guido Lindner, Reza Nakhaeizadeh, Iain Paterson, Yonghong Peng, Rui Pereira, Johann Petrak, Bernhard Pfahringer, Luboš Popelínský, Ljupčo Todorovski, Dietrich Wettschereck, Gerhard Widmer, Adam Woznica and Bernard Zenko.

We are greatly indebted to several other colleagues for their many comments and suggestions that helped improve earlier versions of this book: Bart Bakker, Theodoros Evgeniou, Tom Heskes, Rich Maclin, Andreas Maurer, Tony Martinez, Massimiliano Pontil, Rajat Raina, André Rossi, Peter Stone, Richard Sutton, Juergen Schmidhuber, Matthew Taylor, Lisa Torrey and Roberto Valerio.

Pavel Brazdil would also like to gratefully acknowledge that Robert Kowalski drew his attention to the topic of metareasoning in the 70's and

to express his gratitude to the late Donald Michie for having accepted the proposal to include metalearning in the preparatory stages of the StatLog project [169] in 1989.

Finally, we are most grateful to our editor at Springer, Ronan Nugent, for his patience, gentle prodding and encouragement throughout this project.

Porto, Portugal; Provo, Houston, USA
July 2008

Pavel Brazdil
Christophe Giraud-Carrier
Carlos Soares
Ricardo Vilalta

Contents

1 Metalearning: Concepts and Systems 1

2 Metalearning for Algorithm Recommendation:
an Introduction 11

3 Development of Metalearning Systems for Algorithm
Recommendation 31

4 Extending Metalearning to Data Mining and KDD 61

5 Combining Base-Learners 73

6 Bias Management in Time-Changing Data Streams 91

7 Transfer of Metaknowledge Across Tasks 109

8 Composition of Complex Systems: Role
of Domain-Specific Metaknowledge 129

References 153

A Terminology 171

B Mathematical Symbols 173

Index 175

Metalearning: Concepts and Systems

1.1 Introduction

Current data mining (DM) and machine learning (ML) tools are characterized by a plethora of algorithms but a lack of guidelines to select the right method according to the nature of the problem under analysis. Applications such as credit rating, medical diagnosis, mine-rock discrimination, fraud detection, and identification of objects in astronomical images generate thousands of instances for analysis with little or no additional information about the type of analysis technique most appropriate for the task at hand. Since real-world applications are generally time-sensitive, practitioners and researchers tend to use only a few available algorithms for data analysis, hoping that the set of assumptions embedded in these algorithms will match the characteristics of the data. Such practice in data mining and the application of machine learning has spurred the research community to investigate whether learning from data is made of a single operational layer – search for a good model that fits the data – or whether there are in fact several operational layers that can be exploited to produce an increase in performance over time. The latter alternative implies that it should be possible to learn about the learning process itself, and in particular that a system could learn to profit from previous experience to generate additional knowledge that can simplify the automatic selection of efficient models summarizing the data.

This book provides a review and analysis of a research direction in machine learning and data mining known as *metalearning*.¹ From a practical standpoint, the goal of metalearning is twofold. On the one hand, we wish to overcome some of the challenges faced by users with current data analysis tools. The aim here is to aid users in the task of selecting a suitable predictive model (or combination of models) while taking into account the domain of application. Without some kind of assistance, model selection and combination

Meta-
learning

¹ We assume here that the reader is familiar with concepts in machine learning. Many books that provide a clear introduction to the field of machine learning are now available (e.g., [82, 26, 3, 174]).

can turn into solid obstacles to end users who wish to access the technology more directly and cost-effectively. End users often lack not only the expertise necessary to select a suitable model, but also the availability of many models to proceed on a trial-and-error basis. A solution to this problem is attainable through the construction of metalearning systems that provide automatic and systematic user guidance by mapping a particular task to a suitable model (or combination of models).

On the other hand, we wish to address a problem commonly observed in the practical use of data analysis tools, namely how to profit from the repetitive use of a predictive model over similar tasks. The successful application of models in real-world scenarios requires continuous adaptation to new needs. Rather than starting afresh on new tasks, one would expect the learning mechanism itself to relearn, taking into account previous experience (e.g., [50, 254, 193]). This area of research, also known as *learning to learn*, has seen many new developments in the past few years. Here too, metalearning systems can help control the process of exploiting cumulative expertise by searching for patterns across tasks.

Our goal in this book is to give an overview of the field of metalearning by attending to both practical and theoretical concepts. We describe the current state of the art in different topics such as techniques for algorithm recommendation, extending metalearning to cover data mining and knowledge discovery, combining classifiers, time-changing data streams, inductive transfer or transfer of metaknowledge across tasks, and composition of systems and applications. Our hope is to stimulate the interest of both practitioners and researchers to invest more effort in this interesting field of research. Despite the promising directions offered by metalearning and important recent advances, much work remains to be done. We also hope to convince others of the important task of expanding the adaptability of current computer learning systems towards understanding their own learning mechanisms.

1.1.1 Base-Learning vs. Metalearning

We begin by clarifying the distinction between the traditional view of learning – also known as base-learning – and the one taken by metalearning. Metalearning differs from base-learning in the scope of the level of adaptation; whereas learning at the base level is focused on accumulating experience on a specific learning task, learning at the meta level is concerned with accumulating experience on the performance of multiple applications of a learning system. In a typical inductive learning scenario, applying a base-learner (e.g., decision tree, neural network, or support vector machine) on some data produces a predictive function (i.e., hypothesis) that depends on the *fixed* assumptions embedded in the learner. Learning takes place at the base level because the quality of the function or hypothesis normally improves with an increasing number of examples. Nevertheless, successive applications of the

Learning to
learn

Base-
learning

learner on the same data always produces the same hypothesis, independently of performance; no knowledge is extracted across domains or tasks.

As an illustration, consider the task of learning to classify medical patients in a hospital according to a list of potential diseases. Given a large dataset of patients, each characterized by multiple parameters (e.g., blood type, temperature, blood pressure, medical history, etc.) together with the diagnosed disease (or alternatively no disease), one can train a learning algorithm to predict the right disease for a new patient. The resulting predictive function normally improves in accuracy as the list of patients increases. This is learning at the base level where additional examples (i.e., patients) provide additional statistical support to unveil the nature of patterns hidden in the data.

Working at the base level exhibits two major limitations. First, data patterns are usually not placed aside for interpretation and analysis, but rather embedded in the predictive function itself. Successive training of the learning algorithm over the same data fails to accumulate any form of experience. Second, data from other hospitals can seldom be exploited unless one merges all inter-hospital patient data into a single file. The experience or knowledge gained when applying a learning algorithm using data from one hospital is thus generally not readily available as we move to other hospitals. A key to solving these problems is gathering knowledge about the learning process, also known as *metaknowledge*. Such knowledge can be used to improve the learning mechanism itself after each training episode. Metaknowledge may take on different forms and applications, and can be defined as any kind of knowledge that is derived in the course of employing a given learning system. Advances in the field of metalearning hinge on the acquisition and effective exploitation of knowledge about learning systems (i.e., metaknowledge) to understand and improve their performance.

Meta-
knowledge

1.1.2 Dynamic Bias Selection

The field of metalearning studies how learning systems can become more effective through experience. The expectation is not simply that a good solution be found, but that this be done increasingly more effectively through time. The problem can be cast as that of determining the right bias for each task. The notion of learning bias is at the core of the study of machine learning. Bias refers to any preference for choosing one hypothesis explaining the data over other (equally acceptable) hypotheses, where such preference is based on extra-evidential information independent of the data (see [173, 112] for other similar definitions of bias).

Learning
bias

Unlike base-learning, where the bias is *fixed* a priori or user-parameterized, metalearning studies how to choose the most adequate bias dynamically. The view presented here is aligned with that formulated originally by Rendell et al. [206]: *Metalearning is to learn from experience when different biases are appropriate for a particular problem*. This definition leaves some important issues unresolved, such as the role of metaknowledge (explained below) and

how the process of adaptation takes place. We defer giving our own definition of metalearning (until Section 1.3) after we provide additional concepts through a brief overview on the contents of the book.

Declarative
bias

Metalearning covers both declarative and procedural bias. Declarative bias specifies the representation of the space of hypotheses, and affects the size of the search space (e.g., represent hypotheses using linear functions only, or conjunctions of attribute values). Procedural bias imposes constraints on the ordering of the inductive hypotheses (e.g., prefer smaller hypotheses). Both types of bias affect the effectiveness of a learning system on a particular task. Searching through the (declarative and procedural) bias space causes a metalearning algorithm to engage in a time-consuming process. An important aim in metalearning is to exploit metaknowledge to make the search over the bias space manageable.

Procedural
bias

In the following introductory sections we discuss how metaknowledge can be employed in different settings. We consider for instance the problem of selecting learning algorithms. We then broaden the analysis to discuss the impact of metalearning on knowledge discovery and data mining. Finally, we extend our analysis to adaptive learning, transfer of knowledge across domains and composition of complex systems, and the role metaknowledge plays in each situation.

1.2 Employing Metaknowledge in Different Settings

We proceed in this section by showing that knowledge gained through experience can be useful in many different settings. Our approach is to provide a brief introduction – a foretaste – of what is contained in the remainder of the book. We begin by considering the general problem of selecting machine learning (ML) algorithms for a particular application.

1.2.1 Selecting and Recommending Machine Learning Algorithms

Consider the problem of selecting or recommending a suitable subset of ML algorithms for a given task. The problem can be cast as a search problem, where the search space includes the individual ML algorithms, and the aim is to identify the set of learning algorithms with best performance. A general framework for selecting learning algorithms is illustrated in Figure 1.1. According to this framework, the process can be divided into two phases. In the first phase the aim is to identify a suitable subset of learning algorithms given a training dataset (Figure 1.1a), using available metaknowledge (Figure 1.1c). The output of this phase is a ranked subset of ML algorithms (Figure 1.1d), which represents the new, reduced bias space. The second phase of the process then consists of searching through the reduced space. Each learning algorithm is evaluated using various performance criteria (e.g., accuracy, precision, recall, etc.) to identify the best alternative (Figure 1.1e).

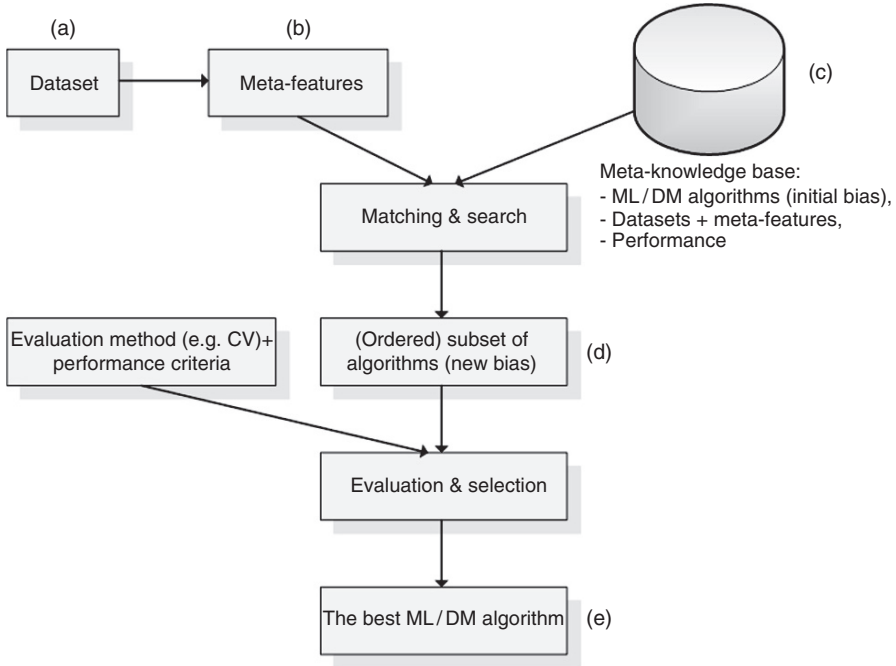


Fig. 1.1. Selection of ML/DM algorithms: finding a reduced space and selecting the best learning algorithm

The above framework differs from traditional approaches in that it exploits a metaknowledge base. As previously mentioned, one important aim in metalearning is to study how to extract and exploit metaknowledge to benefit from previous experience. Information contained in the metaknowledge base can take different forms. It may include, for instance, a set of learning algorithms that have shown good (a priori) performance on datasets similar to the one under analysis; algorithms to characterize ML algorithms and datasets and metrics available to compute dataset similarity or task relatedness. Hence, metaknowledge encompasses not only information useful to perform dynamic bias selection, but also functions and algorithms that can be invoked to generate new useful information.

We note that metaknowledge does not generally completely eliminate the need for search, but rather provides a more effective way of searching through the space of alternatives. It is clear that the effectiveness of the search process depends on the quality of the available metaknowledge.

1.2.2 Generation of Metafeatures

Following the above example, one may ask how the subset of ML algorithms is identified. One form of metaknowledge used during the first phase refers

to dataset characteristics or metafeatures (Figure 1.1b); these provide valuable information to differentiate the performance of a set of given learning algorithms. The idea is to gather descriptors about the data distribution that correlate well with the performance of learned models. This is a particularly relevant contribution of metalearning to the field of machine learning, as most work in machine learning focuses instead on the design of multiple learning architectures with a variety of resulting algorithms. Little work has been devoted to understanding the connection between learning algorithms and the characteristics of the data under analysis.

So far, three main classes of metafeatures have been proposed. The first one includes features based on statistical and information-theoretic characterization. These metafeatures, estimated from the dataset, include number of classes, number of features, ratio of examples to features, degree of correlation between features and target concept and average class entropy [1, 88, 106, 120, 169, 238]. This method of characterization has been used in a number of research projects that have produced positive and tangible results (e.g., ESPRIT Statlog and METAL).

A different form of dataset characterization exploits properties of some induced hypothesis. As an example of this model-based approach, one can build a decision tree from a dataset and collect properties of the tree (e.g., nodes per feature, maximum tree depth, shape, tree imbalance, etc.), to form a set of metafeatures [22, 188].

Finally, a different idea is to exploit information obtained from the performance of a set of simple and fast learners that exhibit significant differences in their learning mechanism [20, 190]. The accuracy of these so-called *landmarkers* is used to characterize a dataset and identify areas where each type of learner can be regarded as an expert [104, 237].

The measures discussed above can be used to identify a subset of accurate models by invoking a meta-level system that maps dataset characteristics to models. As an example, work has been done with the k -Nearest Neighbor method (k -NN) at the meta level to identify the most similar datasets for a given input dataset [41]. For each of the neighbor datasets, one can generate a ranking of the candidate models based on their particular performance (e.g., accuracy, learning time, etc.). Rankings can subsequently be aggregated to generate a final recommended ranking of models. More details on these issues are discussed in Chapters 2 and 3.

1.2.3 Employing Metalearning in KDD and Data Mining

The algorithm selection framework described above can be further generalized to the KDD/DM process. Consider again Figure 1.1, but this time assume that the output of the system is not a learning algorithm but a flexible planning system. The proposed extension can be justified as follows. Typically, the KDD process is represented in the form of a sequence of operations, such as data selection, preprocessing, model building, and post-processing,

Simple,
statistical
and
information-
theoretic
metafea-
tures

Model-
based
metafea-
tures

Land-
markers

KDD/DM
process

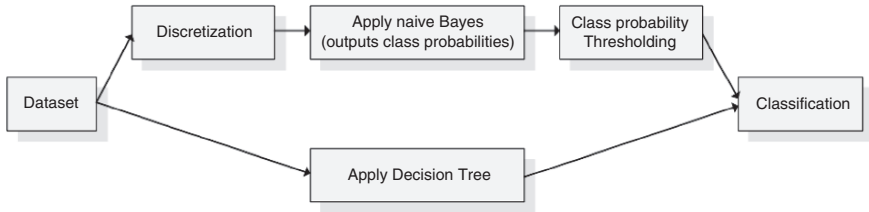


Fig. 1.2. Example of a partial order of operations (plan)

among others. Individual operations can be further decomposed into smaller operations. Operations can be characterized as simple sequences, or, more generally, as partially ordered acyclic graphs. An example of a simple partial order of operations is shown in Figure 1.2 (this example has been borrowed and adapted from [24]). Every partial order of operations can be regarded as an executable plan. When executed, the plan produces certain effects (for instance, classification of input instances).

Partial
order of
operations
Plan

Under this extended framework, the task of the data miner is to elaborate a suitable plan. In general the problem of generating a plan may be formulated as that of identifying a partial order of operations, so as to satisfy certain criteria and (or) maximize certain evaluation measures. Producing good plans is a non-trivial task. The more operations there are, the more difficult it is to arrive at an optimal (or near-optimal) solution.

A plan can be built in two ways. One is by placing together individual constituents, starting from an empty plan and gradually extending it through the composition of operators (as in [24]). Another possibility is to consider previous plans, identify suitable ones for a given problem, and adapt them to the current situation (e.g., see [176]).

Although any suitable planning system can be adopted to implement these ideas, it is clear that the problem is inherently difficult. One needs to consider many possible operations, some of them with high computational complexity (e.g., training a classifier on large datasets). Metaknowledge can be used to facilitate this task. Existing plans can be seen as embodying certain procedural metaknowledge about the compositions of operations that have proved useful in past scenarios. This can be related to the notion of macro-operators in planning. Knowledge can also be captured about the applicability of existing plans to support reuse. Finally, one can also try to capture knowledge describing how existing plans can be adapted to new circumstances. Many of these issues are discussed in Chapter 4.

1.2.4 Employing Metalearning to Combine Base-Level ML Systems

A variation on the theme of combining DM operations, discussed in the previous section, is found in the work on model combination. By drawing on information about base-level learning, in terms of the characteristics of either

Model com-
bination

various subsets of data or various learning algorithms, model combination seeks to build composite learning systems with stronger generalization performance than their individual components. Examples of model combination approaches include boosting, stacked generalization, cascading, arbitrating and meta-decision trees.

Because it uses results at the base level to construct a learner at the meta level, model combination may clearly be regarded as a form of metalearning. Although many approaches focus exclusively on using such metalearning to achieve improved accuracy over base-level learning, some of them offer interpretable insight into the learning process by deriving explicit metaknowledge in the combination process. Model combination is the subject of Chapter 5.

1.2.5 Control of the Learning Process and Bias Management

We have discussed the issue of how metaknowledge can be exploited to facilitate the process of learning (Figure 1.1). We now consider situations where the given dataset is very large or potentially infinite (e.g., processes modeled as continuous data streams).

We can distinguish among several situations. For example, consider the case where the dataset is very large (but not infinite). Assume we have already chosen a particular ML algorithm and the aim is to use an appropriate strategy to mitigate the large dataset problem. Different methods are described in the literature to cope with this problem. Some rely on data reduction techniques, while others provide new functionalities on existing algorithms [99].

One well-known strategy relies on *active learning* [281] in which examples are processed in batches: the initial model (e.g., a decision tree) is created from the first batch and, after the initial model has been created, the aim is to select *informative examples* from the next batch while ignoring the rest.

The idea of controlling the process of learning can be taken one step further. For example, metalearning can be done dynamically, where the characterization of a new dataset is done progressively, testing different algorithms on samples of increasing size. The results in one phase determine what should be done in the next. The aim is to reduce the bias error (by selecting the most appropriate base-algorithm) effectively.

Another example involves learning from data streams. Work in this area has produced a control mechanism that enables us to select different kinds of learning system as more data becomes available. For instance, the system can initially opt for a simple naïve bayes classifier, but, later on, as more data becomes available, switch to a more complex model (e.g., bayesian network²).

In Section 1.2.1, we saw how data characteristics can be used to preselect a subset of suitable models, thus reducing the space of models under consideration. In learning from data streams, the control mechanism is activated in

² The description of naïve bayes and bayesian networks can be found in many books on machine learning. See, e.g., [174].

a somewhat different way. The quantity of data and data characteristics are used to determine whether the system should continue with the same model or take corrective action. If a change of model appears necessary, the system can extend the current model or even relearn from scratch (e.g., when there is a concept shift). Additionally, the system can decide that a switch should be made from one model type to another. More details on these issues can be found in Chapter 6.

1.2.6 Transfer of (Meta)Knowledge Across Domains

Another interesting problem in metalearning consists of finding efficient mechanisms to transfer knowledge across domains or tasks. Under this view, learning can no longer be simply seen as an isolated task that starts accumulating knowledge afresh on every new problem. As more tasks are observed, the learning mechanism is expected to benefit from previous experience. Research in *inductive transfer* has produced multiple techniques and methodologies to manipulate knowledge across tasks [192, 258]. For example, one could use a *representational transfer* approach where knowledge is first generated in one task, and subsequently exploited to help in another task. Alternatively one can use a *functional transfer* approach where various tasks are learned simultaneously; the latter case is exemplified in what is known as *multitask learning*, where the output nodes in a multilayer network represent more than one task and internal nodes are shared by different tasks dynamically during learning [50, 51].

Transfer of
knowledge

In addition, the theory of metalearning has been enriched with new information quantifying the benefits gained by exploiting previous experience [16]. Classical work in learning theory bounding the true risk as a function of the empirical risk (employing metrics such as the Vapnik-Chervonenkis dimension) has been extended to deal with scenarios made of multiple tasks. In this case the goal of the metalearner is to output a hypothesis space with a learning bias that generates accurate models for a new task. More details concerning this topic are given in Chapter 7.

Meta-
learner

1.2.7 Composition of Complex Systems and Applications

An attractive research avenue for future knowledge engineering is to employ ML techniques in the construction of new systems. The task of inducing a complex system can then be seen as a problem of inducing the constituting elements and integrating them. For instance, a text extraction system may be composed of various subsystems, one oriented towards tagging, another towards morphosyntactic analysis and yet another towards word sense disambiguation, and so on. This idea is somewhat related to the notion of layered learning [243, 270].

Composition
of complex
systems

If we use the terminology introduced earlier, we can see this as a problem of planning to resolve multiple (interacting) tasks. Each task is resolved using

a certain ordering of operations (Section 1.2.3). Metalearning here can help in retrieving previous solutions conceived in the past and reusing them in new settings. More details concerning this topic are given in Chapter 8.

1.3 Definition, Scope, and Organization

We have introduced the main ideas related to the field of metalearning covered by this book. Our approach has been motivated by both practical and theoretical aspects of the field. Our aim was to present the reader with diverse topics related to the term metalearning. We note that different researchers hold different views of what the term metalearning exactly means. To clarify our own view and to limit the scope of what is covered in this book, we propose the following definition:

Metalearning is the study of principled methods that exploit metaknowledge to obtain efficient models and solutions by adapting machine learning and data mining processes.

Our definition emphasizes the notion of metaknowledge. We claim a unifying point in metalearning lies in how to exploit such knowledge acquired on past learning tasks to improve the performance of learning algorithms. The answer to this question is key to the advancement of the field and continues being the subject of intensive research.

The definition also mentions machine learning processes; each process can be understood as a set of operations that form a learning mechanism. In this sense, a process can be a preprocessing step to learning (e.g., feature selection, dimensionality reduction, etc.), an entire learning algorithm, or a component of it (e.g., parameter adjustment, data splitting, etc.). The process of adaptation takes place when we replace, add, select, remove or change an existing operation (e.g., selecting a learning algorithm, combining learning algorithms, changing the value for a capacity control parameter, adding a data preprocessing step, etc.). The definition is then broad enough to capture a large set of possible ways to adapt existing approaches to machine learning.

The last goal is to produce efficient models under the assumption that bias selection is improved when guided by experience gained from past performance. A model will often be predictive in that it will be used to predict the class of new data instances, but other types of models (e.g., descriptive ones) will also be considered.

Metalearning for Algorithm Recommendation: an Introduction

2.1 Introduction

Data mining applications normally involve preparation of a dataset that can be processed by a learning algorithm (Figure 2.1). Given that there are usually several algorithms available, the user must select one of them. Additionally, most algorithms have parameters which must be set, so, after choosing the algorithm, the user must decide the values for each one of its parameters. The choice of algorithm is guided by some kind of *metaknowledge*, that is, knowledge that relates the characteristics of datasets with the performance of the available algorithms. This chapter describes how a simple metalearning system can be developed to generate metaknowledge that can be used to make recommendations concerning which algorithm to use on a given dataset. More details about various options are described in the next chapter.

As there are many alternative algorithms for a given task (for instance, decision trees, neural networks and support vector machines can be used for classification), the approach of trying out all alternatives and choosing the best one becomes infeasible. Although, normally, only a limited number of existing methods are available for use in a given application, the number of these methods may still be too large to rule out extensive experimentation. An approach followed by many users is to make some preselection of a small number of alternatives based on knowledge about the data and the available methods. The methods are applied to the dataset and the best one is normally chosen taking into account the results obtained. Although feasible, this approach may still require considerable computing time. Additionally, it requires that a highly skilled expert preselect the alternatives, and even the most skilled expert may sometimes fail, and so the best option may be left out.

It is thus important to develop methods to reduce the number of alternatives to be experimented with. The need for such methods has been recognized both in machine learning (e.g., [174, ch. 1]) and data mining (e.g., [34]). For instance, a survey of data mining applications in the Netherlands has identified

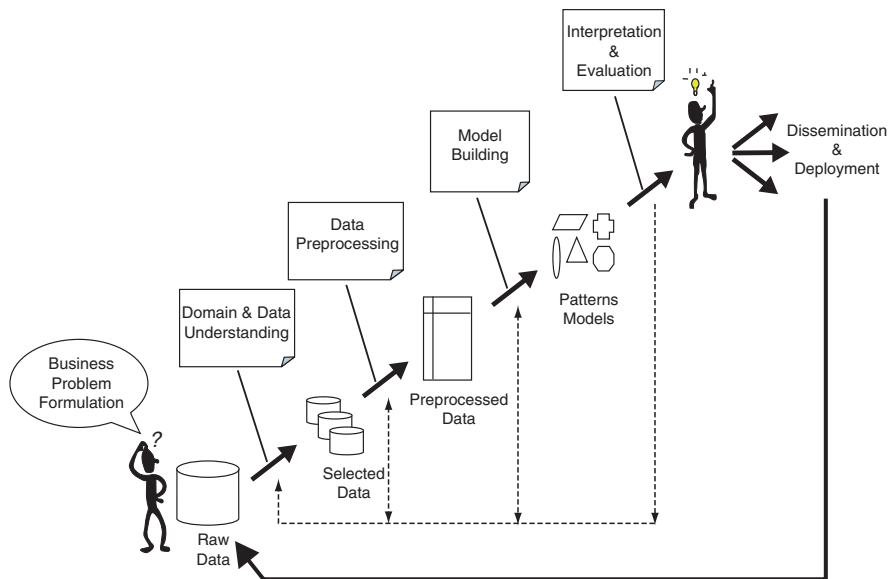


Fig. 2.1. The data mining process: after the dataset is prepared (*Data Preprocessing*), an algorithm to process it must be selected (*Model Building*)

the “lack of procedures and tools to support the search for the best technique” as an important problem [274]. In a panel at the 2001 KDD conference, the need for “automatic, data-dependent selection of data mining parameters and algorithms” was recognized as an important research issue [109]. This point was reiterated by Fogelman [111] in another panel discussion held at one of the 2006 KDD workshops. The problem of algorithm recommendation has been addressed in several European research projects, namely ML Toolbox [230], StatLog [169], METAL [166] and MiningMart [176], each of them contributing important advances.

This chapter shows how metalearning can be used for recommendation of learning algorithms. A simple system is described for illustration purposes, focusing on classification algorithms (Section 2.2). The specificities of the algorithm recommendation task requires that a suitable methodology be used for evaluation of metalearning systems. One such methodology is described in Section 2.3. Finally, an approach to adapt the metalearning system described for the task of recommending parameter settings is presented in Section 2.4. Some of the issues involved in the development of metalearning systems for algorithm recommendation are identified. A more thorough discussion of those issues is given in the next chapter.

2.2 Algorithm Recommendation with Metalearning

A system for algorithm recommendation can be defined as a tool that supports the user in the algorithm selection step of the data mining process (Figure 2.1). Given a dataset, it indicates which algorithm should be used to achieve the best possible results. If sufficient computational resources are available to try several algorithms, it should also indicate which ones should be executed and in which order. It is possible to say that, in practice, such a system guides the experimental process in a data mining application.

Algorithm
recommen-
dation

From the point of view of the user, the goal of algorithm recommendation can be stated as

Save time by reducing the number of alternative algorithms tried out on a given problem with minimal loss in the quality of the results obtained when compared to the best possible ones.

To achieve this goal it is not as important for an algorithm recommendation method to accurately predict the *true performance* of the algorithms as it is to predict their *relative performance*. Therefore, the task of algorithm recommendation can be defined as the ranking of algorithms according to their predicted performance.

Meta-
learning

To address this problem using a machine learning approach it is necessary to use data describing the performance of algorithms and the characteristics of problems, which we will refer to as *metadata* (Figure 2.2). Performance

Metadata

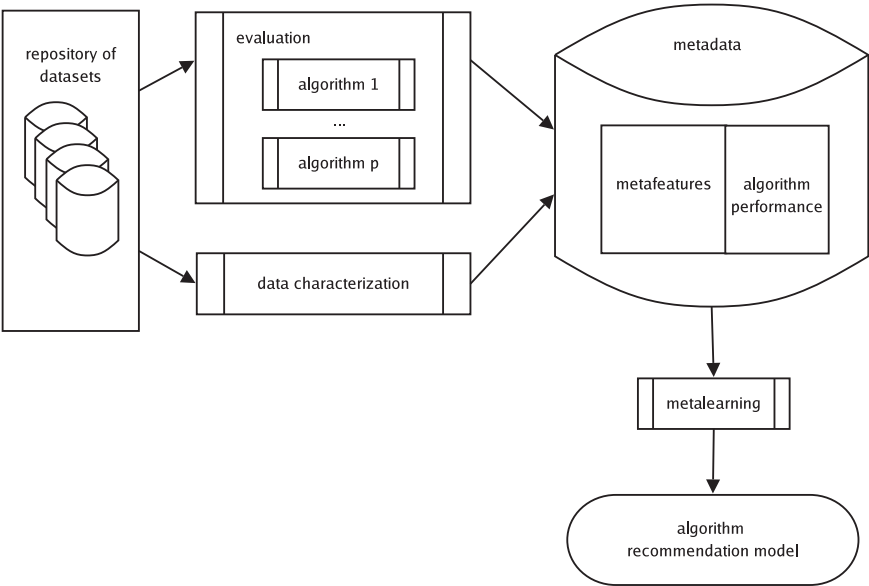


Fig. 2.2. Metalearning to obtain metaknowledge for algorithm selection

data are used to compute the rankings of the algorithms. These rankings, referred to as *target rankings*, are the target feature of this learning task. The measures that are used to characterize the problems represent features independent of the particular task. Here they are referred to as *metafeatures*. Metadata will be discussed in the following section and, for the moment, it is assumed that such data are available. Based on these concepts, metalearning with the purpose of developing systems for algorithm recommendation can be defined as:

Metalearning is the use of a machine learning approach to generate metaknowledge mapping the characteristics of problems (metafeatures) to the relative performance of algorithms.

This definition is similar to that used in the StatLog project [39] and is somewhat more specific than the one given in Chapter 1.

2.2.1 *k*-Nearest Neighbors Ranking Method

For illustration purposes, we describe how a simple learning method, the *k*-nearest neighbors (*k*-NN), can be adapted for the task of ranking classification algorithms [234, 41].

The *k*-NN algorithm is a very simple form of Instance-Based Learning (IBL)¹ [174]. In the IBL approach to induction, learning simply consists of storing the training examples. The prediction for a new example (dataset) is generated in two steps:

1. Select a set of training examples containing the ones that are most similar to the new example (dataset) in terms of their description (i.e., the values of the features).
2. Combine the target values of all the selected examples to generate a prediction for the new example (dataset).

The similarity between examples is usually based on some simple distance measure (e.g., Euclidean). Predictions are also generated using simple rules like the *majority* class for classification problems and the *mean* value for regression problems.

Next, we describe how each of these two components can be adapted for the task of learning rankings (Algorithm 2.1 and Figure 2.3). We also present an example of a ranking predicted using the *k*-NN ranking method.

Distance Function

The set of distance functions that can be used in the *k*-NN algorithm depends on the types of features that are used to describe examples (e.g., continuous,

¹ Locally weighted learning algorithms, which include the *k*-NN, are thoroughly discussed by Atkeson et al. [9].

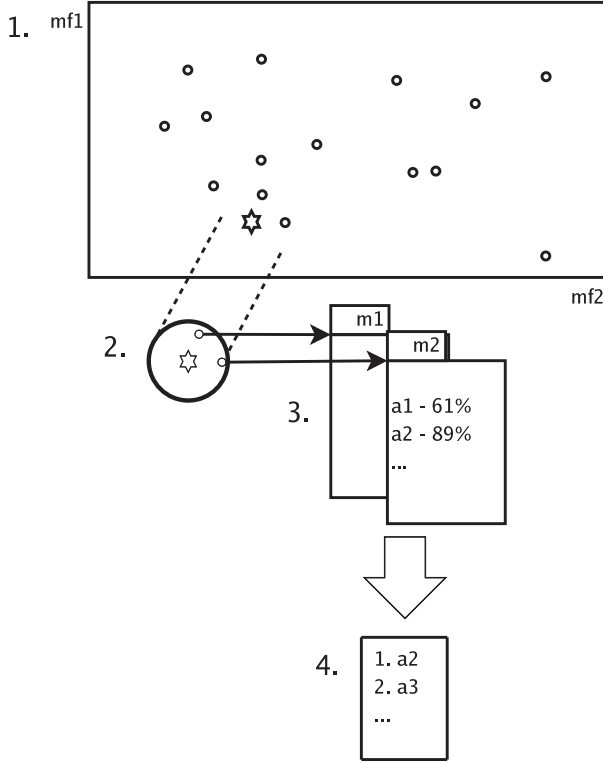


Fig. 2.3. The k -nearest neighbors algorithm for ranking

discrete) and not on the type of task (i.e., the target feature). As only continuous and binary metafeatures are considered in this example, any common distance measure, such as unweighted (or weighted) Euclidean distance, may be used.² Here, the k -NN method is illustrated using the unweighted L_1 norm:

$$distance(i, j) = \sum_{p=1}^m \frac{|x_{i,p} - x_{j,p}|}{\max_l(x_{l,p}) - \min_l(x_{l,p})} \quad (2.1)$$

where $\mathbf{x}_i = (x_{i,1}, x_{i,2}, \dots, x_{i,m})$ is the metafeature vector of meta-example i and m is the number of metafeatures. The distance value for each metafeature is normalized by dividing it by the corresponding range of values.

Prediction Method

The second step in the IBL approach is to generate a prediction based on the target values of the selected examples (representing datasets). The k examples

² Distance measures are discussed by Atkeson et al. [9].

```

input :  $\mathbf{T} = \{(\mathbf{x}_i, \mathbf{p}_i)\}_{i=1}^m$  // Training metadata where  $\mathbf{x}_i$  are the
        metafeatures of dataset  $i$  and  $\mathbf{p}_i$  are the performance estimates
        associated with dataset  $i$ 
         $T = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$  // New dataset
         $k$  // Number of neighbors
output:  $\mathbf{R} = \langle r_1, \dots, r_n \rangle$  // The recommended ranking for dataset
         $T$ , where  $r_j = i$  means that algorithm  $a_i$  is ranked in position  $j$ 
        and  $n$  is the number of algorithms
begin
    // Characterize the new dataset  $T$ 
     $\mathbf{x}_T \leftarrow \text{metafeatures}(T)$ 
    // Identify  $k$  datasets in metadata  $\mathbf{T}$  that are most similar to
    new dataset  $T$ 
     $\mathbf{nn}_T \leftarrow \{nn_1, \dots, nn_k\} : \forall i < j \text{ distance}(\mathbf{x}_T, \mathbf{x}_{nn_i}) \leq \text{distance}(\mathbf{x}_T, \mathbf{x}_{nn_j})$ 
    // Recommend ranking for new dataset  $T$  based on performance
    information from its nearest neighbors
     $\mathbf{R} \leftarrow \text{aggregate}(\mathbf{p}_{nn_1}, \dots, \mathbf{p}_{nn_k})$ 
end

```

Algorithm 2.1: The k -NN ranking algorithm for algorithm recommendation

selected in the way described in the previous section are the ones that are most similar to the test example in terms of the metafeatures (and the distance measure used). Therefore, assuming an adequate choice of metafeatures and distance measure, the target of the test example is expected to be similar to the targets of the k examples. However, the set of k targets selected may differ, and, thus, some policy to handle conflicts needs to be devised. Similar situations occur in other supervised learning problems. For instance, in a classification problem, the k examples may belong to several different classes. This can be dealt with, for instance, by predicting the most frequent class in the selected examples.

Recall that the target feature in this case consists of a ranking of the base-algorithms. A simple approach is to aggregate the k target rankings with the *Average Ranks* (AR) method. Let $R_{i,j}$ be the rank of base-algorithm a_j ($j = 1, \dots, n$) on dataset i , where n is the number of algorithms. The *average rank* for each a_j is:

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

The final ranking is obtained by ordering the average ranks and assigning ranks to the algorithms accordingly. An example is given next.

Example of a Predicted Ranking

The use of the k -NN ranking method on the problem of algorithm recommendation is illustrated here with real metadata. The metadata used concern ten

Table 2.1. Classification algorithms

bC5	Boosted decision trees (C5.0)
C5r	Decision tree-based rule set (C5.0)
C5t	Decision tree (C5.0)
IB1	1-nearest neighbor (MLC++)
LD	Linear discriminant
Lt	Decision trees with linear combination of attributes
MLP	Multilayer perceptron (Clementine)
NB	Naïve Bayes
RBFN	Radial basis function network (Clementine)
RIP	Rule sets (RIPPER)

Table 2.2. Example of a ranking predicted with 3-NN for the **letter** dataset, based on datasets **byzantine**, **isolet** and **pendigits**, and the corresponding target ranking

Ranking	bC5	C5r	C5t	MLP	RBFN	LD	Lt	IB1	NB	RIP
byzantine	2	6	7	10	9	5	4	1	3	8
isolet	2	5	7	10	9	1	6	4	3	8
pendigits	2	4	6	7	10	8	3	1	9	5
$\bar{R}_i$	2.0	5.0	6.7	9.0	9.3	4.7	4.3	2.0	5.0	7.0
predicted	1	5	7	9	10	4	3	1	5	8
target	1	3	5	7	10	8	4	2	9	6

classification algorithms (Table 2.1) and 57 datasets from the UCI repository [8]. More information about the experimental setup can be found in [41].

The goal in this example is to predict the ranking of the algorithms on the **letter** dataset with, say, the 3-NN ranking method. First, the three nearest neighbors are identified based on the L_1 distance on the space of metafeatures used. These neighbors are the datasets **byzantine**, **isolet** and **pendigits**. The corresponding target rankings as well as the $\bar{R}_i$ scores and the ranking obtained by aggregating them with the AR method are presented in Table 2.2. This ranking provides guidance concerning the experiments to be carried out. It recommends the execution of **bC5** and **IB1** before all the others, then of **Lt**, and so on. However, it contains two pairs of ties (between **bC5** and **IB1** and between **C5r** and **NB**). A tie means that there is no evidence that two or more algorithms will achieve different performance, based on the metadata used. The user can select which algorithm to run first based on personal preferences, on expected execution time (e.g., the mean execution time of **IB1** on the three neighbors was less than that of **bC5**) or on mean performance across all datasets (e.g., **bC5** achieved better mean accuracy than **IB1**). The choice can also be random, as the algorithms that are tied are expected to achieve similar performance.

The question that follows is whether the predicted (or recommended) ranking is an accurate prediction of the target ranking, i.e., of the relative

performance of the algorithms on the **letter** dataset. The target ranking (last row of Table 2.2) is based on estimates of the true performance of the algorithms, such as those obtained with cross-validation. We observe that the two rankings are more or less similar. The largest error is made in the prediction of the ranks of LD and NB (four positions) but the majority of the errors are of two positions. Nevertheless, a proper evaluation methodology is necessary. That is, we need methods that enable us to quantify and compare the accuracy of rankings in a systematic way.

2.3 Experimental Evaluation

Here we describe two methods to assess the quality of the rankings predicted by metalearning methods, such as the k -NN ranking method described earlier. The first aims to assess the accuracy of the ranking, while the second tries to assess the value of the recommendation in terms of the results obtained on the base-level problem. The methods are illustrated on the problem in Section 2.2, in which the goal is to provide a recommendation concerning which of a set of ten algorithms to use on classification datasets.

2.3.1 Evaluation of Ranking Accuracy

Different predicted rankings have different degrees of accuracy. For instance, given the target ranking $(1, 2, 3, \dots, n-1, n)$, the ordering $(2, 1, 3, \dots, n-1, n)$ is intuitively a better prediction (i.e., it is more accurate) than the ordering $(n, n-1, \dots, 3, 2, 1)$. This is because the former ordering is more similar to the target ranking than the latter one. In this section we will discuss one measure (based on rank correlation) that can be used to assess how similar two rankings are [238, 40, 41]. This will enable us to assess the accuracy of a given ranking method. Ranking method A will be considered *more accurate* than ranking method B if it generates rankings that are more similar to the target ranking than those obtained by ranking method B.

Assessing Ranking Accuracy

We can measure the similarity between predicted and target rankings using Spearman's rank correlation coefficient [239, 181, Ch. 9].³

$$r_S = 1 - \frac{6 \sum_{i=1}^n (\hat{R}_i - R_i)^2}{n^3 - n} \quad (2.2)$$

where $\hat{R}_i$ and R_i are, respectively, the predicted and target ranks of item i and n is the number of items.

³ The formula presented assumes that there are no ties [181].

Table 2.3. Accuracy of a ranking predicted for the **letter** dataset

Ranking	bC5	C5r	C5t	MLP	RBFN	LD	Lt	IB1	NB	RIP
predicted	1.5	5.5	7	9	10	4	3	1.5	5.5	8
target	1	3	5	7	10	8	4	2	9	6
$(\hat{R}_i - R_i)^2$	0.25	6.25	4	4	0	16	1	0.25	12.25	4
r_S	0.709									

An interesting property of Spearman’s coefficient is that it is basically the sum of squared rank errors, which can be related to the normalized mean squared error measure, commonly used in regression [265]. The sum is rescaled to yield more meaningful values: the value of 1 represents perfect agreement, and -1 represents complete disagreement. A correlation of 0 means that the rankings are not related, which would be the expected score of a random ranking method. The statistical significance of the values of r_S can be obtained from the corresponding table of critical values, which can be found in many textbooks on statistics (e.g., [181]).

The use of Spearman’s rank correlation coefficient (Equation 2.2) to evaluate ranking accuracy is illustrated in Table 2.3.⁴ Given that the number of algorithms is $n = 10$, we obtain $r_S = 1 - \frac{6 \cdot 48}{10^3 - 10} = 0.709$. According to the table of critical values of r_S , the value obtained is significant at a significance level of 2.5% (one-sided test).⁵ Therefore, Spearman’s coefficient confirms that this ranking is a good approximation to the target ranking.

A resampling strategy can be used to estimate the performance of a metalearning method in the same way as that of any machine learning algorithm (Figure 2.4). For instance, a leave-one-out procedure could be applied. In this procedure, $m - 1$ datasets are used as training metadata to provide a recommendation for the remaining (test) dataset. The accuracy of the recommendation is measured using Spearman’s correlation as explained earlier. This is repeated m times, each time with a different test dataset. The performance of the method is the mean accuracy of the m recommendations. As an example, Figure 2.5 plots the estimated accuracy of the k -NN ranking method for different values of k , obtained using the leave-one-out procedure just described [41]. We note that in this case the best ranking accuracy is obtained for relatively low numbers of k (1 or 2).

Recommended Ranking Baseline

To determine whether the accuracy of some particular recommended ranking can be regarded as high or not, a baseline method is required. For instance, the accuracy of the example given in Table 2.3, measured with Spearman’s

⁴ Note that although the recommended ranking is the same as the one presented in Table 2.2, ties are handled here as in statistics [181].

⁵ The significance level is (1–confidence level).

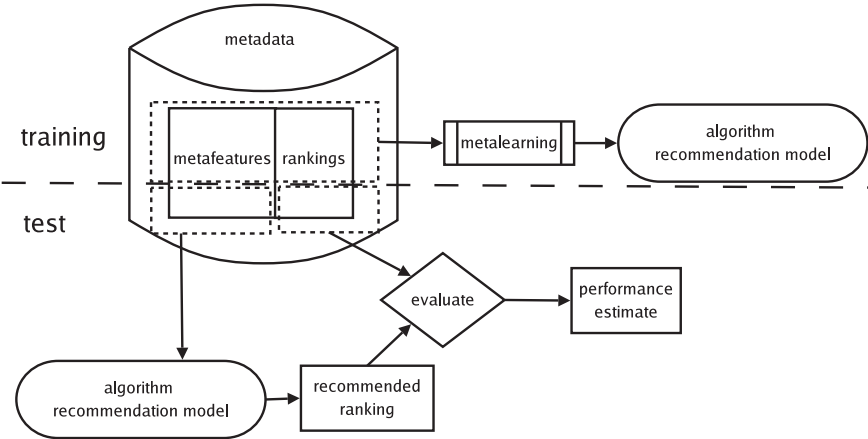


Fig. 2.4. One step of a resampling strategy for the evaluation of a metalearning method for recommendation

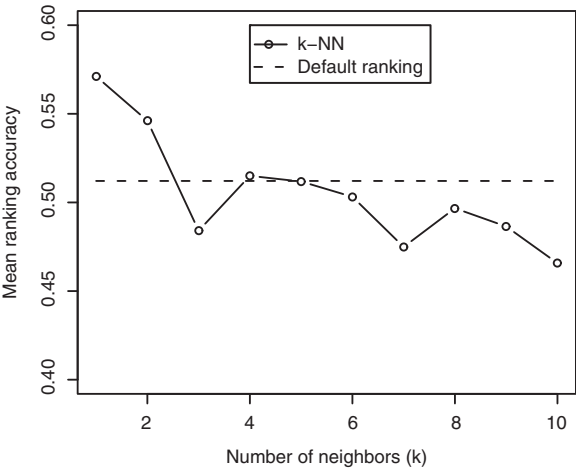


Fig. 2.5. Mean ranking accuracy of the k -NN ranking method and of the default ranking

correlation coefficient, is 0.709. Given that the values of Spearman’s coefficient range from -1 to 1, one may be inclined at first glance to argue that the predicted ranking is highly accurate. However, this may not always hold. There may exist a trivial method that produces comparable (or even better) results.

In machine learning, simple prediction strategies are usually employed to set a baseline for more complex methods. For instance, a baseline commonly used in classification is the most frequent class in the dataset, referred to as the default class. In regression, the mean and the median of the target

Table 2.4. Accuracy of the default ranking on the **letter** dataset

Ranking	bC5	C5r	C5t	MLP	RBFN	LD	Lt	IB1	NB	RIP
default	1	2	4	7	10	8	3	6	9	5
target	1	3	5	7	10	8	4	2	9	6
$(\hat{R}_i - R_i)^2$	0	1	1	0	0	0	1	16	0	1
r_S	0.879									

values are commonly used as baselines. In both cases, the baseline is obtained by summarizing the values of the target variable for all the examples in the dataset. In ranking, a similar approach consists of applying the Average Ranks (AR) method described earlier to all the target rankings in the metadata.

The ranking obtained is called the *default ranking*. The default ranking for the experimental setup considered here is presented in Table 2.4. The accuracy of this ranking on the **letter** dataset is 0.879. This means that, although the ranking generated with the 3-NN is quite accurate ($r_S = 0.709$), it is actually not as accurate as the default ranking on this particular dataset.

Default
ranking

The results in Figure 2.5 show that the default ranking also obtains a high mean accuracy across all datasets, $\overline{r_S}$. However the k -NN obtains results that are clearly better than the baseline for small k .

Statistical Comparison of Ranking Methods

The availability of methodologies for the empirical comparison of methods is as important in the context of ranking methods as in other learning tasks. For instance, it may be of interest to compare the k -NN ranking method with another ranking method and also with the baseline default ranking.

The need for those methodologies is motivated by the fact that showing that a ranking method generates more accurate rankings than the baseline method on average is not sufficient. The values used in the comparison are *estimates* of the corresponding true ranking accuracies, obtained using a sample of datasets. These estimates, like the estimates of the accuracy of algorithms in other learning tasks, have a certain variance, which may imply that the differences between two methods are not statistically significant. Therefore, we need a methodology to assess the statistical significance of the differences between ranking methods. A combination of Friedman's test and Dunn's multiple comparison procedure can be used for this purpose [41].

2.3.2 Top-N Evaluation

So far the quality of a ranking method has been assessed by measuring the *accuracy* of the prediction, represented by the similarity between the recommended and the target ranking. However, the accuracy of a given ranking does not contain information about its *value*, i.e., the final outcome if the

recommended ranking is followed. Ultimately, the user of the metalearning system is interested in the quality of the base-level model obtained using the recommendation given by the predicted ranking. As an example, knowing the accuracy of a ranking predicted by the k -NN method does not provide any information about the classification accuracy that can be obtained if, say, the top two algorithms in the ranking are tried out. The top-N evaluation method described here can be used for that purpose.

Assessing the Value of a Ranking

Here it is assumed that, given a ranking, the choice of which items to select is a compromise between costs and benefits. For instance, given a ranking of learning algorithms and a dataset, it is possible to increase the chance of finding the truly best one by increasing the number of algorithms tried out. The user would normally choose the best algorithms from the subset he or she has examined. However, if more algorithms are executed, the computational costs also increase. Thus, the value of a subset of items (algorithms) is a function of the total benefit and the total cost of applying them to the given dataset. In the algorithm recommendation setting the benefit is represented by the maximum accuracy achieved and the cost is represented by the total computational resources required (such as execution time).

When a recommendation is provided in the form of a ranking, it is reasonable to expect that the order recommended will be followed. The item ranked at the top is expected to be considered first, followed by the one ranked second, and so on. However it is difficult to guess how many items a particular user will select. A *top-N* evaluation method can be used for this purpose [41]. This method consists of simulating that the top N items will be selected, while varying the value of N . Algorithm 2.2 summarizes how the method can be applied to the problem of algorithm recommendation.

The method is illustrated by evaluating the recommended ranking presented in Table 2.5 for the **waveform40** dataset. The table also presents the accuracy obtained by each algorithm and the corresponding execution time. The results of carrying out top-N evaluation based on this information is given in Figure 2.6. In the left plot, computational cost is measured simply by counting the number of algorithms executed (N). The first algorithm recommended for this dataset is the Multi-Layer Perceptron (MLP), obtaining an accuracy of 81.4%. If the user also executes the next algorithm in the ranking, Radial Basis Function Network (RBFN), a significant increase in accuracy is obtained, up to 85.1%. The execution of the next algorithm in the ranking, Linear Discriminant (LD), yields a gain of less than 1% (86.0%). The remaining algorithms obtain lower accuracies.

Alternatively, in the right plot, costs are represented as execution time. This plot provides further information that is relevant for the assessment of a ranking. It shows that although the execution of RBFN provides a significant improvement in accuracy, it does so at the cost of a comparatively much larger

```

input :  $\mathbf{R} = \{r_1, \dots, r_n\}$  // The recommended ranking for the test
dataset  $T$ , where  $r_j = i$  means that algorithm  $a_i$  is ranked in
position  $j$  and  $n$  is the number of algorithms

       $\mathbf{g}_T$ 
       $\mathbf{c}_T$  // Estimates of performance of the base-algorithms  $\mathbf{A}$  on
dataset  $T$ , where  $g_i$  and  $c_i$  represent the estimates of
generalization performance (e.g., classification accuracy) and
computational cost (e.g., execution time) of algorithm  $a_i$ ,
respectively
output:  $\mathbf{tg}_T$ 
       $\mathbf{tc}_T$  // Estimates of Top- $N$  performance of recommended
ranking  $\mathbf{R}$  on dataset  $T$ , where  $tg_i$  and  $tc_i$  represent the
estimates of generalization performance and computational cost
(total time) of executing the top  $i$  algorithms in the ranking,
respectively
begin
   $\mathbf{tg}_1 \leftarrow g_{r_1}$ 
   $\mathbf{tc}_1 \leftarrow c_{r_1}$ 
  foreach  $i \in \{2, \dots, n\}$  do
    // Determine the value of executing the algorithm ranked  $i^{th}$ 
    after executing the algorithms ranked higher
     $\mathbf{tg}_i \leftarrow \max(\mathbf{tg}_{i-1}, g_{r_i})$ 
     $\mathbf{tc}_i \leftarrow \mathbf{tc}_{i-1} + c_{r_i}$ 
  end
end
Note: In case of ties, select the alternative with the lowest mean error in all
training data.

```

Algorithm 2.2: Top-N evaluation

Table 2.5. Ranking recommended for the **waveform40**, the accuracy (%) obtained by the algorithms and their execution times (in seconds)

Rank	1	2	3	4	5	6	7	8	9	10
Algorithm	MLP	RBFN	LD	Lt	bC5	NB	RIP	C5r	C5t	IB1
Accuracy	0.81	0.85	0.86	0.84	0.82	0.80	0.79	0.78	0.76	0.70
Time	99.70	441.52	1.73	9.78	44.91	3.55	66.18	11.44	4.05	34.91

execution time. It takes approximately 450 sec. while MLP took 100 sec. The plot also shows that although the gain obtained with LD is smaller, it is quite fast (less than 2 sec. to execute).

This example clearly illustrates the need for an evaluation of rankings by taking both their benefits and their costs into account.

Assessing the Value of a Ranking Method

In the previous section, a recommended ranking for a single dataset was evaluated using the top-N method. However, research and development of algorithm

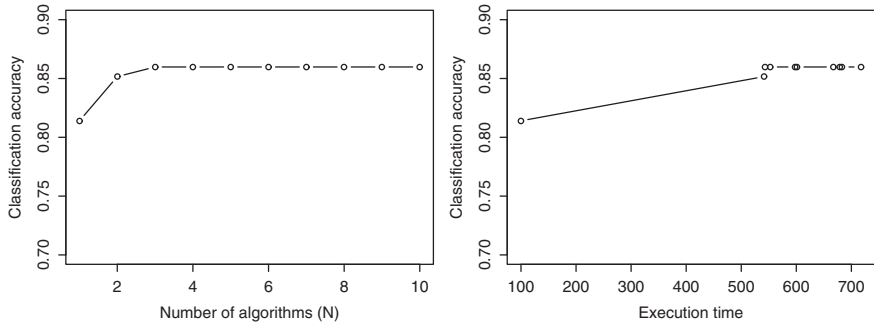


Fig. 2.6. Top-N evaluation of the recommendation obtained with 1-NN for the waveform40 dataset

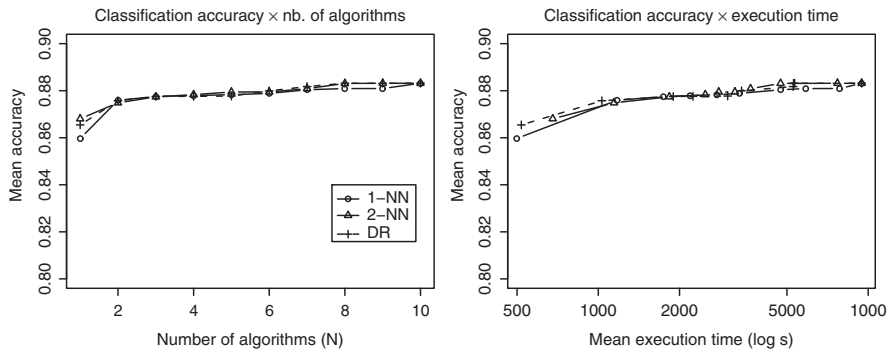


Fig. 2.7. Mean top-N performance across all datasets of 1-NN, 2-NN and the default ranking (DR)

recommendation methods requires that their performance be compared across several datasets. For that purpose it is necessary to aggregate the top-N curves obtained for several datasets.

Top-N performance of a ranking method on several datasets can be assessed simply by averaging the values of benefit (accuracy) and cost (number of algorithms executed or execution time) across all datasets for each value of N . In Figure 2.7 we illustrate this approach by presenting the mean top-N results obtained by 1-NN, 2-NN and the default ranking on 57 datasets. Contrary to the results obtained with ranking accuracy, which clearly indicates an advantage of the k -NN ranking method in comparison to the default ranking, the curves we observe are generally very similar. These results illustrate the need to complement the evaluation based on ranking accuracy with another method which takes into account the value of the recommendation.

An additional observation can be made with regard to Figure 2.7. The difference between the mean accuracy of the top-1 algorithm recommended by the 2-NN ranking method and the brute force strategy of executing all

algorithms is only 1.5%. However, this difference is reduced to 0.8% if the second algorithm is executed (i.e., if the user follows a top-2 strategy for algorithm selection). This gain in accuracy is obtained at the cost of an admissible increase in computational cost (from approximately 11 to 20 minutes, while the strategy of executing all algorithms takes more than 2.5 hours, on average). These results indicate that a top-1 strategy is not competitive in comparison to top-2 and, thus, discourages the use of the former. This would not be possible if metalearning were addressed as a classification problem, which would recommend a single algorithm for each dataset. This clearly demonstrates the advantage of using ranking methods in the problem of algorithm recommendation.

The method described so far enables the comparison and choice of a metalearning method based on mean performance. However, there may be situations where it is also important to assess the risk of using a metalearning method. For instance, when selecting algorithms for critical applications (e.g., medical applications in which lives are at stake), we must ensure that the recommendations provided by the method will yield satisfactory performance on all the datasets. In those situations an algorithm recommendation method that generally provides good recommendations and never provides very bad ones may be preferred over a system that finds the best algorithm often but makes a few very bad recommendations. Therefore, it is important to make a *worst-case analysis* of the top- N performance of metalearning methods. The experimental setup considered here can be used to illustrate this issue. As shown in Table 2.6, the mean accuracy of **bc5** is less than 3% smaller than the accuracy obtained with the best algorithm for each dataset. However, this difference is much higher in some datasets, achieving a maximum of 34%.

Therefore, given the set of top- N results of a metalearning method on a group of datasets, its worst-case top- N performance can be assessed by determining the dataset for which the lowest accuracy was obtained, for each value of N . Given that the minimum value is very sensitive to outliers, we can obtain a more robust estimate of worst-case performance using the 1st quartile function [233].⁶ The worst-case top- N results for 1-NN, 2-NN and the default ranking are plotted in Figure 2.8. Although they confirm in general mean

Table 2.6. The largest differences and the mean difference between the accuracy obtained by using the best algorithm for each dataset and always using **bc5**

Dataset	best - bc5 (%)
task1	34.0
krkopt	11.2
internetad	11.2
<i>all datasets</i>	2.3

⁶ Using the 1st quartile rather than the minimum accuracy could be named “bad-case” rather than “worst-case” analysis.

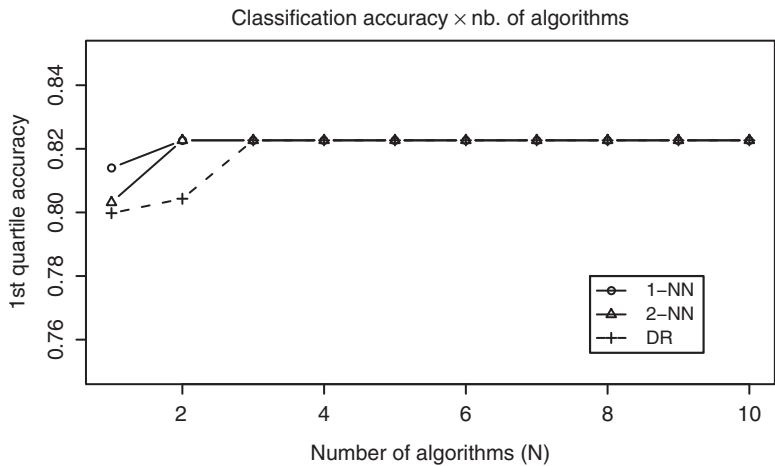


Fig. 2.8. Worst-case analysis of top-N performance of the 1-NN, 2-NN and the default ranking (DR)

performance results (Figure 2.7), that the three methods perform similarly, they illustrate the usefulness of this analysis nicely. The curve representing 1-NN, which was the lowest in Figure 2.7, is now clearly above the others, for $N = 1$ and $N = 2$. According to the figure, when worst-case performance is relevant, 1-NN should probably be the selected metalearning method.

Alternatively, we may also be concerned with worst-case performance in terms of computational cost. For instance, the available time for executing an algorithm may be bounded. In this case, it is necessary to analyze the worst-case performance of a metalearning method in terms of maximum execution time. This would be done in a way similar to that described for minimum accuracy.

2.4 Recommendation of Algorithm Parameters Using Metalearning

For the sake of simplicity, it has been assumed so far that the algorithms have no parameters. In practice, this is equivalent to saying that the algorithms considered have been used with the set of parameter values selected by default in the corresponding implementation. Given that it is well-known that the choice of parameters affects the quality of the results obtained, in some cases dramatically, metalearning methods for selection of algorithms should also provide recommendations concerning the values of the parameters.

Here, the focus is on the problem of recommending parameters for the Support Vector Machine (SVM) algorithm [236]. SVM is a kernel-based algorithm which combines sound and elegant theoretical foundations with good empirical results over a wide range of applications. The algorithm is not explained

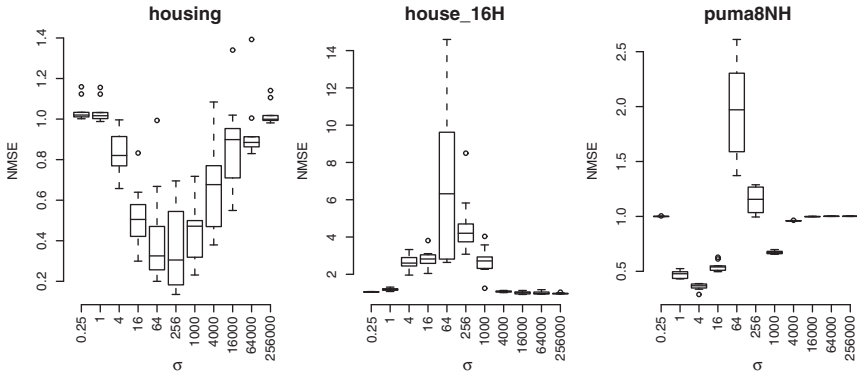


Fig. 2.9. Distribution of the error (NMSE) obtained by SVM with Gaussian kernel on three regression problems with 11 different settings of the kernel width (σ) parameter

here as several papers and books provide suitable explanations with varying degrees of complexity (e.g., [18, 48, 178, 70]), as well as pointers to several applications.

One of the most important issues concerning the use of SVM is the choice of kernel function [48], which determines the hypothesis space. Given different kernel functions, it is possible to induce very different models, such as linear models (linear kernel), radial basis functions (Gaussian kernel) and two-layer sigmoidal neural networks (sigmoidal kernel). The choice is obviously important to achieve good results. Furthermore, most kernel functions have specific parameters which also affect the performance of the algorithm. Figure 2.9 illustrates the significant differences in the errors that can be obtained using different values for the parameters of a given kernel. The figure shows that the best value also varies significantly across different problems.

2.4.1 Methods to Set Parameters of SVMs

There are three approaches to set the parameters of SVMs, namely estimation of the generalization error, optimization and the use of heuristics. The estimation of the generalization error is based on the empirical error. Three common ways of obtaining the empirical error are cross-validation (CV), the bayesian evidence framework and the PAC framework [178]. These approaches have the disadvantage of requiring that the SVM model be induced for every setting considered. Given that the computational requirements of SVMs are significant both in the training and in the test phases [48], this can be computationally very demanding, especially when dealing with large datasets.

Optimization approaches exist to set not only a single parameter [71] but also multiple parameters simultaneously [60]. Again, these approaches can be computationally very expensive because the SVM algorithm must be executed for each value selected by the optimization method.

To avoid this problem, the choices concerning parameter settings are often driven by heuristics. For instance, the Gaussian kernel is generally a good choice when only the smoothness of the data can be assumed [231]. A common heuristic to set the width of the Gaussian kernel is based on the distances between the examples in the attribute space [130].

2.4.2 k -NN Ranking Method for Parameter Setting Recommendation

The metalearning method described earlier can be directly applied when the number of alternatives is finite, such as in the case of choosing the value for a parameter such as the kernel function of SVM. However, many parameters have an infinite number of values, particularly continuous parameters, such as σ , the width of the Gaussian kernel. In this case, the parameter must be discretized to yield a set of finite alternatives which may be ranked. The selection of an appropriate subset of alternatives is discussed in more detail in Section 3.4.1.

An application of this approach is presented for illustration purposes. The problem addressed is the recommendation of the width of the Gaussian kernel of SVMs for regression problems [236]. A set of 11 values of the parameter are considered, approximately following a geometric progression starting from 0.25 and with factor 4: 0.25, 1, $\dots$, 256, 1 000, 4 000, $\dots$, 256 000. The performance of the algorithm is assessed using the *normalized mean squared error*:

$$NMSE = \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2}$$

where n is the number of cases, y_i and $\hat{y}_i$ are the target and the predicted values for case i , and $\bar{y}$ is the mean of the target values. The values of NMSE range from 0 to ∞ , with 1 representing the error of a baseline strategy of predicting the mean target value. Values larger than 1 mean that the algorithm performs worse than this baseline strategy, which is not very common.

Figure 2.10 presents results obtained on 42 datasets [236], in terms of ranking accuracy (Section 2.3.1) and top-N performance (Section 2.3.2). These results show that metalearning can also be successfully used to recommend parameter settings for SVM.

2.5 Discussion

This chapter addresses one of the applications which metalearning can be used for, namely the recommendation of algorithms and parameter settings.

A simple solution is described for illustration purposes. It is clear from the description that the development of a metalearning system for algorithm

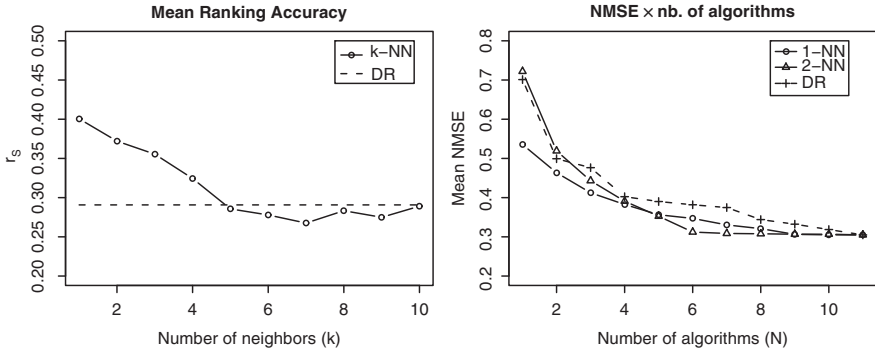


Fig. 2.10. Performance of k -NN ranking method on the problem recommending the width of the Gaussian kernel of SVM for regression: ranking accuracy (left) and top- N (right)

recommendation involves many decisions. These include the choice of base-level algorithms, of the form of recommendation and of the metafeatures. These decisions will be discussed in the following chapter.

Development of Metalearning Systems for Algorithm Recommendation

3.1 Introduction

In the previous chapter, a metalearning approach to support the selection of learning algorithms was described. The approach was illustrated with a simple method that provides a recommendation concerning which algorithm to use on a given learning problem. The method predicts the relative performance of algorithms on a dataset based on their performance on datasets that were previously processed.

The development of metalearning systems for algorithm recommendation involves addressing several issues not only at the *meta level* (lower part of Figure 3.1) but also at the *base level* (top part of Figure 3.1). At the meta level, it is necessary, first of all, to choose the type of the target feature (or *metatarget*, for short), that is, the form of the recommendation that is provided to the user. In the system presented in the previous chapter, the form of recommendation adopted was rankings of base-algorithms. The type of metatarget determines the type of *meta-algorithm*, that is, the metalearning methods that can be used. This in turn determines the type of *metaknowledge* that can be obtained. The meta-algorithm described in the previous chapter was an adaptation of the *k*-nearest neighbors (*k*-NN) algorithm for ranking. The metatarget and the meta-algorithm are discussed in more detail in Section 3.2.

To perform metalearning it is necessary to build an adequate *metadatabase*. Firstly, it is necessary to gather *meta-examples* which are datasets or, more generally, learning problems. One source of (classification) learning problems is repositories, such as the UCI repository [8]. The goal of the process is to obtain metaknowledge that relates *properties* of those datasets to the relative performance of algorithms. Therefore, it is necessary to define which properties are important to characterize those datasets and to develop *metafeatures* that represent those properties. For instance, one metafeature for classification datasets is the number of classes. In Section 3.3 we discuss these issues in more detail.

Metatarget

Meta-
algorithmMeta-
knowledgeMeta-
database
Meta-
examplesMeta-
features

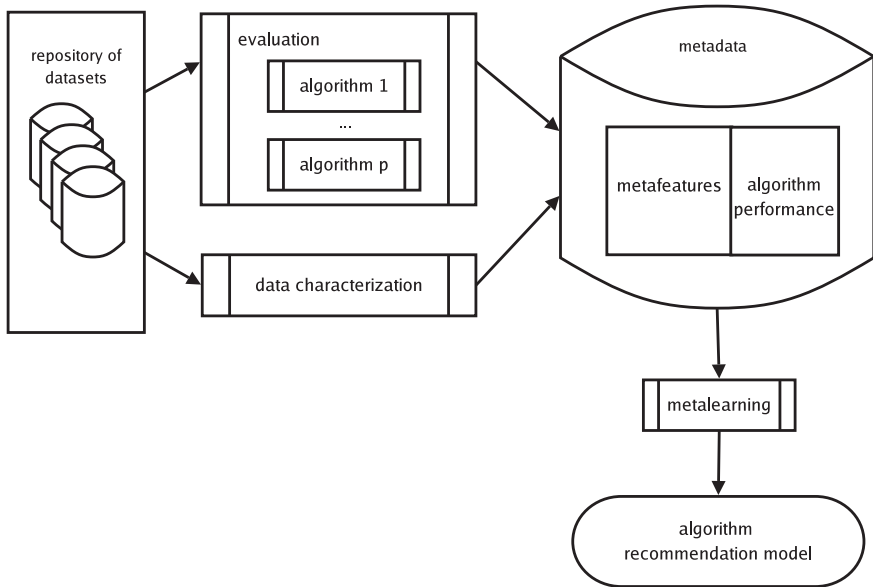


Fig. 3.1. Metalearning to obtain metaknowledge for algorithm selection

Besides storing information concerning the properties of datasets, the metadatabase must also store information about the performance of the *base-algorithms* on the selected datasets. The first step is to select the base-algorithms, that is, the set of algorithms¹ that the recommendation of the system will be based on. Additionally, the measure(s) that will be used to evaluate the performance of the algorithms must be identified. Different measures may be suitable for different applications (e.g., classification accuracy or area under the ROC curve). These *base-learning* issues are discussed in Section 3.4.

Data quality is as important in metalearning as it is in any machine learning task. Common problems, such as missing values or noise, may occur in metadata as in any dataset and affect the quality of the recommendations generated by the metalearning system. A few issues regarding metadata quality are discussed in Section 3.5.

The goal of this chapter is to develop a deeper understanding of these issues and to provide an overview of the state-of-the-art approaches to address them. Most approaches described here are independent of the base-learning task addressed (e.g., classification or regression). We essentially focus on the recommendation of classification algorithms because it has been more extensively researched than other learning tasks. However, other tasks, such as regression and time series forecasting, are discussed where appropriate to

¹ Here, as in the previous chapter, we will use the term *algorithms* to represent both different algorithms and different parameter settings of a single algorithm.

illustrate how the type of base-learning tasks affects the development of the metalearning system.

3.2 Meta-level Learning

The first decision that must be made concerning the development of the meta-level learning part of the algorithm recommendation system (lower part of Figure 3.1) is about the form of the recommendation that is to be provided to the user, i.e., the type of the metatarget. Existing possibilities are discussed in Section 3.2.1.

After choosing the type of the metatarget it is necessary to choose the algorithm for metalearning, i.e., the meta-algorithm. In Section 3.2.2, we discuss meta-algorithms that have been previously used for algorithm recommendation.

3.2.1 Target Metafeature

The form of recommendation provided by the metalearning system should be selected taking into account the desired usage. In some cases, the user may simply be interested in knowing which algorithm is the best. In other cases, more detailed information about the performance of the set of base-algorithms may be required. The form of recommendation determines the type of target metafeature, or *metatarget*, to learn. In the following sections, four different types of metatargets are discussed: best algorithm in a set, subset of algorithms, ranking of algorithms and estimated performance of algorithms. For illustration purposes, we will consider imaginary sets of p algorithms, $\{a_1, a_2, \dots, a_p\}$, and q datasets, $\{d_1, d_2, \dots, d_q\}$.

Metatarget

Best Algorithm in a Set

The first form consists of recommending the algorithm that is expected to obtain the best performance in the set of base-algorithms [190, 134]. For each dataset, d_i , the recommendation will consist of a single base-algorithm, a_j (row 1 in Table 3.1).

Best
algorithm
in a set

An advantage of this form is that the metalearning problem becomes a classification task, and so the development of recommendation systems can benefit from the vast amount of research on this task. A very important disadvantage is when the recommended algorithm fails. If this happens, the user is left on his or her own, without any information on which algorithm to try next. Besides, there is no guarantee that the algorithm recommended is truly the best one.

In the particular case of predicting the value of a numerical parameter of a base-algorithm (e.g., the width of the kernel of SVM), an alternative

Table 3.1. Examples of different forms of recommendation

1. Best in a set	a_3					
2. Subset	$\{a_3, a_1, a_5\}$					
	Rank					
	1	2	3	4	5	6
3. Ranking (linear and complete)	a_3	a_1	a_5	a_6	a_4	a_2
4. Ranking (weak and complete)	$\overline{a_3}$	$\overline{a_1}$	a_5	$\overline{a_6}$	$\overline{a_4}$	a_2
5. Ranking (linear and incomplete)	a_3	a_1	a_5	a_6		
	Algorithms					
	a_1	a_2	a_3	a_4	a_5	a_6
6. Estimates of performance	0.89	0.68	0.90	0.74	0.81	0.75

approach is possible. Given that the metatarget is actually a numerical feature, it is possible to address this as a regression task [156]. If several numerical parameters exist, i.e., if a_i represents a set $\{par_1, par_2, \dots\}$, where par_j is the value of parameter j , then it is possible to combine several regression models, one for each parameter, par_j .

Subset of Algorithms

Methods that use the second form of recommendation suggest a (usually small) subset of algorithms that are expected to perform well on the given problem [263, 139, 134]. For a given dataset, the recommendation is $\{a_i\} \subset \{a_1, a_2, \dots, a_p\}$ (row 2 in Table 3.1).

The notion of “performing well” on a given problem is typically defined in relative terms. One approach is to establish a margin relative to the performance of the best algorithm on that problem. All the algorithms with a performance within the margin are considered to perform well. In classification, the margin can be defined in the following way [38, 106, 263]:

$$\left[e_{min}, e_{min} + k \sqrt{\frac{e_{min} (1 - e_{min})}{n}} \right) \quad (3.1)$$

where e_{min} is the error of the best algorithm, n is the number of examples and k is a user-defined parameter determining the size of the margin.

An alternative approach is to carry out statistical tests to compare the significance of the difference in performance between algorithms [139, 134]. An algorithm is considered to perform well if it is not significantly worse than the best one. Both approaches are related because the margin used in the

former can be regarded as an interval of confidence of the performance of the best algorithm. Thus, any algorithm with a performance above the threshold can be considered to be not significantly worse than the best one.

This form of recommendation has the advantage that the user is provided with more than one algorithm to try out, unlike in the previous case. However, a recommendation is provided as an unordered subset of the original set of alternatives. Therefore, no guidance is provided concerning which of the algorithms to try first, which second, and so on.

Ranking of Algorithms

Rankings

The lack of order in the subset approach can be remedied if a ranking of the algorithms is provided [38, 234, 141, 41]. Typically, the order indicated in the ranking is the order that should be followed in the experimentation process. Several types of rankings are shown in Table 3.1 (rows 3–5).

The first type of ranking is a *linear ranking* because the ranks are different for all algorithms. Additionally, it is a *complete ranking* because all the algorithms $a_1, \dots, a_p$ have their rank defined [67]. This type of ranking may not, however, be suitable for all algorithm recommendation applications. Firstly, a linear ranking cannot represent the case when the metalearning model predicts that two algorithms will be tied on a given problem (i.e., their performance is not significantly different). In such cases, a *weak* ranking may be used.² In the weak ranking of Table 3.1 (row 4), the line above two or more algorithms (as in $\overline{a_3 a_1}$) indicates that the performances of the corresponding algorithms are not significantly different. When the metalearning method is unable to provide a recommendation concerning one or more of the base-algorithms, complete rankings cannot be derived. This may happen if there is not enough data concerning the performance of those algorithms to predict their (relative) performance on the dataset at hand (e.g., their execution has failed on relevant datasets or the corresponding experiments have not been carried out yet). In this case, it may be better not to include these algorithms in the recommendation, thus yielding an *incomplete ranking* (row 5 in Table 3.1).³

Hasse diagrams provide a simple visual representation of rankings [187], where each node represents an algorithm and directed edges represent the relation “significantly better than”. Figure 3.2 (a) shows a linear and complete ranking, Figure 3.2 (b) a weak and complete ranking, and Figure 3.2 (c) a linear and incomplete ranking, each corresponding to the rankings in rows 3 through 5 of Table 3.1).

A metalearning method that provides recommendations in the form of weak rankings is proposed in [42]. The method is an adaptation of the k -NN

² Linear and weak rankings can also be referred to as simple and quasi-linear rankings, respectively [68].

³ Although complete and incomplete rankings can also be named *total* and *partial rankings*, we prefer to use the former terminology because total and partial orders are reflexive, which is not the case with the “significantly better than” relation.

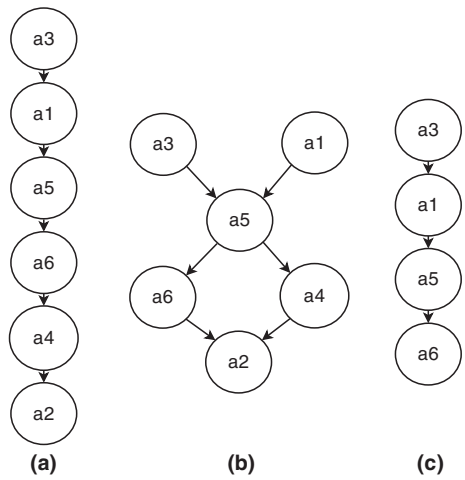


Fig. 3.2. Representation of rankings using Hasse diagrams: a) linear and complete ranking; b) weak and complete ranking; c) linear and incomplete ranking

ranking approach described in the previous chapter that identifies algorithms which are expected to tie, and provides *reduced rankings* by actually including only one of them in the recommendation.

Rankings are particularly suitable for algorithm recommendation because the metalearning system can be developed without any information about how many base-algorithms the user will try out. This number depends on the available computational resources and the importance that algorithm performance has on the problem at hand, so it is expected to vary in different situations. If time is the critical factor, only one or very few alternatives are selected. On the other hand, if the critical factor is accuracy, then the more the number of algorithms tried out, the higher the probability that a good result is obtained. Existing experimental results provide evidence in favor of this argument (e.g., [41]). On the other hand, there is significantly less work on methods for learning rankings than for other tasks (Section 3.2.2). Additionally, a ranking provides no information concerning what performance can be expected and how many alternatives the user should try out.

Estimates of Performance

If one is interested in actual performance rather than simply relative performance as offered by the ranking approach, the metalearning system should provide recommendations in the form of a value indicating the performance that each algorithm is expected to achieve (row “Estimates of performance” in Table 3.1). This approach can transform the problem of algorithm recommendation into several regression problems, one for each base-algorithm [106, 238, 151, 21].

Estimation
of perfor-
mance

The metatarget in this kind of approach may be the performance of the algorithm. However, given that the range of performance values for different datasets may vary substantially (e.g., an accuracy of 90% may be quite high on a classification problem but trivial on another one), it is important to rescale the values. Three methods have been proposed [106]: the distance to the performance of the best algorithm; the distance to some baseline performance measure (e.g., default accuracy in classification); and normalization of the performance.⁴

A different approach to estimate the performance of an algorithm is based on the use of metalearning to predict the performance of a base-level algorithm on each individual base-level example [268]. For instance, in a classification problem, the metalearning model predicts whether the base-level algorithm correctly guesses the class for each test example. A prediction of the performance of the algorithm on the dataset is obtained by aggregating the set of individual predictions that are obtained with the metalearning model. In the classification example, predicted performance of the algorithm could be given by the predicted accuracy, i.e., the proportion of test examples that the metalearning model predicts the base-level algorithm to guess correctly.

By providing estimates for each algorithm, rather than an aggregated recommendation, such as a ranking, we provide more information to the user. This information can be used to decide how many algorithms he or she is going to try. As in classification, metalearning in this case can benefit from a large body of work on regression.

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

Finally, besides their making it possible to provide the estimates directly to the user, these estimates can be transformed to obtain the three other forms of recommendation we have described: best algorithm, i.e., the one that is expected to obtain the best performance [106]; subset of algorithms, containing the ones expected to perform well; and ranking, by ordering the algorithms according to their expected performance [238, 21].

On the other hand, it can be expected that predicting several numerical values is much harder than discriminating between a finite number of classes or predicting rankings. Additionally, the fact that several regression problems are solved independently can be regarded as a disadvantage. In fact, the error in itself is not so important as the question of whether it affects the relative order of the algorithms. For instance, if the estimate of performance of a_3 were 0.92 rather than 0.90 (Table 3.1), then the order of the algorithms would

⁴ Subtraction from the performance value of the algorithm of the mean performance on that dataset and division by the standard deviation.

remain the same. On the other hand, an error of the same magnitude (0.02), but with a negative sign, would cause a_3 to move from first to second position because the new estimated performance of a_3 (0.88) would be lower than that of a_1 (0.89).

As mentioned earlier, a set of estimates concerning the performance of the base-algorithms can be transformed into the other forms of recommendation described in Table 3.1. For instance, the best algorithm in a set can be predicted by selecting the algorithm which is estimated to perform best. However, given that these forms of recommendation are associated with learning tasks that are not regression (e.g., predicting the best in a set of algorithms is a classification task), the regression algorithm that generates the most accurate estimates may not be the one that generates the estimates which are transformed, say, into the most accurate prediction of the best algorithm. An experimental study provides some evidence to support this claim [151]. Therefore, if the goal is to generate recommendations in a form other than that of estimates of performance, then the problem should be addressed as the appropriate task. For instance, if the goal is to predict which algorithm is best, then a classification algorithm should be used. Finally, it could be argued that in many cases the user simply requires some guidance concerning which base-algorithms to execute and in which order, and not really more detailed information on their expected performance.

Little work has been dedicated to comparing empirically the different forms of recommendation discussed here. One study reports that the best results were obtained using regression [151]. However, the authors state that the results were obtained on artificial data and provide some evidence that these conclusions may not be valid in real problems. Another study provides evidence that, somewhat surprisingly, better rankings can be obtained by combining estimates of algorithm performance than by using a ranking algorithm [21]. However, a thorough comparison of the several forms of recommendation has yet to be carried out.

3.2.2 Algorithm for Meta-level Learning

Here we discuss meta-algorithms used in existing metalearning approaches to the algorithm recommendation problem. The choice of meta-algorithms is constrained by the type of metatarget used, as discussed in the previous section. In the cases where it is possible to use classification or regression algorithms, many alternatives are available. Although the choice of ranking algorithms is not so wide, there is growing interest in the area.

Independently of the type of metatarget selected, most metalearning approaches use propositional learning algorithms. However, as will be discussed later (Section 3.3.1), dataset descriptions may be nonpropositional. A few existing approaches that use this information are also discussed below.

Classification Algorithms

Given their wide availability, many different classification algorithms have been tried for meta-level learning. An extreme example is to use at the meta level the same set of algorithms that is considered at the base level [190, 20]. The ten classification algorithms used in these studies are quite diverse, including decision trees, a linear discriminant and neural networks, among others. The authors compare the algorithms on several metalearning problems by analyzing pairs of algorithms and on the problem of choosing the best algorithm from the set, based on results on artificial and UCI datasets. The comparison was not conclusive in one of the studies [20], while the other [190] generally showed that decision tree and rule-based models obtain the best metalearning results.

Compatible results were obtained on a study addressing the problem of selecting a set of algorithms [134, 136]. The authors compare four decision-tree-based algorithms and an IBL with the best results obtained by boosted C5.0 on the meta level.

Regression Algorithms

Not many algorithm recommendation studies carried out so far have used regression rather than classification algorithms at the meta level. The set of regression algorithms considered is smaller and less diverse. In one earlier work, linear regression, regression trees, model trees and IBL were analyzed on the problem of estimating the error of a large number of algorithms [106]. The results indicated that the methods obtain similar performance. More recently, a comparison between Cubist (a regression-tree-based rule system) and a kernel method was also carried out on the problem of estimating the error of ten classification algorithms [21]. Results reported showed a slight advantage for the kernel method.

These approaches generate as many metamodels as there are algorithms. It is, thus, not trivial to understand when an algorithm performs better than another one and vice versa. Take, for instance, the rules presented in Table 3.2,

Table 3.2. Four sample rules that predict the error of C4.5 and CN2 [106]. The metafeatures are: *fract1*, the first normalized eigenvalues of canonical discriminant matrix; *cost*, a boolean value indicating if errors have different costs; and *Ha*, the entropy of attributes

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

that were selected from models that predict the error of C4.5 and CN2 [106]. These models do not describe directly the conditions when C4.5 is better than CN2 and vice versa.

Multitarget
prediction

Clustering trees can be used to induce a single model for multitarget prediction[27].⁵ They are obtained with a common algorithm for top-down induction of decision trees (TDIDT) that tries to minimize the variance of the target variables for the cases in every leaf (and maximize the variance across different leaves). They have been applied to the problem of estimating the performance of several algorithms [261]. The decision nodes represent tests on the values of metafeatures and the leaf nodes represent sets of performance estimates, one for each algorithm (Figure 3.3). However, these models do not necessarily provide explicit metaknowledge concerning the relative performance of algorithms, as illustrated in Figure 3.3. On one hand, the root node does discriminate between datasets in which a_1 is either the best or the worst of the three algorithms. But the test on the second node discriminates datasets in which the algorithms have performances on a different scale, rather than with a different relative performance. Results obtained with clustering trees are comparable to those obtained with the approach using separate models, with the advantage of improved readability, because the former approach generates a single rather than several models [261].

The only metalearning approach besides boosting that combines several models at the meta level has been proposed using regression models [106]. The results obtained by a linear combination of the meta-level models yields better results than any of these models considered individually.

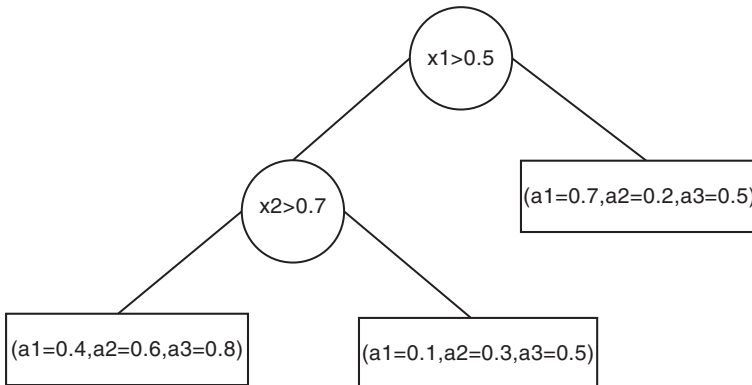


Fig. 3.3. Example of a predictive clustering tree

⁵ In multitarget prediction problems there are several target variables $\mathbf{y}_i$ rather than a single variable $\mathbf{y}$, as is most common in prediction problems such as classification and regression.

Nonpropositional Approaches

The algorithms discussed so far are only able to deal with propositional representations of the metalearning problem. That is, they assume each metaexample is described by a fixed set of metafeatures, $\mathbf{x} = (x_1, x_2, \dots, x_k)$. However, the problem is highly nonpropositional, as will be discussed later (Section 3.3.1). On the one hand, the size of the set of dataset characteristics varies for different datasets (e.g., depending on the number of features). On the other hand, information about the algorithms can also be useful for metalearning (e.g., the interpretability of the generated models). In spite of this, there are very few approaches that use relational learning approaches.

One approach that exploits the nonpropositional description of the datasets uses FOIL, a well-known ILP (Inductive Logic Programming) algorithm [199]. With FOIL, models can be induced that contain existentially quantified rules, such as “CN2 is applicable to datasets which contain a discrete feature with more than 2.3% missing values” [263]. A different approach uses a case-based reasoning tool, CBR-Works Professional [161, 120]. This can be viewed as a k -NN algorithm that allows not only a nonpropositional description of datasets, but also enables the use of information about the algorithms, independently of datasets. This work was recently extended by analyzing different distance measures for nonpropositional representation [138]. Some of these measures enable the distance between two datasets to be defined by a pair of individual features, e.g., the two features which are most similar in terms of one property such as skewness.

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

Ranking Algorithms

Compared to classification or regression, the number of available algorithms to learn rankings is small. Nevertheless, the problem recently started to receive an increasing amount of attention in the machine learning community (e.g., [222, 44]). In metalearning, the most commonly used algorithm is based on k -NN [234, 141, 41, 81]. The choice is essentially motivated by the simplicity of adapting this algorithm for learning rankings, as shown in Chapter 2.

In the k -NN approach to ranking it is necessary to predict the ranking of algorithms for a given problem based on the rankings of its k neighbors. The k rankings may have conflicts (i.e., algorithms with different relative order in different rankings), so some form of aggregation is needed to obtain a recommended ranking. Besides the simple average ranks method presented in Chapter 2, other aggregation methods have been tried, including *Success Rate Ratios* and *Significant Wins* [40]. The first one uses information about the magnitude of the difference in performance between the methods and the second takes into account the significance of the differences in performance.

Ranking
algorithms

Although preliminary results suggested that the average ranks method generates somewhat better rankings, a more thorough study indicates that the observed differences are not significant [233].

A general ranking method that was proposed in the context of metalearning is the ranking trees algorithm [261], based on the clustering trees algorithm mentioned earlier [27]. The adaptation for ranking is obtained by replacing the target values (e.g., the accuracy of the algorithms) by the corresponding positions in the ranking. A comparison of this approach with previously reported results obtained with the k -NN and the regression-based ranking methods [21] indicate that ranking trees obtain the most accurate rankings [261].

Several authors (e.g., [207]) have noted that the choice of metalearning method represents a *metametalearning* problem. In general, we may say that the results of comparative studies of metalearning methods have not lead to conclusive results so far.

3.3 Metadata

Metalearning is based on a database containing information about the performance of a set of algorithms on a set of datasets and about the characteristics of those datasets (Figure 3.1). The characterization of datasets is probably the issue that has attracted the most attention in metalearning research, due to its importance in the process: success is possible only if the metafeatures contain information that is useful for discriminating between the performance of the base-algorithms. In Section 3.3.1, we discuss the issues involved in designing metafeatures.

Additionally, a learning approach to algorithm recommendation cannot be carried out without examples. The gathering of meta-examples is discussed in Section 3.3.2.

3.3.1 Metafeatures

The goal of metalearning is to relate the performance of learning algorithms to data characteristics, i.e., *metafeatures*. Therefore, it is necessary to compute measures from the data that are good predictors of the relative performance of algorithms.

The development of metafeatures for metalearning should take the following issues into account:

Discriminative power. The set of metafeatures should contain information that distinguishes between the base-algorithms in terms of their performance. Therefore they should be carefully selected and represented in an adequate way.

Computational complexity. The metafeatures should not be too computationally complex. If this is not the case, the savings obtained by not executing

all the candidate algorithms may not compensate for the cost of computing the measures used to characterize datasets. Pfahringer et al. [190] argued that the computational complexity of metafeatures should be at most $O(n \log n)$.

Dimensionality. The number of metafeatures should not be too large compared to the amount of available metadata; otherwise overfitting may occur.

Most metalearning approaches focus on characterizing datasets. However, information about the algorithms may also be useful. For example, Hilario and Kalousis [120] use information concerning: *type of representation* (e.g., type of data they are able to deal with), *approach* (e.g., learning strategy, such as lazy or eager), *resilience* (e.g., sensitivity to irrelevant attributes, based on experimental studies), and *practicality* (e.g., easy parameter handling). A combination of metafeatures describing datasets and algorithms is possible due to the usage of a Case-Based Reasoning (CBR) approach, which allows for a nonpropositional description of cases.

Charac-
terization
of
algorithms

General approaches to data characterization are briefly summarized in the next section, while the following discussion considers how information about the specific metalearning problem can be taken into account in the development of metafeatures. Some issues concerning the representation and selection of metafeatures and the process of computing them are discussed in the last two sections.

Types of Metafeatures

Three different approaches to data characterization can be identified, namely simple, statistical and information-theoretic measures, landmarks and model-based measures.

The most common approach to data characterization consists of the use of descriptive statistics or information-theoretic measures to summarize the dataset (top of Figure 3.4). It can be referred to as the *simple, statistical and information-theoretic* approach and it is extensively used in metalearning (e.g., [38, 39, 106, 263, 161, 21, 139, 238, 275, 151, 134]).⁶ Typically, it includes very simple descriptive measures such as the **number of examples** and the **number of features**, which were first used in the earliest metalearning approaches (e.g., [207, 1]) and are still among the most commonly used metafeatures. Most metafeatures are based on measures used in statistics (e.g., **mean skewness of numeric features**) and information theory (e.g., **class entropy**). However, some metafeatures inspired from other fields, such as machine learning itself (e.g., concept variation [275]) and case-based reasoning (e.g., case base quality assessment-based measures [150]), have been proposed.

Simple,
statistical
and
information-
theoretic
metafea-
tures

Some measures focus on a single independent feature (e.g., **skewness** for numerical features or **entropy of features** for symbolic features) or on the

⁶ A thorough review and explanation of this approach is given by Kalousis [134].

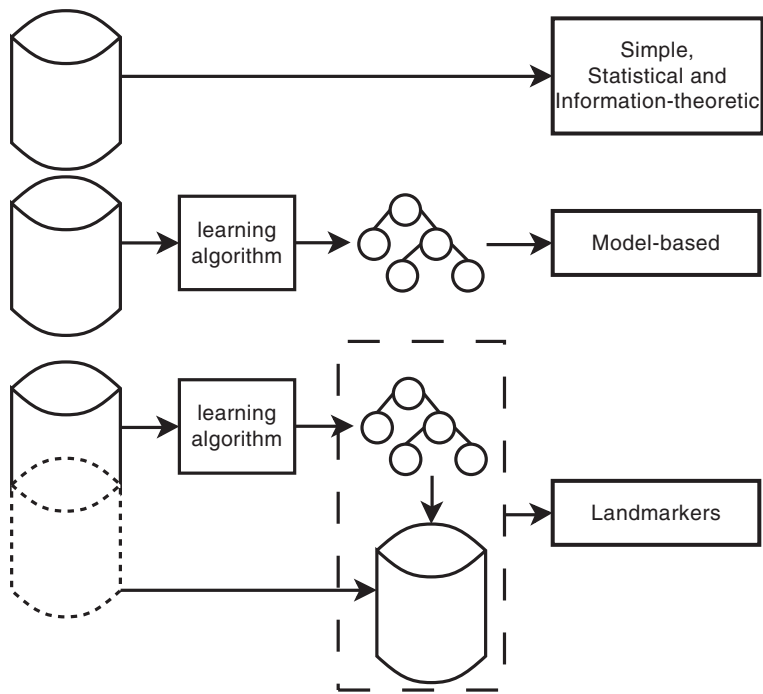


Fig. 3.4. Dataset characterization approaches

target feature (e.g., **entropy of classes** for classification tasks and **ratio of the standard deviation to the mean of the target attribute** for regression tasks). Others characterize the relationship between two or more independent features (e.g., **correlation** for numerical features or **mutual information** for symbolic features) and between independent features and the target (e.g., **correlation between independent feature and the target** for numerical features on regression tasks and **mutual information between independent feature and target** for symbolic features on classification tasks).

This type of metafeature contains information about properties of datasets, such as size, type, distribution, noise, missing values and redundancy, that usually affect the performance of learning algorithms.

A different approach is *model-based* data characterization (middle of Figure 3.4). In this approach a model is induced from the data and the metafeatures are based on properties (e.g., morphological) of that model [19, 189]. An example of a model-based data characteristic is the number of leaf nodes in a decision tree. Metafeatures obtained using this approach are only useful for algorithm recommendation if the induction of the model is sufficiently fast.

Model-based meta-features

Note that in the first approach, consisting of simple, statistical and information-theoretic measures, the metafeatures are computed directly on the dataset. In model-based data characterization, they are obtained indirectly through a model. If this model can be related to the candidate algorithms, then these approaches provide useful metafeatures.

Yet another approach to data characterization is the use of *landmarkers* [20, 190] (bottom of Figure 3.4).⁷ Landmarkers are quick estimates of algorithm performance on a given dataset. They can be obtained in two different ways. The estimates can be obtained by running simplified versions of the algorithms [20, 190]. For instance, a decision stump, i.e., the root node of a decision tree, can be the landmarker for decision trees.

Land-
markers

An alternative way of obtaining quick performance estimates is to run the algorithms whose performance we wish to estimate on a sample of the data, obtaining the so-called *subsampling landmarks* [104, 237, 158]. A different perspective is obtained by considering an ordered sequence of subsampling landmarks for a single algorithm, representing in effect a part of its learning curve [159]. In this case, metalearning can take into account not only the values of the estimates but also the shape of the curve.

Like model-based metafeatures, landmarks characterize the dataset indirectly. But they go one step further, by representing the performance of a model on a sample of the data, rather than representing properties of the model. If the performance of the landmarks is, in fact, related to the performance of the base-algorithms, we can expect this approach to be more successful than the previous ones. Some experimental results exist to support this [160].

Several studies report on comparisons of some of the approaches for data characterization mentioned here (e.g. [21, 150, 261]). However, more work is needed to determine whether one approach is definitively better or worse than the others.

Problem-Specific Data Characterization

The set of metafeatures suitable for different metalearning problems may vary substantially. The best set of metafeatures for a given metalearning problem depends essentially on the task, the datasets and the algorithms, as will be discussed next.

This chapter focuses on classification, but metalearning has been used for the recommendation of algorithms for other learning tasks, such as regression [235, 233] and time series forecasting [196, 81]. The characteristics of the base-level learning task that affect the development of metafeatures are the type of target feature (if any) and the structure of the data.

Metafeatures such as **number of classes** or **class entropy** are suitable to describe the target feature in classification but cannot be used in

Task-
dependent
meta-
features

⁷ The concept of landmarks can be related to earlier work on yardsticks [38].

regression. In this case, one could use measures such as the **number of outliers of the target feature** or the **coefficient of variation of the target**, represented as the ratio of the standard deviation to the mean of target values [235, 233]. Another example is metafeatures that relate the information in the independent features and the target. Measures that are commonly used in classification, such as the **mean mutual information of class and features**, cannot be used in other tasks. For instance, in regression, the **average absolute correlation between numeric features and the target** could be used instead. In the case of unsupervised learning tasks, such as clustering or association rule mining, there is no target variable, and therefore, no need to characterize it. However, to the best of our knowledge, no metalearning approaches have been attempted for algorithm selection in these cases.

So far, we have assumed that the data can be naturally represented using the traditional tabular format. However, this may not be the case. For instance, a simple time series is an ordered set of values. In this case, many metafeatures that are commonly used in classification may not be applicable. For instance the **correlation between numeric features** cannot be computed for a single time series. Therefore, appropriate types of measures must be used to characterize the properties of such data. The literature on the topic is a good source of information. For instance, the sample autocorrelation coefficients (given by the correlation between points which are d positions apart in the series) provide important information about the properties of a time series [64]. Several metafeatures can be derived from these coefficients, such as **mean absolute value of the first five autocorrelations** (i.e., for $d \in \{1, \dots, 5\}$) and **statistical significance of the first autocorrelation coefficient** [196, 81].

The set of base-algorithms should also be taken into account in the development of metafeatures. In the case where diverse algorithms are included, it should be considered that different sets of metafeatures are useful for discriminating the performance of different pairs of algorithms [1, 137, 136]. For instance, the **proportion of continuous features** can be useful to discriminate between naïve Bayes and k -NN, but not between naïve Bayes and a rule-based learner [136]. This is consistent with the knowledge that k -NN is better suited for continuous features than naïve Bayes, but both the naïve Bayes and rule-based systems have problems to deal with this kind of attributes. Therefore, a set of metafeatures that is able to discriminate among all of the algorithms should be used. For instance, a set of seven metafeatures were successfully used to discriminate between a set of very diverse algorithms that included decision trees, neural networks, k -NN and naïve Bayes [41].

Another approach is to transform the problem into several pairwise metalearning problems (i.e., predict whether to use algorithm A or B, or whether they are equivalent) and use different sets of metafeatures for each of them, defined using, for instance, feature selection methods [137].

When the base-algorithms are similar, specific metafeatures that represent the differences between them should be designed. A particular case is when the base-algorithms represent the same algorithm with different parameter settings. In the case of selecting parameters for the kernel of SVM, it has been shown that better results are obtained with algorithm-specific metafeatures than with general ones [235]. The metafeatures used in this work were based on the kernel matrices for the different kernel parameters considered. In a different approach to the selection of the kernel parameters for SVM, metafeatures characterizing the kernel matrix were combined with other metafeatures describing the data in terms of its relation to the margin [268].

Representation and Selection of Measures

The representation of data characteristics is an essential problem for metalearning, as is the representation of the features describing examples when learning at the base level. Some measures may require appropriate transformation to be predictive. For instance, the **proportion of symbolic features** is probably more informative than the **number of symbolic features** because it more accurately informs whether the dataset is essentially symbolic or numerical [41]. Another example is the metafeature **ratio of number of features relative to the number of examples**, which is also probably more suitable to assess the potential effect of the curse of dimensionality than **number of variables** [238].

Some of the measures commonly used as metafeatures are relational in nature. For instance, skewness is calculated for each numeric attribute. Given that the number of attributes varies for different datasets, this implies that the number of values describing the skewness for different datasets also varies. The most common approach to solve this problem is to do some aggregation, for instance by calculating **mean skewness**. However, it should be expected that important information may be lost by this aggregation. Alternatively, Kalousis and Theoharis [139] use a finer-grained aggregation, where histograms with a fixed number of bins are used to construct new metafeatures. For instance, the distribution of skewness values could be represented with three metafeatures corresponding to the number of attributes with skewness smaller than 0.2, between 0.2 and 0.4 and larger than 0.4.

Other approaches have exploited a relational representation of metafeatures using Inductive Logic Programming (ILP) [263] and case-based reasoning [120, 138] methods. For instance, in a dataset with k_c continuous attributes, skewness is described by k_c metafeatures, with the skewness value of each attribute. An ILP approach has also been proposed to take full advantage of the model-based approach to data characterization, which is also nonpropositional [22]. The authors illustrate their proposal by characterizing the dataset using a decision tree induced from that dataset. A typed higher-order logic language, which can describe complex structures, is used.

Besides making the choice of an adequate representation, it is important to select a suitable subset of data characteristics from all the possible alternatives. The number of metafeatures should not be too large compared to the amount of available metadata. An excessively large number of measures may cause overfitting and, thus, poor predictions on unseen data. This is particularly true because the number of examples in metalearning (i.e., datasets) is usually small.

Selection of metafeatures may be done during the development of the metalearning system by including only measures that are expected to be relevant [41]. This can be done by taking into account the characteristics of the metalearning problem, as discussed above. Alternatively, it is possible to include as many metafeatures as possible. A feature selection method can then be applied to obtain a smaller subset of suitable metafeatures. It has been shown that the use of wrapper-based feature selection methods at the meta level improves the quality of the results [262, 137].

In summary, the choice of the metafeatures must take into account the task, the evaluation measure, the characteristics of the data and the alternative methods.

Iterative Data Characterization

In the previous section we have described the process of characterization of datasets that is done prior to its use by a metalearning scheme. An alternative approach consists of gathering the metafeatures in several phases in an iterative fashion [159, 160]. In each phase the system tries to determine whether the currently available set of metafeatures is adequate or whether it should be extended, and if so, how (Figure 3.5). This is done with the help of existing information stored in the metaknowledgebase, and the aim is to determine what happened in similar circumstances in the past. If there is evidence, that some extensions lead to a marked improvement of performance, the system tries to identify the “best one”. This is the one which is expected to provide maximum information while requiring the least computational effort.

In [159, 160], the metafeatures consisted of subsampling landmarks using samples of increasing size, representing in effect *learning curves*. Characterization of datasets starts by running the algorithms on small datasets and the system determines the next sample sizes that should be tried out. We note that the plan of these experiments is built up gradually, by taking into account the results of all previous experiments, both on other datasets and on parts of the new dataset. This approach can, in principle, be adapted to other types of metafeatures.

3.3.2 Meta-examples

From a metalearning perspective, an example, referred to as a *meta-example*, is a base-level learning problem (Figure 3.6). For instance, in the algorithm recommendation setting considered in the previous chapter, each meta-example

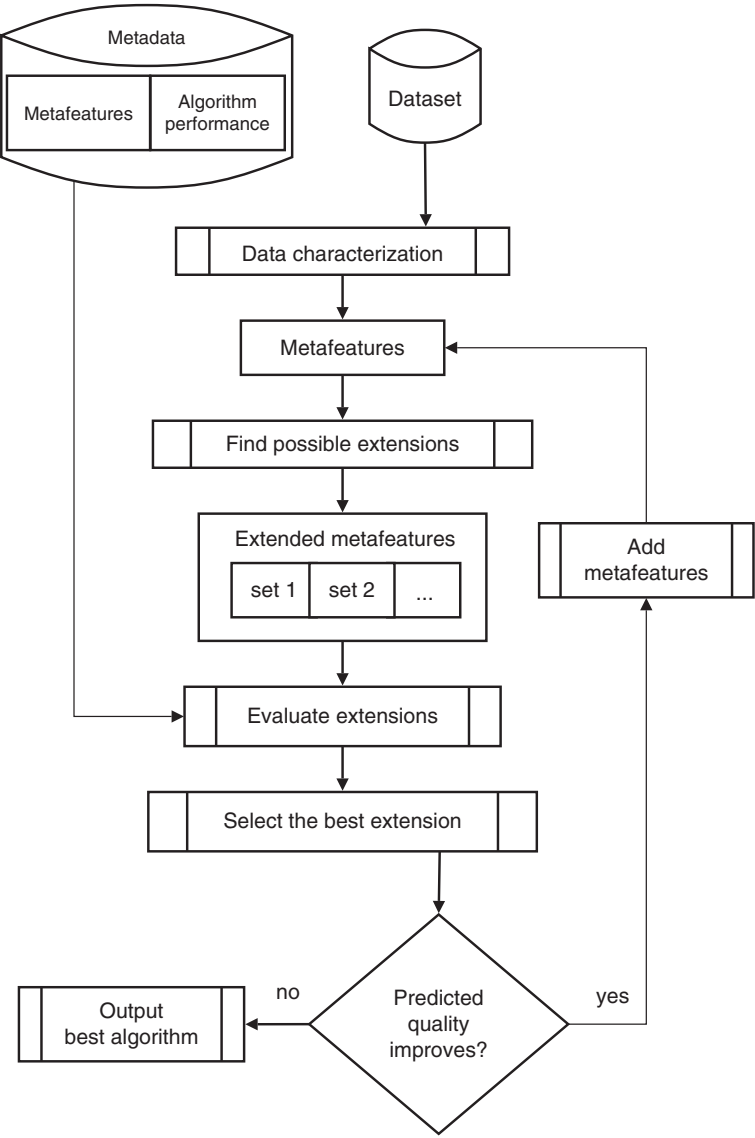


Fig. 3.5. Iterative process of characterizing the new dataset and determining which algorithm is better

captures information about a propositional dataset containing n examples described by m atomic variables representing a fixed set of metafeatures including number of examples, proportion of symbolic features and class entropy.

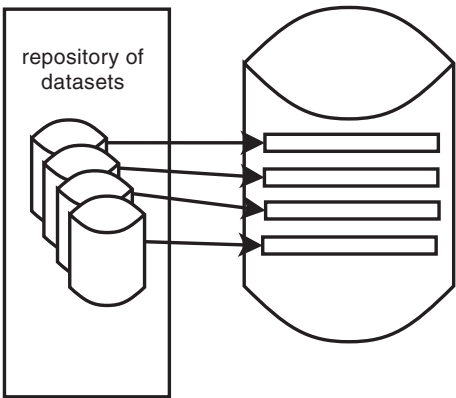


Fig. 3.6. Meta-examples are learning problems

The collection of a suitable set of meta-examples involves issues that are common to most learning problems. Here we discuss one of those issues, which is concerned with the volume of metadata available. Like in any other machine learning task, metalearning requires a number of meta-examples that is sufficient to induce a reliable recommendation model. Other issues concerning the quality of the data are discussed later (Section 3.5).

Despite the claims concerning the existence of a large number of learning problems, only very few are public domain. This is because the owners of data are usually reluctant to make it available, mostly for confidentiality reasons. Therefore only a few dozen datasets are available publicly, most of them from Websites, such as the University of California at Irvine (UCI) Machine Learning Repository [8], the UCI Knowledge Discovery in Databases Archive [118], the University of California at Riverside (UCR) Time Series Data Mining Archive [142], among others. In the UCI repository there are approximately 150 datasets. Although this is sufficient for many purposes (e.g., most comparative studies use at most a few dozen datasets), it is not much for metalearning. We cannot expect to obtain a general model for such a complex problem as algorithm recommendation using a limited number of examples. In an attempt at extending this number, the Data Mining Advisor (Chapter 4) Website [166] invites people to submit their datasets. In situations where users are unable to disclose the data, such as when the data are confidential, there is the possibility to submit only the corresponding metadata.

The generation of synthetic datasets could be regarded as the natural way to extend the number of examples for metalearning. A general methodology for this purpose has been proposed recently [271]. New datasets are generated by varying a set of characteristics that describe the concepts to be represented in the data. The characteristics include the concept model and the size of the model. The datasets generated should have similar properties to natural (i.e., real-world) data. The authors propose the use of existing techniques

Repositories
of datasets

Generation
of datasets

for experimental design as an inspiration to guide dataset generation for metalearning studies. However, they recognize that building such a generator is a challenging and ongoing task. Partial approaches have been proposed, in which the correlation between features and concepts are obtained by recursive partitioning on the space of features [220]. Given that it is difficult to make sure that the datasets generated are similar to natural ones, this approach is more suitable for understanding algorithm behavior than for the purpose of algorithm recommendation.

An alternative method to obtain more metadata is to generate new datasets by manipulating existing ones. This may be done in two ways: changing the distribution of data (e.g., adding noise to the values of independent features or changing class distribution) and by changing the structure of the problem (e.g., adding irrelevant or noisy features) [1, 119]. Usually changes are done separately on independent (e.g., adding redundant features) and dependent (e.g., adding noise to the target feature) features. The metaknowledge that can be obtained from such datasets is focused on a certain aspect of the behavior of the given algorithms. For instance, the addition of a varying number of redundant features can be used to investigate the resilience of some algorithms to redundancy.

Manipulation of datasets

However, the metaknowledge obtained by generating datasets or manipulating existing ones may not be very useful for algorithm recommendation purposes. What ultimately affects the performance of algorithms is the joint distribution between the dependent features and the target. Unfortunately, changes in the joint distribution of a given dataset, such as those carried out when manipulating datasets, are either random, thus reducing to the case of adding noise to the target feature, or made according to some model, which, of course, entails a bias. This bias will, naturally, favor some algorithms relative to others. Similar drawbacks apply to methods that generate artificial datasets. However, given that no data is available to start with, the joint distribution must be defined a priori. If it is random, the data is mostly useless; otherwise some kind of bias is again favored. As mentioned earlier, this does not mean that methods that manipulate the joint distribution of existing datasets or that generate artificial ones are not useful. It simply means that the metaknowledge that can be obtained is too specific for the purpose of algorithm recommendation.

The problem of obtaining a sufficient number of meta-examples is not so acute in two emerging areas (Figure 3.7). In massive data streams, large volumes of new data are continuously available. These data are typically from a relatively stable phenomenon and the goal is either to generate new models for new batches of data or update existing models. The second area is *extreme data mining* [96], in which a large database is segmented into a large number of subsets (e.g., by customer or product) and different models are generated for each. In both cases, by regarding each batch of data as a dataset, there should be plenty of meta-examples for metalearning.

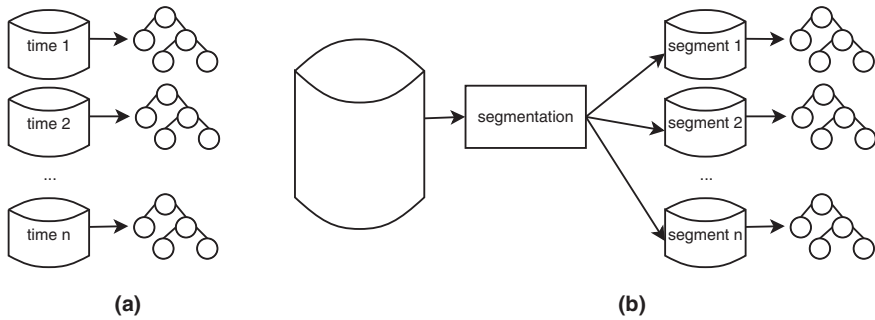


Fig. 3.7. Two emerging areas with large volume of meta-examples: (a) massive data streams and (b) extreme data mining

3.4 Base-level Algorithms

The ultimate goal of the use of metalearning for algorithm recommendation is to achieve good performance on the base-level learning problems. Therefore, careful selection of the set of base-algorithms must be made. Additionally, the selection of the measures that will be used to evaluate the performance of the algorithms must be identified, taking into account the goals of the base-level learning problems. Finally, the methodology for performance estimation must be defined, so that the values would be reliable and comparable for different algorithms. These issues are discussed in the following sections.

3.4.1 Preselection of Base-algorithms

To obtain the metadata required for the metalearning approach to algorithm recommendation, it is necessary to evaluate a set of algorithms by running them on a sufficiently large number of datasets (Figure 3.1). As this task is carried out during the development of the metalearning system, time is not such a critical factor as it is when the system is deployed.

However, as computational resources are limited, it is not possible to consider every possible alternative; otherwise metadata would probably not be ready in time for the system to be useful. Additionally, one must not forget that most algorithms have parameters. In some cases, such as in neural networks and support vector machines (SVMs), the performance of the algorithm varies significantly with different parameter settings. Many of these parameters (e.g., width of the kernel in SVM with Gaussian kernel) are continuous, meaning that the number of alternative values is infinite.

In most applications some hard constraints are used to simplify this task considerably. The choice is limited by *availability* and *applicability*. Availability simply means that the user can only consider an algorithm if he or she has access to an implementation of that algorithm. Many users rely on commercial

data mining suites such as SAS Enterprise Miner⁸ and SPSS Clementine,⁹ or tools implementing a single algorithm, such as See5.¹⁰ Others rely on free software, such as the WEKA data mining suite¹¹ or in-house developed tools. Many tools implement less than ten algorithms, which means that most of the time the number of algorithms actually available will be of that order. The choice of values of parameters to consider is more complex essentially because the number of alternatives is very large, as mentioned above.

Applicability depends on whether the use of the algorithm for the specific problem is possible. For instance, if the goal is to predict a continuous variable, then only regression algorithms can be used.

However, the number of available alternatives is still large and the computational costs of generating metadata are sufficient to justify a careful selection of the set of algorithms and parameter settings to be considered. Information that is relevant for that purpose can be obtained from prior knowledge about the general behavior of algorithms as well as about the learning problems to which the metalearning system is to be applied. Additionally, it is important to have some method to determine whether a given set of alternatives is suitable or not. These issues are discussed next.

Existing Metaknowledge

The literature on learning, for both theoretical and empirical approaches, contains some metaknowledge that can be useful to preselect base-learners. This kind of knowledge is usually suitable to eliminate alternatives which, although applicable, are very unlikely to obtain competitive results. However, it is not sufficiently detailed to reduce the number of alternatives enough for recommendation purposes, i.e., to pick a small set of alternatives which are expected to perform best.

Some of this metaknowledge is of theoretical origin. For instance, some algorithms are based on strong assumptions concerning the data. Two examples are discriminant analysis methods that assume that the data are normally distributed and naïve Bayes that assumes that variables are independent. Although there are empirical results which show that these algorithms tolerate some violations of their underlying assumptions (e.g., [39, 80]), this metaknowledge can be used to eliminate some options. For instance, when it is known that the metalearning system will be deployed on data containing many variables that are dependent on each other, naïve Bayes should not be included in the set of preselected base-algorithms.

Metaknowledge can also be obtained from empirical studies. However, this kind of metaknowledge is usually based on a small set of problems, which

Theoretical
meta-
knowledge

Empirical
meta-
knowledge

⁸ <http://www.sas.com/technologies/analytics/datamining/miner/>.

⁹ <http://www.spss.com/clementine/>.

¹⁰ <http://www.rulequest.com/see5-info.html>.

¹¹ <http://www.cs.waikato.ac.nz/ml/weka/>.

affects its generality, and should therefore be used with caution. A recent study involving ten classification algorithms and 80 UCI datasets empirically investigates questions such as [135]:

- what are the characteristics of datasets on which the given algorithms exhibit very low or very high error correlation?
- what are the characteristics of datasets on which all given algorithms are expected to perform the same?
- when does boosting significantly improve the results obtained with C5.0?

One of the observations made is that the algorithms analyzed tended to have higher error correlation on datasets with insufficient data (i.e., low number of classes or unbalanced class distributions, limited number of examples relative to the number of classes or attributes). This metaknowledge can be used for preselection of base-algorithms: to provide a recommendation for problems with a limited amount of data, only a subset of those algorithms need be considered.

In a different study, three methods for the construction of ensembles of decision trees were compared, namely bagging, boosting and randomization [77]. In this work, several useful indications were also obtained. For instance, the results presented show that boosting is more adequate than bagging in situations with little classification noise, and viceversa.

An alternative approach was taken in the StatLog project [169], in which 23 algorithms were tested on 22 datasets. The results of the algorithms were analyzed not only in terms of data characteristics but also by grouping the datasets in application domains, such as credit risk and image processing. Some rather interesting observations were made [39]. For instance, algorithms for the induction of decision trees obtained better results on credit risk applications than the others. The explanation given for this is that these datasets contain historical data of applications that were judged by human experts. Given that these judgments are based on the attributes of those applications, these datasets contain partitioning concepts, which makes it easier for recursive partitioning datasets to induce the corresponding model accurately.

In organizations where data mining is a regular activity, the knowledge about the data, the learning problem and past results with learning algorithms can also be used in the preselection of alternatives for the metalearning system.

Heuristic Experimental Evaluation of a Set of Base-algorithms

Given a set of preselected base-algorithms, it is necessary to assess whether they are suitable for a given algorithm recommendation problem or not. On one hand, the set should not be too large; otherwise the computational effort required to generate the metadata is too cumbersome. On the other hand, it should contain the algorithms that enable the users of the metalearning system to obtain satisfactory results.

A heuristic experimental approach to determine the adequacy of a set of base-algorithms has recently been proposed [236]. It consists of applying the preselected algorithms to a sample of datasets from the application domain. The selected algorithms are deemed adequate if the results verify a few properties. The proposed properties are concerned with the whole set of algorithms (“overall” rules) or each of them individually. Furthermore, the properties concern either the relevance of the algorithms (i.e., whether nontrivial results can be obtained) or their competitiveness (i.e., whether near-optimal results can be obtained). Given a set of datasets and assuming that the preselected set of m alternative algorithms, and possibly also parameter settings, is $P = p_1, \dots, p_m$, the properties are:

1. Overall relevance: For most datasets there should be a p_i that obtains better performance than a suitable baseline. A baseline is a simple method which establishes a reference for minimum acceptable results.
2. Overall competitiveness: Given some preselected set P , the results cannot be further significantly improved by adding additional elements to it.
3. Individual competitiveness: For every element p_i , it should be possible to identify at least one dataset for which p_i is the best alternative from the preselected set, P .
4. Individual relevance: For every p_i , there should not exist a p_j such that the performance of p_i is never significantly better than that of p_j for all datasets considered.

In practice it is difficult to guarantee that a given set of algorithms verifies all four properties. However, simple methods have been proposed that enable the estimation of how adequate the selection is [236]. These methods were tested in the context of recommending parameter settings for SVM, which was described in the previous chapter. However, they are equally applicable in the case of sets of learning algorithms.

3.4.2 Evaluation of Base-level Algorithms

The target variable in a metalearning system for algorithm recommendation is based on the performance of the base-level algorithms. The performance of learning algorithms can be quantified in many different ways, including accuracy and area under the ROC curve for classification [123] and residual-based measures (e.g., mean squared error) and ranking-based measures for regression [209]. Additionally, there are several experimental procedures to estimate the performance, including hold-out and cross-validation [114, Ch. 7]. Whatever measure and procedure are selected, the basic metalearning method remains unaltered; therefore the issue is not discussed here and the reader is referred to appropriate sources for more information.

On the other hand, evaluation of learning models in data mining applications using a single criterion is often insufficient [179]. Users may require

that algorithms be fast and generate interpretable models, besides being accurate. It is, thus, important that metalearning systems be able to deal with multicriteria evaluation of learning algorithms [152]. One of the difficulties of multicriteria evaluation is that different users have different preferences concerning the relative importance of the criteria involved. For instance, one user may prefer faster algorithms that generate interpretable models even if they are not so accurate. Another user may need accurate and fast algorithms and not be interested in analyzing the model. Even within the same organization, there may be users with different profiles, as illustrated by a characterization of the profiles of different data mining users in the automotive industry [25]. This makes it difficult to generate metaknowledge that is applicable over a wide range of different profiles.

Typically, the combination of several criteria is done by constructing an aggregate measure based on those criteria. In this case, the user has to quantify how important each criterion is. One such measure for algorithm evaluation combines the accuracy and execution time of classification algorithms [41]. The relative importance of both criteria is determined by the amount of accuracy the user is willing to trade for a tenfold increase or decrease in execution time.

An alternative is to use Data Envelopment Analysis (DEA) [63] for multicriteria evaluation of learning algorithms [179]. One of the important characteristics of DEA is that the weights of the different criteria are determined by the method and not the user. However, this flexibility may not always be entirely suitable, and so a variant of DEA that enables personalization of the relative importance of different criteria should be used [180].

The development of suitable multicriteria measures to evaluate learning algorithms faces two challenges. Firstly, the compromise between the criteria should be defined in a way that is clear to the end user. Secondly, the measure should yield values that can be clearly interpreted by him or her.

3.5 Quality of Metadata

We have discussed how to generate metadata both for characterizing datasets (Section 3.3) and for assessing the performance of base-algorithms (Section 3.4). The metadata may contain deficiencies that affect the reliability of the algorithm recommendation systems obtained by metalearning (Section 3.2). Here, we discuss some issues that affect metadata quality, including the representativeness of the data sample and missing or unreliable values.

3.5.1 Representativeness of Metadata

Learning is only useful if the data sample used for training is representative of its domain. If not, the model that is obtained cannot be used to make reliable predictions in future cases. In metalearning, obtaining a representative sample

of metadata means that the datasets that constitute the meta-examples for learning should be representative of the datasets for which the system will provide recommendations in the future. For instance, metalearning research often aims to develop general algorithm recommendation methods based on datasets from the UCI repository [8]. In spite of positive results, one should not forget that the number of datasets is relatively small, precluding the generation of metaknowledge that would be widely applicable. On the other hand, the datasets used need not be relevant for a new application domain.

Additionally, some argue that the datasets in the UCI repository [8] cannot be regarded as a sample of “real-world” data mining applications [212]. First of all, because most of the problems in these repositories consist of datasets which have already been heavily preprocessed, while data mining problems typically require a significant amount of preparation before a learning algorithm can be applied. Additionally, it is argued that they are not relevant for real world applications in general, although they may be useful to establish some relationships between classes of problems and the performance of algorithms [212]. However, notwithstanding their use as training data for metalearning systems, very few attempts have been made to systematically investigate the real-world relevance of repository datasets (e.g., [232]).

The problem of the representativeness of the metadata is minimized in the areas of massive data streams and extreme data mining, discussed earlier (Section 3.3.2). In these cases, the meta-examples are different batches of data from the same application. Therefore, they can typically be expected to be representative samples of future batches.

3.5.2 Missing and Unreliable Performance Data

As shown in Figure 3.1, metadata include information about the performance of the base-algorithms on selected datasets. These data may be missing for several reasons.

Performance data could be missing because the corresponding experiments have not been executed. This may occur when the system is being extended with new datasets or new base-algorithms. A new dataset represents a new meta-example and, as mentioned earlier, the more the number of meta-examples available, the better the expected metaknowledge. Therefore, it is important to extend the system with the metadata from new datasets when they become available. This implies running all the available algorithms on the new dataset, which may be computationally very expensive. This cost increases with the size of the dataset and the number of alternative algorithms.

Alternatively, when a new base-algorithm becomes available it is necessary to update the metaknowledge, so that the system could consider it in the recommendations provided. For that purpose, the metadata describing the performance of algorithms on known datasets (i.e., meta-examples) must be extended with information concerning the new algorithm. It is therefore necessary to run it on those datasets, which may require significant computational effort.

Update of
metadata

One approach is to run all experiments off-line and update the metadata only after all results become available. It is clear that this will take a long time. An alternative approach is to again use metalearning, this time to support the process of metadata collection. This can be done in two ways. Firstly, it can be used to generate estimates of the performance that replace the true performance data until it becomes available. Secondly, it can guide the experimentation process (i.e., the algorithms that are expected to perform best are executed first), akin to active learning. As experiments finish, the corresponding results are added to the metadata and replace the information provided by the initial recommendation. It is conceivable that the system may function quite well without ever completing all the tests. An important line of future work is to establish which tests should be run and which ones could be omitted and yet maintain a satisfactory level of performance.

A different cause for missing performance data is failures in the execution of base-algorithms. In some cases it is possible to recover from such failures (e.g., insufficient memory) but there are cases when the performance of an algorithm on a dataset cannot be estimated (e.g., software bug). The former case is similar to the missing data problem described above, and can be solved by adequate corrective measures (e.g., add more memory). The latter type of missing data is quite different. If an algorithm cannot be applied to a dataset, its performance is not quantifiable, although it is not missing. One approach to deal with this issue is to penalize such algorithms by following some strategy. The simplest strategy could be to make predictions based on simple statistics from the data. In classification, this would be predicting the most frequent class and in regression it would be predicting the mean target value. The estimated performance of this default strategy would be used to replace the performance of the algorithms that fail. More complex default strategies could be to use a fast algorithm (e.g., linear discriminant or linear regression).

Even when performance metadata is available, it can be quite unreliable. Performance metadata is usually estimated using methods such as hold-out or cross-validation [114]. The values obtained are estimates of the true performance of the algorithms and may be misleading. For instance, the estimated performance of two base-algorithms may be different, but this difference may not be statistically significant. If metalearning methods do not take into account the significance of the differences between the algorithms, then they may generate models that provide erroneous recommendations. It is, therefore, important that metadata also includes information about the confidence interval of the performance estimates and that metalearning methods make good use of that information. For instance, when the algorithm recommendation problem is split into several pairwise comparison subproblems (i.e., select algorithm A or B), the metalearning method can be developed to deal with a third possibility, which is that the algorithms are tied [139]. In ranking, the metalearning method can also be prepared to deal with ties, which happens when two or more algorithms are ranked in the same position.

Failures of
algorithms

Estimation
of perfor-
mance

3.5.3 Missing Metafeatures

The values of metafeatures may also be missing in metadata. Again, this may be due to a failure in the computation of the metafeature. In this case, independently of their being recoverable or not, we may use common methods for filling in missing values [210].

A more complex problem arises when a given metafeature is not computable for a given dataset. For instance, **mean skewness**, which has been mentioned earlier, can only be computed if the dataset contains at least one numeric feature. One approach is to use a special value to represent the **mean skewness** of datasets which have no numeric features, such as **not applicable** [41, 139]. The method used for meta-level learning should be able to handle such a special value. In the k -NN method described in Chapter 2, this affects how distances are calculated. For instance, if two datasets have no numeric features, it seems reasonable to assume that they are close to each other with respect to this metafeature. So, it makes sense to define that the distance is 0. Furthermore, if one dataset has no numeric features but the other has at least one, they can be considered quite different with respect to the **mean skewness** metafeature. In this case the distance is assigned a very high value.

3.6 Discussion

The development of a metalearning system for algorithm recommendation involves many complex issues. Some, such as the form of recommendation, affect the kind of usage that the system can have. Others, including the set of metafeatures used to describe datasets, affect the quality of the metalearning model. Finally, there are issues that have impact on the computational complexity of the metalearning process (e.g., the number of base-level algorithms) and on the recommendation process (e.g., the computational complexity of the data characterization methods). These issues have been discussed in this chapter, which has provided an overview of existing approaches.

Extending Metalearning to Data Mining and KDD

Although a valid intellectual challenge in its own right, metalearning finds its real *raison d'être* in the practical support it offers Data Mining practitioners. The metaknowledge induced by metalearning provides the means to inform decisions about the precise conditions under which a given algorithm, or sequence of algorithms, is better than others for a given task. Without such knowledge, intelligent but *uninformed* practitioners faced with a new Data Mining task are limited to selecting the most suitable algorithm(s) by trial and error. With the large number of possible alternatives, an exhaustive search through the space of algorithms is impractical; and simply choosing the algorithm that somehow “appears” most promising is likely to yield suboptimal solutions. Furthermore, the increased amount and detail of data available within organizations is leading to a demand for a much larger number of models, up to hundreds or even thousands, a situation leading to what has been referred to as Extreme Data Mining [96]. Current approaches to Data Mining remain largely dependent on human efforts and are thus not suitable for this kind of extreme setting because of the large amount of human resources required. Since metalearning can help reduce the need for human intervention, it may be expected to play a major role in these large-scale Data Mining applications. In this chapter, we describe some of the most significant attempts at integrating metaknowledge in Data Mining decision support systems.

While Data Mining software packages (e.g., Enterprise Miner,¹ Clementine,² Insightful Miner,³ PolyAnalyst,⁴ KnowledgeStudio,⁵ Weka,⁶ Rapid-Miner,⁷ Xelopes⁸) provide user-friendly access to rich collections of algorithms,

¹ <http://www.sas.com/technologies/analytics/datamining/miner/>

² <http://www.spss.com/clementine/>

³ <http://www.insightful.com/products/iminer/default.asp>

⁴ <http://www.megaputer.com/products/pa/index.php3>

⁵ <http://www.angoss.com/products/studio.php>

⁶ <http://www.cs.waikato.ac.nz/ml/weka/>

⁷ <http://rapid-i.com/content/blogcategory/10/69/> (formerly known as Yale)

⁸ <http://www.prudsys.com/Produkte/Algorithmen/Xelopes/>

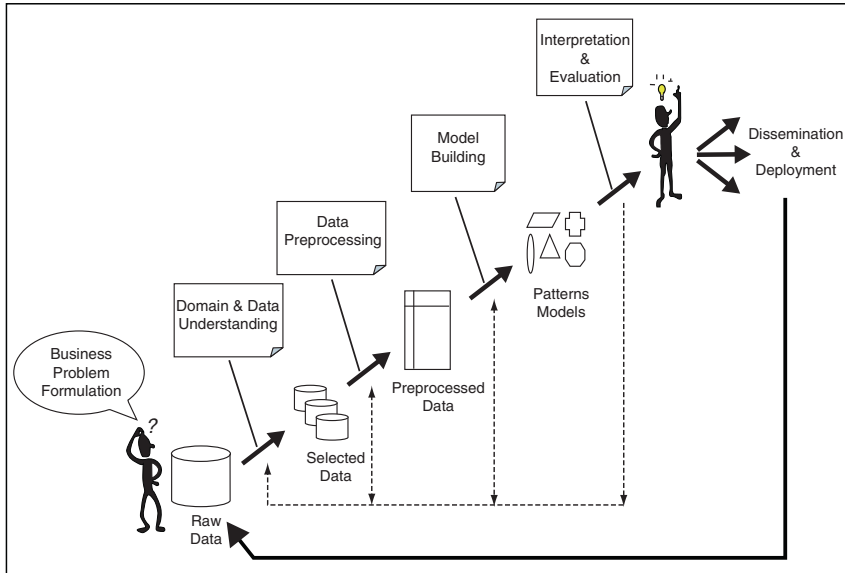


Fig. 4.1. The KDD process

they generally offer no real decision support to nonexpert end users. Similarly, tools with emphasis on advanced visualization (e.g., [121, 122]) help users understand the data (e.g., to select adequate transformations) and the models (e.g., to adjust parameters, compare results, and focus on specific parts of the model), but treat algorithm selection as an activity driven by the users rather than the system. The discussion in this chapter purposely leaves out such software packaging and visualization tools. The focus is strictly on systems that guide users by producing *explicit* advice automatically.

It is clear that not all decision points in the KDD process (see Figure 4.1) lend themselves naturally to automatic advice. Typically, both the early stages (e.g., problem formulation, domain understanding) and the late stages (e.g., interpretation, evaluation) require significant human input as they depend heavily on business knowledge.

The more algorithmic stages (i.e., preprocessing and model building), on the other hand, are ideal candidates for automation through adequate use of metaknowledge. Some decision systems focus exclusively on one of these stages, while others take a holistic approach, considering all stages of the KDD process collectively (i.e., as sequences of steps, or plans). In this chapter, we examine representatives of both types of systems. We further distinguish between approaches where the advice takes the form of “select 1 in N ” alternatives, and those that produce a ranking of all of the alternatives. Finally, we conclude with a brief description of agent-based approaches to metalearning.

4.1 Consultant and Selecting Classification Algorithms

The European ESPRIT research project MLT [175, 147, 69] was one of the first formal attempts at addressing the *practice* of machine learning. To facilitate such practice, MLT produced a rich toolbox consisting of a number of symbolic learning algorithms for classification, datasets, standards and know-how. Considerable insight into many important machine learning issues was gained during the project, much of which was translated into rules that form the basis of Consultant-2, the user guidance system of MLT.

Consultant

Consultant-2 is a kind of expert system for algorithm selection. It functions by means of interactive question-answer sessions with the user. Its questions are intended to elicit information about the data, the domain and user preferences. Answers provided by the user then serve to fire applicable rules that lead to either additional questions or, eventually, a classification algorithm recommendation. Several extensions to Consultant-2, including user guidance in data preprocessing, were suggested and reflected in the specification of a next version called Consultant-3 [224]. To the best of our knowledge, however, Consultant-3 has never been implemented.

Although its knowledge base is built through expert-driven knowledge engineering rather than via metalearning, Consultant-2 stands out as the first automatic tool that systematically relates application and data characteristics to classification learning algorithms.

4.2 DMA and Ranking Classification Algorithms

The Data Mining Advisor (DMA) [79] is the main product of METAL, another European ESPRIT research project [166]. The DMA is a Web-based metalearning system for the automatic selection of model building algorithms in the context of classification tasks.⁹ Given a dataset and goals defined by the user in terms of accuracy and training time, the DMA returns a list of algorithms that are ranked according to how well they meet the stated goals. The ten algorithms considered by the DMA are: three variants of C5.0 (c50rules, c50tree and c5.0boost) [198], Linear tree (ltree) [107], linear discriminant (lindiscr) [169], MLC++ IB1 (mlcib1) and naïve Bayes (mlcnb) [148], SPSS Clementine's Multilayer Perceptron (clemMLP) and RBF Networks (clemRBFN), and Ripper [66].

Data
Mining
Advisor

The DMA guides the user through a wizard-like step-by-step process consisting of the following activities.

1. Upload Dataset. The user is asked to identify the dataset of interest and to upload it into the DMA. Sensitive to the confidential nature of some data, the DMA offers three levels of privacy, as follows.

⁹ METAL also studied automatic algorithm selection in the context of regression, but the corresponding research results are not reflected in the DMA yet.

- Low: Both base-level data and derived metadata (i.e., task characterization) are public. All users of the DMA have full access to the dataset and its characterization.
 - Intermediate: The base-level data is private but the derived metadata is public. Only the data owner may access the dataset and run algorithms on it, but all users may generate rankings for it and use its associated characterization.
 - High: Both base-level data and metadata are private. Only the data owner may access the dataset, generate rankings for the associated task, run algorithms on it, and use it as metadata.
2. Characterize Dataset. Once the dataset is loaded, its characterization, consisting of statistical and information-theoretic measures, such as number of instances, skewness and mutual entropy, is computed. This characterization becomes the meta-level instance that serves as input to the DMA's metalearner.
 3. Parameter Setting and Ranking. The user chooses the selection criteria and the ranking method, and the DMA returns the corresponding ranking of all available algorithms.
 - Selection criteria: There are two criteria influencing selection, namely accuracy and training time. In the current implementation, the user may choose among three predefined trade-off levels corresponding intuitively to main emphasis on accuracy, main emphasis on training time, and compromise between the two.
 - Ranking method: The DMA implements two ranking mechanisms, one based on exploiting the ratio of accuracy and training time [41] and the other based on the idea of Data Envelopment Analysis [5, 25].
 4. Execute. The user may select any number of algorithms to execute on the dataset. Although the induced models themselves are not returned, the DMA reports tenfold cross-validation accuracy, true rank and score, and, when relevant, training time. A simple example is shown in Figure 4.2, where some algorithms were selected for execution (the main selection criteria here are accuracy in (a) and training time in (b)).

The DMA's choice of providing rankings rather than "best-in-class" is motivated by a desire to give as much information as possible to the user. In a "best-in-class" approach, the user is left with accepting the system's prediction or rejecting it, without being given any alternative. Hence, there is no recourse for an incorrect prediction by the system. Since ranking shows all algorithms, it is much less brittle as the user can always select the next best algorithm if the current one does not appear satisfactory. In some sense, the ranking approach subsumes the "best-in-class" approach. Empirical evidence suggests that the best algorithm is generally within the top three in the rankings [41].

Ranking table
[Download 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
[Download 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

Fig. 4.2. Proposed and selected actual rankings for a sample task

4.3 MiningMart and Preprocessing

MiningMart, another large European research project [170], focused its attention on algorithm selection for preprocessing rather than for model building [90, 91, 176, 89]. Preprocessing generally consists of nontrivial sequences of operations or data transformations, and is widely recognized as the most time consuming part of the KDD process, accounting for up to 80% of the overall effort. Hence, automatic guidance in this area can indeed greatly benefit users.

The goal of MiningMart is to enable the reuse of successful preprocessing phases across applications through case-based reasoning. A model for meta-data, called M4, is used to capture information about both data and operator chains through a user-friendly computer interface. The complete description of a preprocessing phase in M4 makes up a *case*, which can be added to

Mining-
Mart

MiningMart’s case base.¹⁰ “To support the case designer a list of available operators and their overall categories, e.g., feature construction, clustering or sampling is part of the conceptual case model of M4. The idea is to offer a fixed set of powerful pre-processing operators, in order to offer a comfortable way of setting up cases on the one hand, and ensuring re-usability of cases on the other.” [176].

Given a new mining task, the user may search through MiningMart’s case base for the case that seems most appropriate for the task at hand. M4 supports a kind of business level, at which connections between cases and business goals may be established. Its more informal descriptions are intended to “help decision makers to find a case tailored for their specific domain and problem.” Once a useful case has been located, its conceptual data can be downloaded. The local version of the system then generates preprocessing steps that can be executed automatically for the current task. MiningMart’s case base is publicly available on the Internet [171]. As of June 2006, it contained five fully specified cases.

A less ambitious, yet worthy of note, attempt at assisting users with pre-processing has been proposed, with specific focus on data transformation and feature construction [191]. The system works at the level of the set of attributes and their domains, and an ontology is used to transfer across tasks and suggest new attributes (in new tasks based on what was done in prior ones). Preliminary results on small cases appear promising.

4.4 CITRUS and Selecting Processes

Born out of practical challenges faced by researchers at Daimler-Benz, AG (now Daimler AG), CITRUS is perhaps the first implemented system to offer user guidance for the complete KDD process, rather than for just a single phase of the process [86, 87, 283, 273].¹¹ Starting from a nine-step process description — a kind of extended version of what CRISP-DM [61] would eventually become — the designers of CITRUS built their system as an extension of SPSS’s well-known KDD tool Clementine.

CITRUS consists of three main components:

1. An information manager that supports modeling and result retrieval via an object-oriented schema,
2. An execution server that supports effective materialization and optimizes sequences of operations, and
3. A user guidance module that assists the user through the KDD process.

¹⁰ Case descriptions are too large to be included here, but MiningMart’s case base can be browsed at <http://mmart.cs.uni-dortmund.de/caseBase/index.html>

¹¹ In the last of these references, the system seems to have been renamed MEDIA (Method for the Development of Inductive Applications).

The philosophy of CITRUS is that “the user is always in control of the process and user guidance is essentially a powerful help mechanism” [87]. Yet, CITRUS offers the following three kinds of rather extensively automated means of building KDD applications, where a KDD application is viewed as a sequence of operations — known as a *stream* in Clementine and a *DM process* in IDAs (see Section 4.5).

- *Design a stream from scratch.* Here, the user is free to construct a stream by connecting together operations selected from Clementine’s rich palette. CITRUS checks preconditions, makes suggestions as to what operations might be required, and essentially maintains the integrity of the stream.
- *Design a stream from existing ones.* Here, the user simply provides a high-level description of the task at hand. CITRUS acts as a kind of case-based reasoning system, which searches for and identifies closest matches in past experiences. These experiences may be real tasks previously performed or basic templates designed by experts. The closest match is presented to the user, who in turn can adapt it to the new target task.
- *Design a stream via task decomposition.* Here, the user provides a (high-level) problem description and a goal. CITRUS acts as a kind of interactive planning system, which guides the user through a series of task decompositions, ultimately leading to specific algorithms that may be executed in sequence on the subtasks to provide the expected result from the stated problem description or start state.

Algorithm selection takes place in two stages, consisting of first mapping tasks to classes of algorithms and then selecting an algorithm from the selected class. The mapping stage is effected via decomposition and guided by high-level pre- and post-conditions (e.g., interpretability). The selection stage uses data characteristics (inspired by the Statlog project [195, 169]) together with a process of elimination (termed “strike-through”), where algorithms that would not work for the task at hand are successively eliminated until the system closes in on one applicable algorithm. Unfortunately, there are insufficient details to understand how data characteristics drive the elimination process.

Although there is no metalearning in the traditional sense in CITRUS, there is still automatic guidance beyond the user’s own input. CITRUS may indeed be regarded as a kind of IDA (see Section 4.5), with the exception that an IDA returns a list of ranked processes, while CITRUS works on a single process.

4.5 IDAs and Ranking Processes

The notion of Intelligent Discovery Assistant (IDA), introduced by Bernstein and Provost [23, 24], provides a template for building ontology-driven, process-oriented assistants for KDD. IDAs encompass the three main algorithmic steps of the KDD process, namely, preprocessing, model building and

post-processing. In IDAs, any chain of operations consisting of one or more operations from each of these steps is called a *Data Mining (DM) process*. The goal of an IDA is to propose to the user a list of ranked DM processes that are both valid and congruent with user-defined preferences (e.g., speed, accuracy).

The IDA's underlying ontology is essentially a taxonomy of DM operations or algorithms, where the leaves represent implementations available in the corresponding IDA. Operations are characterized by at least the following information.

- *Preconditions*: Conditions that must be met for the operation to be applicable (e.g., a discretization operation expects continuous inputs, a naïve Bayes classifier works only with nominal inputs¹²).
- *Post-conditions*: Conditions that are true after the operation is applied, i.e., how the operation changes the state of the data (e.g., all inputs are nominal following a discretization operation, a decision tree is produced by a decision tree learning algorithm).
- *Heuristic indicators*: Indicators of the influence of the operation on overall goals such as accuracy, speed, model size, comprehensibility, etc. (e.g., sampling increases speed, pruning decreases speed but increases comprehensibility).

Clearly, the versatility of an IDA is a direct consequence of the richness of its ontology.

The typical organization of an IDA consists of two components.

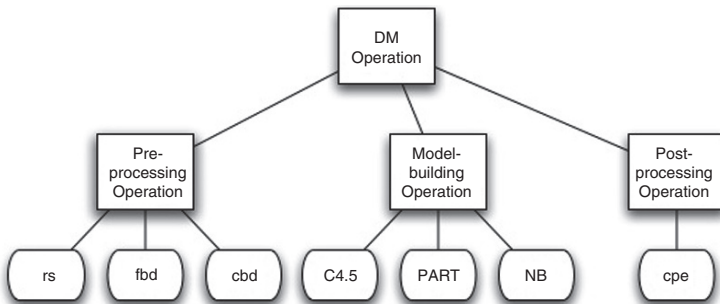
1. A *plan generator* that uses the ontology to build a list of (all) valid DM processes that are appropriate for the task at hand.
2. A *heuristic ranker* that orders the generated DM processes according to preferences defined by the user.

The plan generator takes as input a dataset, a user-defined objective (e.g., build a fast, comprehensible classifier) and user-supplied information about the data, information that may not be obtained automatically. Starting with an empty process, it systematically searches for an operation whose preconditions are met and whose indicators are congruent with the user-defined preferences. Once an operation has been found, it is added to the current process, and its post-conditions become the system's new conditions from which the search resumes. The search ends once a goal state has been reached or when it is clear that no satisfactory goal state may be reached. The plan generator's search is exhaustive: all valid DM processes are computed. Table 4.1 shows the output of the plan generator for the small ontology of only seven operations of Figure 4.3, when the input dataset is continuous-valued and comprehensible classifiers are to be preferred.

¹² In some implementations, a discretization step is integrated, essentially allowing the naïve Bayes classifier to act on any type of input.

Table 4.1. Sample list of IDA-generated DM processes

	Steps
Plan #1	C4.5
Plan #2	PART
Plan #3	rs, C4.5
Plan #4	rs, PART
Plan #5	fbd, C4.5
Plan #6	fbd, PART
Plan #7	cbd, C4.5
Plan #8	cbd, PART
Plan #9	rs, fbd, C4.5
Plan #10	rs, fbd, PART
Plan #11	rs, cbd, C4.5
Plan #12	rs, cbd, PART
Plan #13	fbd, NB, cpe
Plan #14	cbd, NB, cpe
Plan #15	rs, fbd, NB, cpe
Plan #16	rs, cbd, NB, cpe



rs = random sampling (10%), fbd = fixed-bin discretization (10 bins), cbd = class-based discretization, cpe = CPE-thresholding post-processor

Fig. 4.3. Sample IDA Ontology

The restriction of the plan generator to valid processes congruent with user-defined objectives is generally sufficient to make an exhaustive search feasible.¹³ The main advantage of this exhaustivity is that no valid DM process is ever overlooked, as is likely to be the case with most users, including experts. As a result, an IDA may — and evidence suggests that it does — uncover novel processes that experts had never thought about before, thus enriching the community’s metaknowledge.

¹³ It is unclear whether this is true in all cases. In some applications, the number of alternatives may still be too large for practical enumeration and evaluation. The work in [159], which tries to determine the best sampling strategy for a new task, may be relevant here.

Once all valid DM processes have been generated, a heuristic ranker is applied to assist the user further by organizing processes in descending order of “return” on user-specified goals. For example, the processes in Figure 4.3 are ordered from simplest (i.e., least number of steps) to most elaborate. The ranking relies on the knowledge-based heuristic indicators. If speed rather than simplicity were the objective then Plan #3 in Figure 4.3 would be bumped to the top of the list, and all plans involving random sampling (*rs* operation) would also move up. In the current implementation of IDAs, rankings rely on fixed heuristic mechanisms. However, IDAs are independent of the ranking method and, thus, they could possibly be improved by incorporating metalearning to generate rankings based on past performance.

One attractive feature of IDAs is what are called network externalities, where the work and/or expertise of one individual in a community is readily available to all other members of that community at no cost. Here, if a researcher develops some new algorithm *A* and submits it to the IDA with the information required by the ontology, algorithm *A* becomes immediately available to all IDA users, whether they know *A* or not. Hence, no single user ever needs to become an expert on all techniques and algorithms. The good will of all to share allows an IDA to act as a kind of central repository of distributed metaknowledge. This same feature is also part of the DMA (see Section 4.2), except that the filling in of the ontology by a researcher is replaced by automatic, system-generated experiments to update the metalearner.

Recent research has focused on extending the IDA approach by leveraging the synergy between ontology (for deep knowledge) and Case-Based Reasoning (for advice and (meta)learning) [62]. The system uses both declarative information (in the ontology and case base) as well as procedural information in the form of rules fired by an expert system. The case base is built around 53 features to describe cases; the expert system’s rules are extracted from introductory Data Mining texts; and the ontology comes from human experts. The system is still in the early stages of implementation.

4.6 METALA and Agent-Based Mining

Botía et al. have developed METALA, an agent-based architecture for distributed Data Mining, supported by metalearning [29, 30, 31, 32, 116].¹⁴ The aim of METALA is to provide a system that 1) supports an arbitrary number of algorithms and tasks, and 2) automatically selects an algorithm that appears best from the pool of available algorithms, using metalearning.

Each algorithm is characterized by a number of features relevant to its usage, including the type of input data it requires, the type of model it induces, and how well it handles noise. A hierarchical directory structure, based on the X.500 model, provides a physical implementation of the underlying ontology.

¹⁴ The architecture was originally known as GEMINIS and later renamed METALA.

Each learning algorithm is embedded in an agent that provides clients with a uniform interface to three basic services: configuration, model building and model application. Each agent's behavior is correspondingly governed by a simple state-transition diagram, with the three states *idle*, *configured* and *learned*, and natural transitions among them.

Similarly, each task is characterized by statistical and information-theoretic features, as in the DMA (see Section 4.2). METALA is designed so as to be able to autonomously and systematically carry out experiments with each task and each learner and, using task features as meta-attributes, induce a metamodel for algorithm selection. As new tasks and algorithms are added to the system, corresponding experiments are performed and the metamodel is updated.

Meta-
attributes
Metamodel

The latest version of METALA is a J2EE implementation on a JBoss application server. Although they were developed independently of each other, METALA may be viewed as a natural extension of the DMA. It provides the architectural mechanisms necessary to scale the DMA up to any arbitrary number of learners and tasks, in a kind of online or incremental manner. The metalearning task is essentially the same (i.e., use task characterizations and induce a metamodel), and some of the functionality of the DMA (e.g., multicriteria ranking) could be added.

4.7 GLS, Agents and Selecting Processes

Concurrent to the work on CITRUS and METALA, Zhong and his colleagues independently argued for what they called increased autonomy and versatility in KDD systems. Building on their Global Learning Scheme (GLS) [290], an agent-based architecture for the KDD process, they added a sophisticated planning and monitoring facility for automatic goal decomposition and process elaboration [288, 289].

Global
Learning
Scheme

All entities in GLS are typed and formally described using the Object-Oriented Entity Relationship model. GLS includes an extensive type hierarchy or ontology, similar in spirit to IDA's ontology (see Section 4.5). However, types in GLS correspond not only to specific activities in the KDD process (e.g., preprocessing, modeling), but also to data and knowledge (e.g., raw data, selected data, discovered knowledge) used or generated in the process. Each type is characterized by a number of relevant attributes, inherited by all subtypes, which may be descriptive (e.g., created or cleaned, for data types) as well as procedural (e.g., pre- or post-conditions, possible subtasking, actions).

Like CITRUS, GLS starts with a high-level objective and uses its type hierarchy to decompose it, via planning techniques, into a complete KDD process that may subsequently be executed. Unlike CITRUS, however, the decomposition process in GLS is completely automatic. One of the unique features of GLS is its ability to monitor itself and its environment in such a way that process changes, induced by changes in the data or the agents,

may be detected, approved and automatically adapted to. Hence, rather than having to rebuild a process from scratch when a significant change occurs in its environment, GLS uses incremental replanning to adjust the existing process plan to reflect the changes.

At this stage, GLS's meta-abilities (i.e., planning and monitoring) are implemented with static metarules only. Although there is no metalearning in the more traditional sense, GLS's ability to track and adapt to process changes can definitely be regarded as a form of learning.

4.8 Discussion

With the exception of the DMA and MiningMart, none of the systems described here are readily available. In any case, all of them remain very much works in progress. Although the ultimate Data Mining decision support system has not yet been developed — and may still be some way off — the systems described here, all of them partial in their coverage of the Data Mining process, attest to the difficulty of the endeavor. Perhaps the solution is in a combination of their strengths: the ontology of IDA and GLS, the knowledge base of Consultant-2, the metalearning and ranking of the DMA, the planning/decomposition of CITRUS and GLS, the extendible architecture of METALA and GLS, the analogy-based reuse of MiningMart, and the monitoring/adaptation of GLS.

Combining Base-Learners

Model combination consists of creating a single learning system from a collection of learning algorithms. In some sense, model combination may be viewed as a variation on the theme of combining data mining operations discussed in Chapter 4. There are two basic approaches to model combination. The first one exploits variability in the application's data and combines multiple copies of a single learning algorithm applied to different subsets of that data. The second one exploits variability among learning algorithms and combines several learning algorithms applied to the same application's data.

The main motivation for combining models is to reduce the probability of misclassification based on any single induced model by increasing the system's area of expertise through combination. Indeed, one of the implicit assumptions of model selection in metalearning is that there exists an optimal learning algorithm for each task. Although this clearly holds in the sense that, given a task ϕ and a set of learning algorithms $\{A_k\}$, there is a learning algorithm A_ϕ in $\{A_k\}$ that performs better than all of the others on ϕ , the actual performance of A_ϕ may still be poor. In some cases, one may mitigate the risk of settling for a suboptimal learning algorithm by replacing single model selection with model combination.

Because it draws on information about base-level learning — in terms of either the characteristics of various subsets of data or the characteristics of various learning algorithms — model combination is often considered a form of metalearning. This chapter is dedicated to a brief overview of model combination. We limit our presentation to a description of each individual technique and leave it to the interested reader to follow the references and other relevant literature for discussions of comparative performance among them.

To help with understanding and to motivate the chapter's organization, Table 5.1 summarizes, for each combination technique, the underlying philosophy, the type of base-level information used to drive the combination at the meta level (i.e., metadata), and the nature of the metaknowledge generated, whether explicitly or implicitly. Further details are in the corresponding sections.

Table 5.1. Model combination techniques summary

Technique	Philosophy	Metadata	Metaknowledge
Bagging	Variation in data		Implicit in voting scheme
Boosting		Errors (updated distribution)	Voting scheme's weights
Stacking	Variation among learners (multi-expert)	Class predictions or probabilities	Mapping from metadata to class predictions
Cascade generalization		Class probabilities and base level attributes	Mapping from metadata to class predictions
Cascading	Variation among learners (multistage)	Confidence on predictions (updated distribution)	Implicit in selection scheme
Delegating		Confidence on predictions	Implicit in delegation scheme
Arbitrating	Variation among learners (refereed)	Correctness of class predictions, base level attributes and internal propositions	Mappings from metadata to correctness (one for each learner)
Meta-decision trees	Variation in data and among learners	Class distribution properties (from samples)	Mapping from metadata to best model

5.1 Bagging and Boosting

Perhaps the most well-known techniques for exploiting variation in data are bagging and boosting. Both bagging and boosting combine multiple models built from a single learning algorithm by systematically varying the training data.

5.1.1 Bagging

Bagging, which stands for bootstrap aggregating, is due to Breiman [43]. Given a learning algorithm A and a set of training data T , bagging first draws N samples $S_1, \dots, S_N$, with replacement, from T . It then applies A independently to each sample to induce N models $h_1, \dots, h_N$.¹ When classifying a new query instance q , the induced models are combined via a simple voting scheme, where the class assigned to the new instance is the class that is predicted most often among the N models, as illustrated in Figure 5.1. The bagging algorithm for classification is shown in Figure 5.2.

¹ To be consistent with the literature, note that we shall use the term *model* rather than *hypothesis* throughout this chapter. However, we shall retain our established mathematical notation and denote a model by h .

Bagging

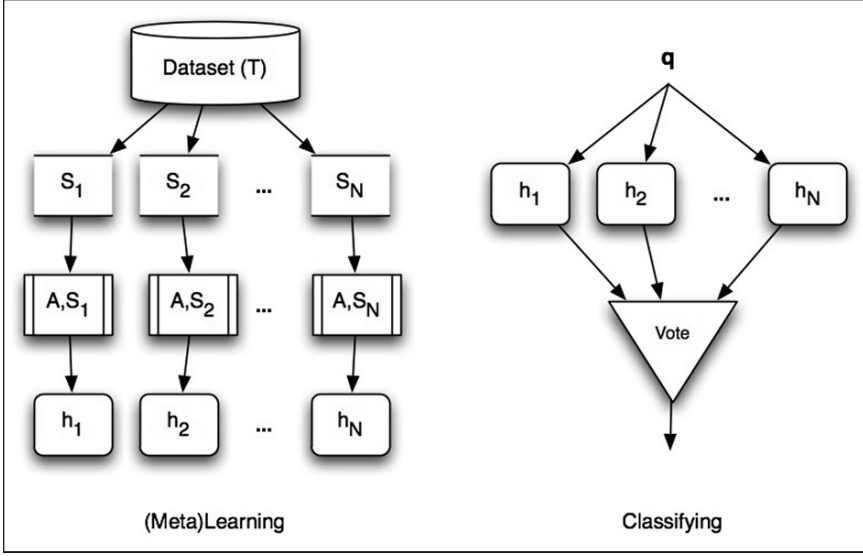


Fig. 5.1. Bagging

Algorithm Bagging(T, A, N, d)

1. For $k = 1$ to N
 2. S_k = random sample of size d drawn from T , with replacement
 3. h_k = model induced by A from S_k
4. For each new query instance q
 5. $\text{Class}(q) = \text{argmax}_{y \in \mathcal{Y}} \sum_{k=1}^N \delta(y, h_k(q))$

where:

T is the training set

A is the chosen learning algorithm

N is the number of samples or bags, each of size d , drawn from T

$\mathcal{Y}$ is the finite set of target class values

δ is the generalized Kronecker function ($\delta(a, b) = 1$ if $a = b$; 0 otherwise)

Fig. 5.2. Bagging algorithm for classification

Bagging is easily extended to regression by replacing the voting scheme of line 5 of the algorithm by an average of the models' predictions:

$$\text{Value}(q) = \frac{\sum_{i=1}^N h_i(q)}{N}$$

Bagging is most effective when the base-learner is *unstable*. A learner is unstable if it is highly sensitive to data, in the sense that small perturbations

in the data cause large changes in the induced model. One simple example of instability is order dependence, where the order in which training instances are presented has a significant impact on the learner’s output.

Bagging typically increases accuracy. However, if A produces interpretable models (e.g., decision trees, rules), that interpretability is lost when bagging is applied to A .

5.1.2 Boosting

Boosting is due to Schapire [215]. While bagging exploits data variation through a learner’s instability, boosting tends to exploit it through a learner’s *weakness*. A learner is weak if it generally induces models whose performance is only slightly better than random. Boosting is based on the observation that finding many rough rules of thumb (i.e., weak learning) can be a lot easier than finding a single, highly accurate prediction rule (i.e., strong learning). Boosting then assumes that a weak learner can be made strong by repeatedly running it on various distributions D_i over the training data T (i.e., varying the focus of the learner), and then combining the weak classifiers into a single composite classifier, as illustrated in Figure 5.3.

Unlike bagging, boosting tries actively to force the (weak) learning algorithm to change its induced model by changing the distribution over the training instances as a function of the errors made by previously generated models. The initial distribution D_1 over the dataset T is uniform, with each instance assigned a constant weight, i.e., probability of being selected for training, of $1/|T|$, and a first model is induced. At each subsequent iteration, the weights of misclassified instances are increased, thus focusing the next model’s attention on them. This procedure goes on until either a fixed number of iterations

Boosting

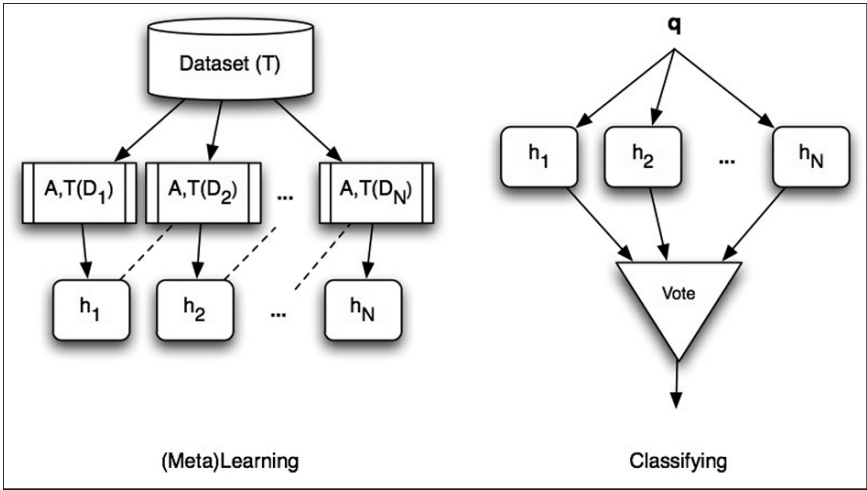


Fig. 5.3. Boosting

Algorithm AdaBoost.M1(T, A, N)

1. For $k = 1$ to $|T|$
2. $D_1(x_k) = \frac{1}{|T|}$
3. For $i = 1$ to N
4. h_i = model induced by A from T with distribution D_i
5. $\epsilon_i = \sum_{k: h_i(x_k) \neq y_k} D_i(x_k)$
6. If $\epsilon_i > .5$
7. $N = i - 1$
8. Abort loop
9. $\beta_i = \frac{\epsilon_i}{1 - \epsilon_i}$
10. For $k = 1$ to $|T|$
11. $D_{i+1}(x_k) = \frac{D_i(x_k)}{Z_i} \times \begin{cases} \beta_i & \text{if } h_i(x_k) = y_k \\ 1 & \text{otherwise} \end{cases}$
12. For each new query instance q
13. $\text{Class}(q) = \underset{y \in \mathcal{Y}}{\operatorname{argmax}} \sum_{i: h_i(q) = y} \log \frac{1}{\beta_i}$

where:

T is the training set

A is the chosen learning algorithm

N is the number of iterations to perform over T

$\mathcal{Y}$ is the finite set of target class values

Z_i is a normalization constant, chosen so that D_{i+1} is a distribution

Fig. 5.4. Boosting algorithm for classification (AdaBoost.M1)

has been performed or the total weight of the misclassified instances exceeds 0.5. The popular AdaBoost.M1 [101] boosting algorithm for classification is shown in Figure 5.4.

The class of a new query instance q is given by a weighted vote of the induced models. The case of regression is more complex. The regression version of AdaBoost, known as AdaBoost.R, is based on decomposition into infinitely many classes. The reader is referred to [100] for details.

Although the argument for boosting originated with weak learners, boosting may actually be successfully applied to any learner.

5.2 Stacking and Cascade Generalization

While bagging and boosting exploit variation in the data, stacking and cascade generalization exploit differences among learners. They make explicit two levels of learning: the base level where learners are applied to the task at hand, and the meta level where a new learner is applied to data obtained from learning at the base level.

5.2.1 Stacking

The idea of stacked generalization is due to Wolpert [284]. Stacking takes a number of learning algorithms $\{A_1, \dots, A_N\}$ and runs them against the dataset T under consideration (i.e., base-level data) to produce a series of models $\{h_1, \dots, h_N\}$. Then, a new dataset $\mathcal{T}$ is constructed by replacing the description of each instance in the base-level dataset by the predictions of each base-level model for that instance.² This new metadataset is in turn presented to a new learner A_{meta} that builds a metamodel h_{meta} mapping the predictions of the base-level learners to target classes, as illustrated in Figure 5.5. The stacking algorithm for classification is shown in Figure 5.6.

A new query instance q is first run through all the base-level learners to compose the corresponding query meta-instance q' , which serves as input to the metamodel to produce the final classification for q .

Note that the base-level models' predictions in line 5 (Figure 5.6) are obtained by running each instance through the models induced from the base-level dataset (lines 1 and 2). Alternatively, more statistically reliable predictions could be obtained through cross-validation as proposed in [85]. In this case, lines 1 through 6 are replaced with the following:

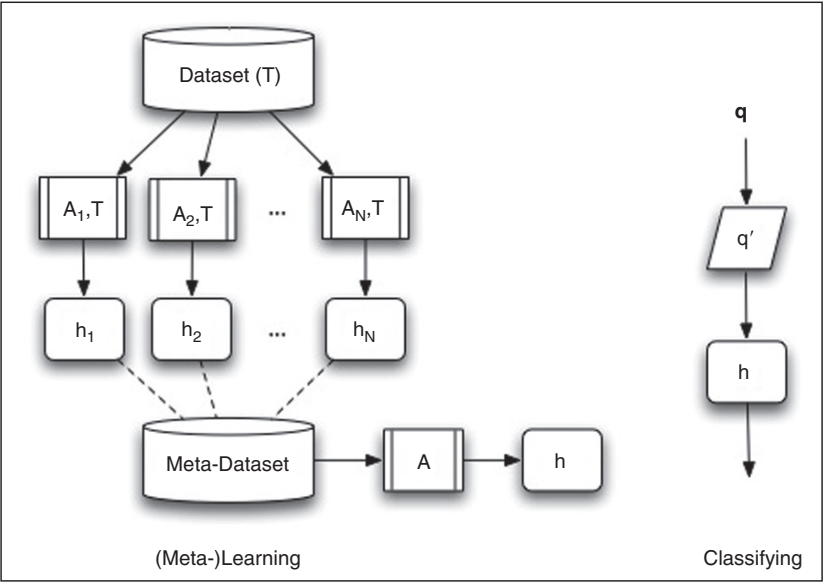


Fig. 5.5. Stacking

² In some versions of stacking, the base-level description is not replaced by the predictions, but rather the predictions are appended to the base-level description, resulting in a kind of hybrid meta-example.

Stacking

Meta-
instance

```

Algorithm Stacking( $T, \{A_1, \dots, A_N\}, A_{meta}$ )
1. For  $i = 1$  to  $N$ 
2.    $h_i$  = model induced by  $A_i$  from  $T$ 
3.  $\mathcal{T} = \emptyset$ 
4. For  $k = 1$  to  $|T|$ 
5.    $E_k = \langle h_1(x_k), h_2(x_k), \dots, h_N(x_k), y_k \rangle$ 
6.    $\mathcal{T} = \mathcal{T} \cup \{E_k\}$ 
7.  $h_{meta}$  = model induced by  $A_{meta}$  from  $\mathcal{T}$ 
8. For each new query instance  $q$ 
9.    $\text{Class}(q) = h_{meta}(\langle h_1(q), h_2(q), \dots, h_N(q) \rangle)$ 

```

where:

T is the base-level training set
 N is the number of base-level learning algorithms
 $\{A_1, \dots, A_N\}$ is the set of base-level learning algorithms
 A_{meta} is the chosen meta-level learner

Fig. 5.6. Stacking algorithm

1. For $i = 1$ to N
2. For $k = 1$ to $|T|$
3. $E_k[i] = h_i(x_k)$ obtained by cross-validation
4. $\mathcal{T} = \emptyset$
5. For $k = 1$ to $|T|$
6. $\mathcal{T} = \mathcal{T} \cup \{E_k\}$

A variation on stacking is proposed in [259], where the predictions of the base-level classifiers in the metadataset are replaced by class probabilities. A meta-level example thus consists of a set of N (the number of base-level learning algorithms) vectors of $m = |\mathcal{Y}|$ (the number of classes) coordinates, where p_{ij} is the posterior probability, as given by learning algorithm A_i , that the corresponding base-level example belongs to class j . Other forms of stacking, based on using partitioned data rather than full datasets, or using the same learning algorithm on multiple, independent data batches, have also been proposed (e.g., see [59, 260]).

The transformation applied to the base-level dataset, whether through the addition of predictions or class probabilities, is intended to give information about the behavior of the various base-level learners on each instance, and thus constitutes a form of metaknowledge.

5.2.2 Cascade Generalization

Gama and Brazdil proposed another model combination technique known as cascade generalization, that also exploits differences among learners [108]. In cascade generalization, the classifiers are used in sequence rather than in parallel as in stacking. Instead of the data from the base-level learners feeding

Cascade
generaliza-
tion

into a single meta-level learner, each base-level learner A_{i+1} (except for the first one, i.e., $i > 0$) also acts as a kind of meta-level learner for the base-level learner A_i that precedes it. Indeed, the inputs to A_{i+1} consist of the inputs to A_i together with the class probabilities produced by h_i , the model induced by A_i . A single learner is used at each step and there is, in principle, no limit on the number of steps, as illustrated in Figure 5.7. The basic cascade generalization algorithm for two steps is shown in Figure 5.8.

This two-step algorithm is easily extended to an arbitrary number of steps — defined by the number of available classifiers — through successive invocation of the `ExtendDataset` function, as illustrated in Figure 5.9, where the recursive algorithm begins with $i = 1$.³

A new query instance q is first extended into a meta-instance q' as it gathers metadata through the steps of the cascade. The final classification is then given by the output of the last model in the cascade on q' .

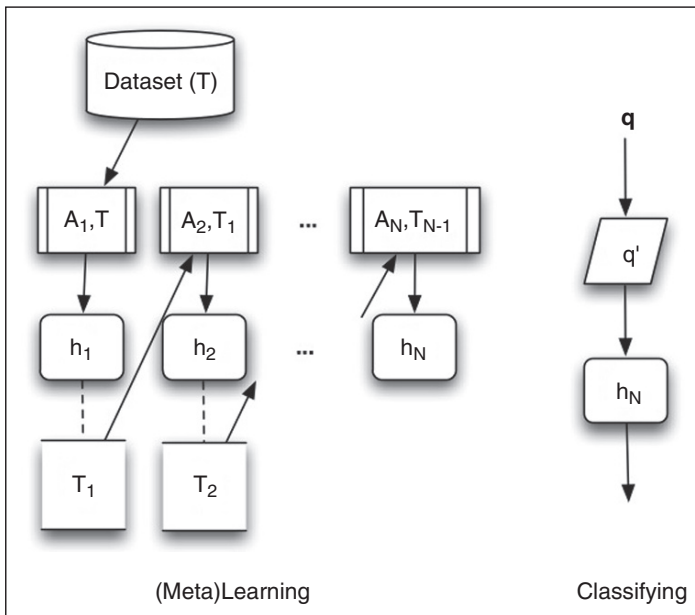


Fig. 5.7. Cascade generalization

³ To use this N -step version of cascade generalization for classification, it may be advantageous to implement it iteratively rather than recursively, so that intermediate models may be stored and used when extending new queries.

Algorithm CascadeGeneralization($\{A_1, A_2\}, T$)

1. h_1 = model induced by A_1 from T
2. T_1 = ExtendDataset(h_1, T)
3. h_2 = model induced by A_2 from T_1
4. For each new query instance q
5. $q' = \text{ExtendDataset}(h_1, \{q\})$
6. Class(q) = $h_2(q')$

where:

T is the original base level training set
 A_1 and A_2 are base level learning algorithms

Algorithm ExtendDataset(h, T)

1. $newT = \emptyset$
2. For each $\mathbf{e} = (\mathbf{x}, y) \in T$
3. For $j = 1$ to $|\mathcal{Y}|$
4. p_j = probability that e belongs to y_j according to h
5. $e' = (\mathbf{x}, p_1, \dots, p_{|\mathcal{Y}|}, y)$
6. $newT = newT \cup \{e'\}$
7. Return $newT$

where:

h is a model induced by a learning algorithm
 T is the dataset to be extended with data generated from h
 $\mathcal{Y}$ is the finite set of target class values

Fig. 5.8. Cascade generalization algorithm (two steps)

Algorithm CascadeGeneralizationN($\{A_1, \dots, A_N\}, T, i$)

1. h = model induced by A_i from T
2. If ($i == N$)
3. Return h
4. $T' = \text{ExtendDataset}(h, T)$
5. CascadeGeneralizationN($\{A_1, \dots, A_N\}, T', i + 1$)

where:

T is the original base-level training set
 N is the number of steps in the cascade
 $\{A_1, \dots, A_N\}$ is the set of base-level learning algorithms

Fig. 5.9. Cascade generalization for arbitrary number of steps

5.3 Cascading and Delegating

Like stacking and cascade generalization, cascading and delegating exploit differences among learners. However, whereas the former produce multi-expert

classifiers (all constituent base classifiers are used for classification), the latter produce multistage classifiers, in which not all base classifiers need be consulted when predicting the class of a new query instance. Hence, classification time is reduced.

5.3.1 Cascading

Alpaydin and Kaynak [4, 140] developed the idea of cascading, which may be viewed as a kind of multilearner version of boosting. Like boosting, cascading varies the distribution over the training instances, here as a function of the confidence of the previously generated models.⁴ Unlike boosting, however, cascading does not strengthen a single learner, but uses a small number of different classifiers of increasing complexity, in a cascade-like fashion, as shown in Figure 5.10.

The initial distribution D_1 over the dataset T is uniform, with each training instance assigned a constant weight of $1/|T|$, and a model h_1 is induced with the first base-level learning algorithm A_1 . Then, each base-level learner A_{i+1} is trained from the same dataset T , but with a new distribution D_{i+1} , determined by the confidence of the base-level learner A_i that precedes it. The confidence of the model h_i , induced by A_i , on a training instance x is

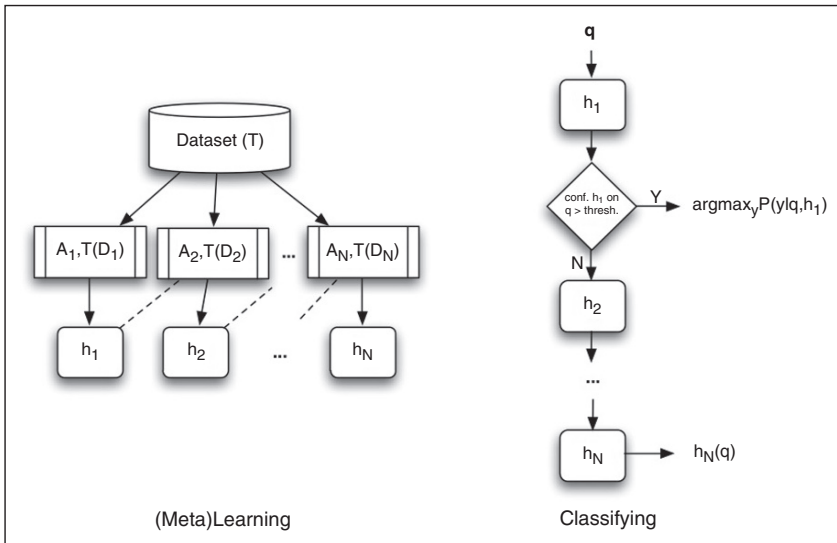


Fig. 5.10. Cascading

⁴ This is a generalization of boosting's function of the errors of the previously generated models. Rather than biasing the distribution to only those instances the previous layers *misclassify*, cascading biases the distribution to those instances the previous layers are *uncertain* about.

defined as $\delta_i(x) = \max_{y \in \mathcal{Y}} P(y|x, h_i)$. At step $i + 1$, the weights of instances whose classification is uncertain under h_i (i.e., below a predefined confidence threshold) are increased, thus making them more likely to be sampled when training A_{i+1} . Early classifiers are generally semi-parametric (e.g., multilayer perceptrons) and the final classifier is always non-parametric (e.g., k -nearest-neighbor). Thus, a cascading system can be viewed as creating rules, which account for most instances, in the early steps, and catching exceptions at the final step. The generic cascading algorithm is shown in Figure 5.11.

When classifying a new query instance q , the system sends q to all of the models and looks for the first model, h_k , from 1 to N , whose confidence on q is above the confidence threshold. If h_k is an intermediate model in the cascade, the class of the new query instance is the class with highest probability (line 15, Figure 5.11). If h_k is the final (non-parametric) model in the cascade, the class of the new query instance is the output of $h_k(q)$ (line 13, Figure 5.11).

Although the weighted iterative approach is similar, cascading differs from boosting in several significant ways. First, cascading uses different learning algorithms at each step, thus increasing the variety of the ensemble. Second, the final k -NN step can be used to place a limit on the number of steps in

Algorithm Cascading($T, \{A_1, \dots, A_N\}$)

1. For $k = 1$ to $|T|$
2. $D_1(x_k) = \frac{1}{|T|}$
3. For $i = 1$ to $N - 1$
4. h_i = model induced by A_i from T with distribution D_i
5. For $k = 1$ to $|T|$
6. $D_{i+1}(x_k) = \frac{1 - \delta_i(x_k)}{\sum_{m=1}^{|T|} 1 - \delta_i(x_m)}$
7. $h_N = k$ -NN
8. For each new query instance q
9. $i = 1$
10. While $i < N$ and $\delta_i(q) < \Theta_i$
11. $i = i + 1$
12. If $i = N$ Then
13. Class(q) = $h_N(q)$
14. Else
15. Class(q) = $\operatorname{argmax}_{y \in \mathcal{Y}} P(y|q, h_i)$

where:

T is the base-level training set

N is the number of base-level learning algorithms

$A_1, \dots, A_N$ are the base-level learning algorithms

Θ_i is the confidence threshold associated with A_i , s.t. $\Theta_{i+1} \geq \Theta_i$

$\mathcal{Y}$ is the finite set of target class values

$\delta_i(x) = \max_{y \in \mathcal{Y}} P(y|x, h_i)$ is the confidence function for model h_i

Fig. 5.11. Cascading algorithm

the cascade, so that a small number of classifiers is used to reduce complexity. Finally, when classifying a new instance, there is no vote across the induced models; only one model is used to make the prediction.

5.3.2 Delegating

A cautious, delegating classifier is a classifier that provides classifications only for instances above a predefined confidence threshold, and passes (or delegates) other instances to another classifier. The idea of delegating classifiers comes from Ferri et al. [95]. It is similar in spirit to cascading. In cascading, however, all instances are (re-)weighted and processed at each step. In delegating, the next classifier is specialized to those instances for which the previous one lacks confidence, through training *only* on the delegated instances, as illustrated in Figure 5.12. The delegation stops either when there are no instances left to delegate or when a predefined number of delegation steps has been performed. The delegating algorithm is shown in Figure 5.13.

The function $\text{getThreshold}(h, T)$ may be implemented in two different ways as follows:

- *Global Percentage.* $\tau = \max\{t : |\{e \in T : h^{CONF}(e) > t\}| \geq \rho \cdot |T|\}$, where ρ is a user-defined fraction.

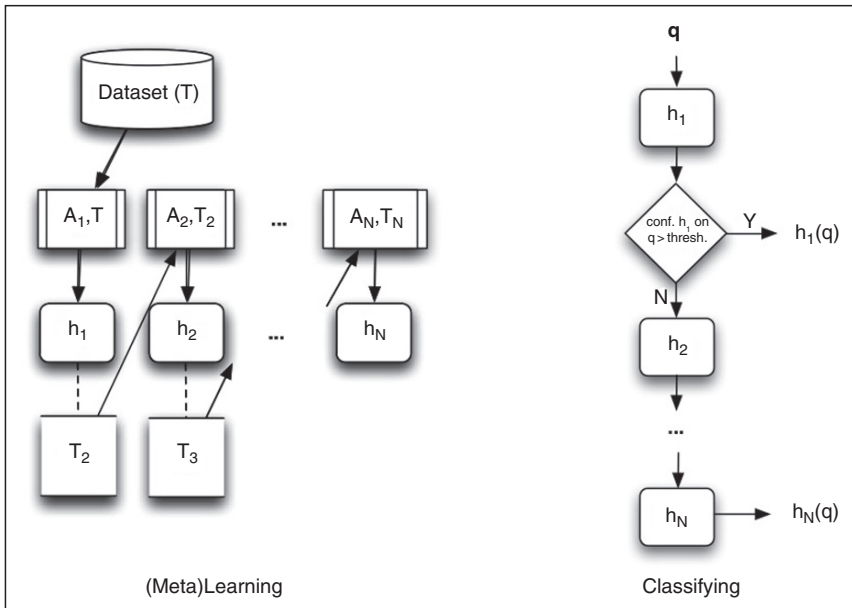


Fig. 5.12. Delegating

Algorithm Delegating($T, \{A_1, \dots, A_N\}, N, Rel$)

1. $T_1 = T$
2. $i = 0$
3. Repeat
4. $i = i + 1$
5. $h_i = \text{model induced by } A_i \text{ from } T_i$
6. If ($Rel = \text{True}$ and $i > 1$) Then
7. $\tau_i = \text{getThreshold}(h_i, T_{i-1})$
8. Else
9. $\tau_i = \text{getThreshold}(h_i, T)$
10. $T_{h_i}^> = \{e \in T_i : h_i^{CONF}(e) > \tau_i\}$
11. $T_{h_i}^{\leq} = \{e \in T_i : h_i^{CONF}(e) \leq \tau_i\}$
12. $T_{i+1} = T_{h_i}^{\leq}$
13. Until $T_{h_i}^> = \emptyset$ or $i > N$
14. For each new query instance q
15. $m = \min_k \{h_k(q) \geq \tau_k\}$
16. Class(q) = $h_m(q)$

where:

T is the base-level training set

N is the maximum number of delegating stages

$A_1, \dots, A_N$ are the base-level learning algorithms

$h_i^{CONF}(e)$ is the confidence of the prediction of model h_i for example e

Rel is a Boolean flag (true if τ_i is to be computed relative to delegated examples) $\text{getThreshold}(h, T)$ returns a confidence threshold for classifier h relative to T

Fig. 5.13. Delegating algorithm

- *Stratified Percentage.* For each class c , $\tau^c = \max\{t : |\{e \in T_c : h^{PROB_c}(e) > t\}| \geq \rho \cdot |T_c|\}$, where $h^{PROB_c}(e)$ is the probability of class c under model h for example e , and T_c is the set of examples of class c in T .

Note that there are actually four ways to compute the threshold, based on the value of the parameter Rel . When Rel is true (i.e., each threshold is computed relative to the examples delegated by the previous classifier), the approaches are called Global Relative Percentage and Stratified Relative Percentage, respectively; and when Rel is false, they are called Global Absolute Percentage and Stratified Absolute Percentage, respectively.

When classifying a new query instance q , the system first sends q to h_1 and produces an output for q based on one of several delegation mechanisms, generally taken from the following alternatives.

- Round-rebound (only applicable to two-stage delegation): h_1 defers to h_2 when its confidence is too low, but h_2 rebounds to h_1 when its own confidence is also too low.

- Iterative delegation: h_1 defers to h_2 , which in turn defers to h_3 , which in turn defers to h_4 , and so on until a model h_k is found whose confidence on q is above threshold or h_N is reached. The algorithm of Figure 5.13 implements this mechanism (lines 14 to 16).

Delegation may be viewed as a generalization of divide-and-conquer methods (e.g., see [98, 103]), with a number of advantages including:

- Improved efficiency: each classifier learns from a decreasing number of examples,
- No loss of comprehensibility: there is no combination of models; each instance is classified by a single classifier, and
- Possibility to simplify the overall multi-classifier: see for example the notion of grafting for decision trees [279].

5.4 Arbitrating

A mechanism for combining classifiers by way of arbitration, originally introduced as Model Applicability Induction, has been proposed by Ortega et al. [185, 186].⁵ As with delegating, the basic intuition behind arbitrating is that various classifiers have different areas of expertise (i.e., portions of the input space on which they perform well). However, unlike in delegating, where successive classifiers are specialized to instances for which previous classifiers lack confidence, all classifiers in arbitrating are trained on the full dataset T and specialization is performed at run time when a query instance is presented to the system. At that time, the classifier whose confidence is highest in the area of input space close to the query instance is selected to produce the classification. The process is illustrated in Figure 5.14.

The area of expertise of each classifier is learned by its corresponding referee. The referee, although it can be any learned model, is typically a decision tree which predicts whether the associated classifier is correct or incorrect on some subset of the data, and with what reliability. The features used in building the referee decision tree consists of at least the primitive attributes that define the base-level dataset, possibly augmented by computed features (e.g., activation values of internal nodes in a neural network, conditions at various nodes in a decision tree) known as internal propositions, which assist in diagnosing examples for which the base-level classifier is unreliable (see [186] for details). The basic idea is that a referee holds meta-information on the area of expertise of its associated classifier, and can thus tell when that classifier reliably predicts the outcome. Several classifiers are then combined through an arbitration mechanism, in which the final prediction is that of the classifier whose referee is the most reliably correct. The arbitrating algorithm is shown in Figure 5.15.

⁵ Interestingly, two other sets of researchers developed very similar arbitration mechanisms independently. See [153, 269].

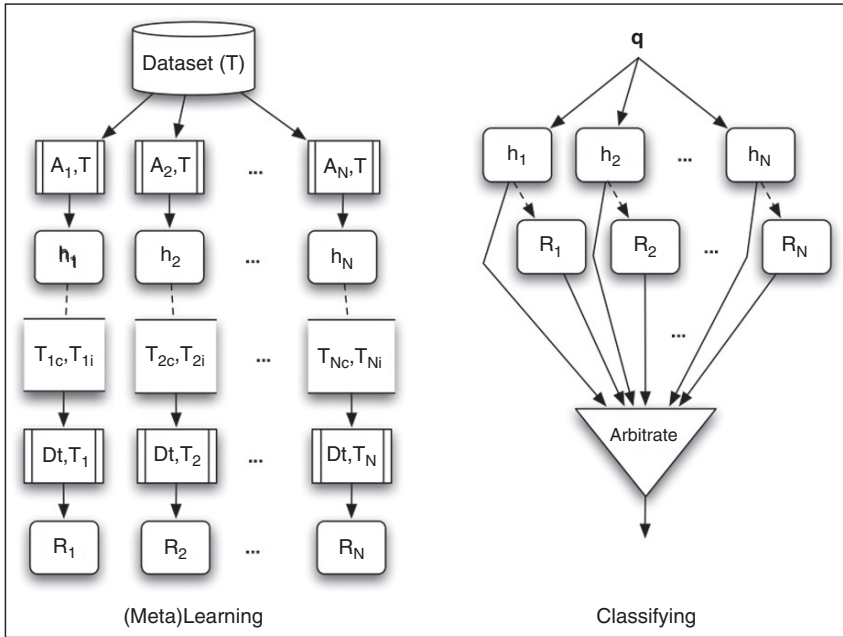


Fig. 5.14. Arbitrating

Interestingly, the neural network community has also proposed techniques that employ referee functions to arbitrate among the predictions generated by several classifiers. These are generally known as Mixture of Experts (e.g., see [131, 132, 278]).

Finally, note that a different approach to arbitration was proposed by Chan and Stolfo [58, 59], where there is generally a unique arbiter for the entire set of N base-level classifiers. The arbiter is just another classifier learned by some learning algorithm on training examples that cannot be reliably predicted by the set of base-level classifiers. A typical rule for selecting training examples for the arbiter is as follows: select example $\mathbf{e}$ if none of the target classes gather a majority vote (i.e., $> N/2$ votes) for $\mathbf{e}$. The final prediction for a query example is then generally given by a plurality of votes on the predictions of the base-level classifiers and the arbiter, with ties being broken by the arbiter. An extension, involving the notion of an arbiter tree is also discussed, where several arbiters are built recursively in a tree-like structure. In this case, when a query example is presented, its prediction propagates upward in the tree from the leaves (base learners) to the root, with arbitration taking place at each level along the way.

Algorithm Arbitrating($T, \{A_1, \dots, A_N\}$)

1. For $i = 1$ to N
2. $h_i =$ model induced by A_i from T
3. $R_i = \text{LearnReferee}(h_i, T)$
4. For each new query instance q
5. For $i = 1$ to N
6. $c_i =$ correctness of h_i on q as per R_i
7. $r_i =$ reliability of h_i on q as per R_i
8. $h^* = \text{argmax}_{h_i: c_i \text{ is 'correct'}} r_i$
9. $\text{Class}(q) = h^*(q)$

where:

T is the base-level training set

N is the number of base-level learning algorithms

$A_1, \dots, A_N$ are the base-level learning algorithms

$\text{LearnReferee}(A, T)$ returns a referee for learner A and dataset T

Function $\text{LearnReferee}(h, T)$

1. $T_c =$ examples in T correctly classified by h
2. $T_i =$ examples in T incorrectly classified by h
3. Select a set of features, including the attributes defining the examples and class, as well as additional features
4. $Dt =$ pruned decision tree induced from T
5. For each leaf L in Dt
6. $N_c(L) =$ number of examples in T_c classified to L
7. $N_i(L) =$ number of examples in T_i classified to L
8. $r = \frac{\max(|N_c(L), N_i(L)|)}{|N_c(L)| + |N_i(L)| + \frac{1}{2}}$
9. If $|N_c(L)| > |N_i(L)|$ Then
10. L 's correctness is 'correct'
11. Else
12. L 's correctness is 'incorrect'
13. Return Dt

Fig. 5.15. Arbitrating algorithm

5.5 Meta-decision Trees

Another approach to combining inductive models is found in the work of Todorovski and Dzeroski on meta-decision trees (MDTs) [264]. The general idea in MDT is similar to stacking in that a metamodel is induced from information obtained using the results of base-level learning, as shown in Figure 5.16. However, MDTs differ from stacking in the choice of what information to use, as well as in the metalearning task. In particular, MDTs build decision trees where each leaf node corresponds to a classifier rather than a classification. Hence, given a new query example, a meta-decision tree

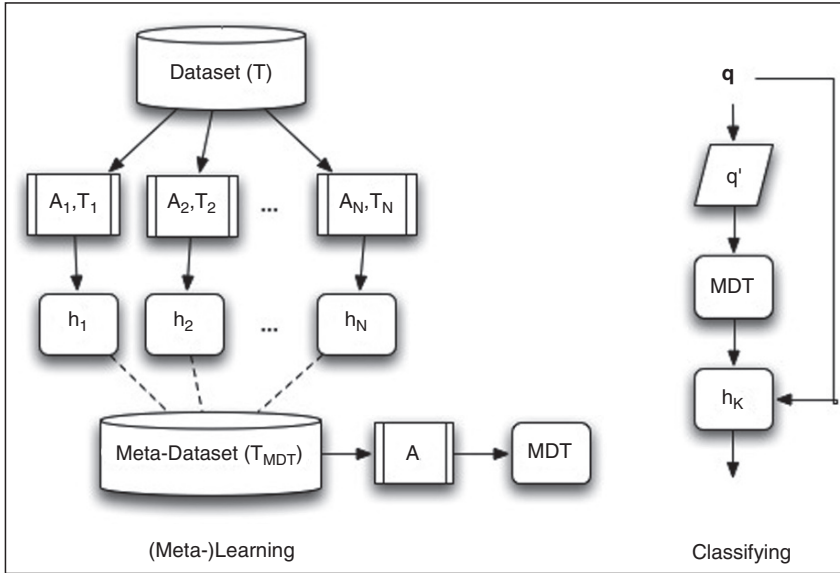


Fig. 5.16. Meta-decision tree

indicates the classifier that appears most suitable for predicting the example's class label. The MDT building algorithm is shown in Figure 5.17.

Class distribution properties are extracted from examples using the base-level learners on different subsets of the data (lines 7 to 9, Figure 5.17). These properties, in turn, become the attributes of the metalearning task. Unlike metalearning for algorithm selection where these attributes are extracted from complete datasets (and thus there is one meta-example per dataset), MDTs have one meta-example per base-level example, simply substituting the base-level attributes with the new computed properties. The metamodel MDT is induced from these meta-examples, T_{MDT} , with a metalearning algorithm $\mathcal{A}$. Typically, $\mathcal{A}$ is MLC4.5, an extension of the well-known C4.5 decision tree learning algorithm [197].

Interestingly, in addition to improving accuracy, MDTs, being comprehensible, also provide some insight about base-level learning. In some sense, each leaf of the MDT captures the relative area of expertise of one of the base-level learners (e.g., C4.5, LTree, CN2, k -NN and Naïve Bayes).

5.6 Discussion

The list of methods presented in this chapter is not intended to be exhaustive. Methods included have been selected because they represent classes of model combination approaches and are most closely connected to the subject of metalearning. A number of so-called *ensemble* methods have been proposed

```

Algorithm MDTBuilding( $T, \{A_1, \dots, A_N\}, m$ )
1.  $\{T_1, \dots, T_m\} = \text{StratifiedPartition}(T, m)$ 
2.  $T_{MDT} = \emptyset$ 
3. For  $i = 1$  to  $m$ 
4.   For  $j = 1$  to  $N$ 
5.      $h_j = \text{model induced by } A_j \text{ from } T - T_i$ 
6.     For each  $x \in T_i$ 
7.        $\text{maxprob}(x) = \max_{y \in \mathcal{Y}} P_{h_j}(y|x)$ 
8.        $\text{entropy}(x) = - \sum_{y \in \mathcal{Y}} P_{h_j}(y|x) \log P_{h_j}(y|x)$ 
9.        $\text{weight}(x) = \text{fraction of training examples used by } h_j \text{ to}$ 
         estimate the class distribution of  $x$ 
10.       $E_j(x) = \langle \text{maxprob}(x), \text{entropy}(x), \text{weight}(x) \rangle$ 
11.       $E_j = \cup_{x \in T_i} E_j(x)$ 
12.     $T_{MDT} = T_{MDT} \cup \text{join}_{j=1}^N E_j$ 
13.   $MDT = \text{model induced by MLC4.5 from } T_{MDT}$ 
14. Return  $MDT$ 
15. For each new query instance  $q$ 
16.   $\text{Class}(q) = MDT(\langle E_1(q), E_2(q), \dots, E_N(q) \rangle)$ 

```

where:

T is the base-level training set

N is the number of base-level learning algorithms

$A_1, \dots, A_N$ are the base-level learning algorithms

m is the number of disjoint subsets into which T is partitioned

$\text{StratifiedPartition}(T, m)$ returns a stratified partition of T into m equally-sized subsets

Fig. 5.17. Meta-decision tree building algorithm

that combine many algorithms into a single learning system (e.g., see [143, 184, 52, 46]). The interested reader is referred to the literature for descriptions and evaluations of other combination and ensemble methods.

Because it uses results at the base level to construct a classifier at the meta level, model combination may clearly be regarded as a form of metalearning. However, its motivation is generally rather different from that of traditional metalearning. Whereas metalearning explicitly attempts to derive knowledge about the learning process itself, model combination focuses almost exclusively on improving base-level accuracy. Although they do learn at the meta level, most model combination methods fail to produce any real generalizable insight about learning, except in the case of arbitrating and meta-decision trees where new metaknowledge is explicitly derived in the combination process. As stated in [277], “by learning or explaining what causes a learning system to be successful or not on a particular task or domain, [metalearning seeks to go] beyond the goal of producing more accurate learners to the additional goal of understanding the conditions (e.g., types of example distributions) under which a learning strategy is most appropriate.”

Bias Management in Time-Changing Data Streams

João Gama* and Gladys Castillo†

6.1 Introduction

The term *bias* has been widely used in machine learning and statistics with somewhat different meanings. In the context of machine learning, Mitchell [174] defines *bias* as *any basis for choosing one generalization over another, other than strict consistency with the instances*. In [112] the authors distinguish two major types of bias: representational and procedural. The former defines the states in a search space. It specifies the language used to represent generalizations of the examples. The latter determines the order of traversal of the states in the space defined by a representational bias. In statistics, *bias* is used in a somewhat different way. Given a learning problem, the *bias* of a learning algorithm is the persistent or systematic error the learning algorithm is expected to achieve when trained with different training sets of the same size.

To summarize, while machine learning bias refers to restrictions in the search space, statistics focuses on the error. Some authors [78, 149] have presented the so-called *bias-variance error decomposition* that gives insight into a unified view of both perspectives. Powerful representation languages explore larger spaces with a reduction on the bias component of the error (although by increasing the variance). Less powerful representation languages are correlated with large error due to a systematic error. Often, as one modifies some aspect of the learning algorithm, it will have an opposite effect on the bias and the variance. For example, usually as one increases the number of degrees of freedom in the algorithm, the bias error shrinks but the error due to variance increases. The optimal number of degrees of freedom (as far as expected loss is concerned) is that which optimizes this trade-off between bias and variance.

Bias

Bias-
variance
error
decompo-
sition

*LIAAD-INESC Porto L.A./Faculty of Economics, University of Porto, Rua de Ceuta 118, 6º, 4050-190 Porto, Portugal, jgama@liaad.up.pt.

†Department of Mathematics/CEOC, University of Aveiro, Campus Universitário de Santiago, 3810-193 Aveiro, Portugal, gladys@ua.pt.

This chapter is mainly concerned with the problem of bias management when there is a continuous flow of training examples, i.e., the number of training examples increases with time. A closely connected problem is that of concept drift in the machine learning community, where the target function generating data changes over time. Machine learning algorithms should strengthen bias management and concept drift management in such learning environments. Both aspects require some sort of control strategy over the learning process. In this chapter, methods for monitoring the evolution of some performance indicators are presented. Since the chosen indicators are based on estimates of the error, these controlling methods are classifier-independent and, as such, are related to *metalearning* and *learning to learn*.

The next section presents the basic concepts on learning from data streams and tracking time-changing concepts. Section 6.2 discusses the problem of dynamic bias selection and presents two examples of bias management learning algorithms: the *Very Fast Decision Tree* and the *Adaptive Prequential Learning Framework*. Section 6.3 summarizes this chapter and discusses general lessons learned.

6.2 Learning from Data Streams

In many applications, learning algorithms act in dynamic environments where the data flows continuously. If the process is not strictly stationary, which is the case of most real-world applications, the target concept can change over time. Nevertheless, most of the work in machine learning assumes that training examples are generated at random according to some stationary probability distribution. In the last two decades, machine learning research and practice have focused on batch learning usually from small datasets. In batch learning, the whole training data is available to the algorithm, which outputs a decision model after processing the data at least once and often multiple times. The rationale behind this practice is that training examples are independent and identically distributed. In order to induce the decision model, most learners use a greedy, hill-climbing search in the space of models. As pointed out by some researchers [35], those learners emphasize variance reduction.

What distinguishes many current data sets from earlier ones is automatic data feeds. We do not just have people entering information into a computer. Instead, we have computers sending data to each other. There are many applications today in which the data is best modeled not as persistent tables but rather as transient data streams. Examples of such applications include network monitoring, Web applications, sensor networks, telecommunications data management and financial applications. In these applications it is not feasible to load the incoming data into a traditional database management system, as those are not designed to support the requirement for continuous queries imposed by the applications [10].

Data mining offers several algorithms for these problems, and learning from data streams poses new challenges to data mining. In these situations the assumption that training examples are generated at random according to a stationary probability distribution will usually not hold. In complex systems and for large time periods, we should expect changes in the distribution of the examples. A natural approach for these *incremental tasks* consists of *adaptive learning algorithms*, that is, incremental learning algorithms that take into account concept drift. Domingos and Hulten [125] have proposed the following set of desirable properties for learning systems that are able to mine continuous, high-volume, open-ended data streams:

- Require small constant time per data example;
- Use fixed amount of main memory, irrespective of the total number of examples;
- Build a decision model using a single scan over the training data;
- Any time model;
- Independence from the order of the examples;
- Ability to deal with changes in the target concept. For stationary data, ability to produce decision models that are nearly identical to the ones we would obtain using batch learning.

Satisfying these properties requires new sampling and randomization techniques, as well as new approximate and incremental algorithms. Some data stream models allow delete and update operators. For these models, in the presence of context change, the incremental property is not sufficient, however. Learning algorithms need forgetting operators that discard outdated parts of the decision model, i.e., decremental unlearning [57].

An important concept throughout the work on change detection is that of *context*. A context is defined as a set of examples where the function generating the examples is stationary [113]. We can thus consider a data stream as a sequence of contexts. Changes between contexts can be gradual when there is a smooth transition between the distributions, or abrupt, when the distribution changes rapidly. The aim of this chapter is to present methods for detecting the several moments when there is a change of context. If we can identify contexts, we can identify which information is outdated and relearn the model only with information relevant to the present context.

6.2.1 Concept Drift

Work on *Statistical Quality Control* presents methods and algorithms for change detection [13, 113]. It is useful to distinguish between *off-line* algorithms and *online* algorithms. In both cases the objective of change detection is to detect whether there is a change in the sequence, and if so, when it happens. In the off-line case, the algorithm uses all the information about the sequence of values. In the online case, the algorithm processes the sequence one element at a time. The goal is to detect a change as soon as possible. Of

Change
detection
Concept
drift

course, online algorithms for change detection are in the framework of data streams. The most used online algorithms for change detection are: *Shewhart control charts*, *CUSUM*-type algorithms, and *GLR* detectors.

Sequential
analysis

Sequential Analysis is a way of solving hypothesis testing problems when the sample size is not fixed a priori, and depends on data that have been observed already [73]. Suppose we are receiving a sequence of observations (y_n) . Assume that the data is being generated at random according to some unknown distribution with parameters θ_0 . At a certain point in time, the parameters of the unknown distribution change to θ_1 . The problem is to detect that the distribution generating the data we are observing now is different from the one that was generating data before the parameter change. The main result of sequential analysis is the *sequential probability ratio test*. It can be used for testing between two alternative hypotheses $H_0 = \theta : \theta_0$ and $H_1 = \theta : \theta_1$. At time n we make one of the following decisions:

- accept H_0 when $S_n \leq -a$
- accept H_1 when $S_n \geq h$
- continue to observe and to test when $-a < S_n < h$

where $S_n = \ln \frac{p_{\theta_1}(y_1^n)}{p_{\theta_0}(Y_1^n)}$ and $-a, h$ are thresholds such that $-\infty < -a < h < \infty$.

Tracking Drifting Concepts

There are several methods in machine learning to deal with changing concepts [146, 145, 144, 282]. Drifting concepts are often handled by time windows or example weighting (according to age or utility). In general, approaches to cope with concept drift can be classified into two categories:

- Approaches that adapt a learner at regular intervals without considering whether changes have really occurred;
- Approaches that first detect concept changes, and next adapt the learner to these changes.

The example weighting approach is based on the simple idea that the importance of an example should decrease with time (implementations of this approach can be found in [146, 145, 157, 164, 282]). When a time window is used, at each time step the learner is induced only from the examples that are included in the window. Here, the key difficulty is how to select the appropriate window size: a small window can assure fast adaptability in phases with concept changes, but in more stable phases it can adversely affect the learner's performance. This is because a larger window would produce good and stable results in stable phases. On the other hand, it cannot react quickly to concept changes. In approaches where the aim is to first detect concept changes, some indicators (e.g., performance measures, properties of the data, etc.) are monitored through time (see [146] for a good classification of these indicators). If during the monitoring process a concept drift is detected, some actions to adapt the learner to these changes can be taken. When a time

window of adaptive size is used these actions usually lead to adjusting the window size according to the extent of concept drift [146]. As a general rule, if a concept drift is detected the window size decreases, otherwise the window size increases. An implementation of this approach is the FLORA family of algorithms developed by Widmer and Kubat [282]. For instance, FLORA2 includes a window adjustment heuristic for a rule-based classifier. To detect concept changes the accuracy and the coverage of the current learner are monitored over time and the window size is adapted accordingly.

Other relevant works in this area include the works of Klinkenberg and Lanquillon. For instance, Klinkenberg et al. [146] proposed monitoring the values of three performance indicators, namely accuracy, recall, and precision, over time, and then comparing them to a confidence interval of standard sample errors for a moving average value (using the last M batches) of each particular indicator. Although these heuristics seem to work well in their particular domain, they have to deal with two main problems: *i*) computing performance measures requires user feedback about the true class; in some real applications only partial user feedback is available; and *ii*) a considerable number of parameters need to be tuned. In a subsequent work Klinkenberg and Joachims [145] presented a theoretically well-founded method to recognize and handle concept changes using support vector machines. The key idea is to select the window size so that the estimated generalization error on new examples is minimized. This approach uses unlabeled data to reduce the need for labeled data, and it does not require complicated parameterization. In section 6.3.2 we discuss a method based on Statical Process Control to monitor the evolution of the learning process, detecting changes in the evolution of the error rate.

6.3 Dynamic Bias Selection

The problem of dynamic bias selection comes from the observation that each learning algorithm has a selective superiority: each is best for some, but not all tasks. Each learning algorithm searches within a restricted generalization space, defined by its representation language, and employs a search bias for selecting a generalization in that space. Given a data set, it is often not clear a priori which representation language is most appropriate for the corresponding problem.

Bias
selection

In the context of batch learning, where the available training data is finite and static, several bias selection methods have been proposed. Methods like selection by cross-validation [214], Stacked Generalization [284] and Model Class Selection System (MCS) [45] are discussed elsewhere in this book. Another related method is the *Cascade-correlation architecture* [94] to train neural networks. It is a generative, feed-forward learning algorithm for artificial neural networks that is able to incrementally add new hidden units to improve its generalization ability. For each new hidden unit, the algorithm tries

to maximize the magnitude of the correlation between the new unit's output and the residual error signal of the net.

We should point out that most heuristic knowledge about the characteristics that indicate one bias is better than another incorporates the number of training examples as a key characteristic (see for example the heuristic rules in MCS). Few works consider bias selection in the context of dynamic training sets, where the number of training examples varies through time.

The next two sections briefly describe illustrative bias management systems. The first one is the *Very Fast Decision Tree* algorithm. The second one is an adaptive algorithm to learn Bayesian Network Classifiers. The Bayesian Network framework provides a stratified family of models, where each stratum allows for higher complexity. In both algorithms, the main issue is the trade-off between the costs of model adaptation and the gain in performance.

6.3.1 The *Very Fast Decision Tree* Algorithm

Learning from large datasets may be more effective when using algorithms that place greater emphasis on bias management. One such algorithm is the VFDT [124] system. VFDT is a decision tree learning algorithm that dynamically adjusts its bias whenever new examples are available. In decision tree induction, the main issue is the decision of when to expand the tree, installing a splitting-test and generating new leaves. The basic idea of VFDT consists of using a small set of examples to select the splitting-test. If after seeing a set of examples, the difference of the merit between the two best splitting-tests does not satisfy a statistical test (the Hoeffding bound), VFDT proceeds by examining more examples. VFDT only makes a decision (i.e., adds a splitting-test in that node), when there is enough statistical evidence in favor of a particular test. This strategy guarantees model stability (low variance), controls overfitting, while it may achieve an increased number of degrees of freedom (low bias) with an increasing number of examples.

In VFDT a decision tree is learned by recursively replacing leaves with decision nodes. Each leaf stores the sufficient statistics about attribute-values. The sufficient statistics are those needed by a heuristic evaluation function that computes the merit of split-tests based on attribute-values. When an example is available, it traverses the tree from the root to a leaf, evaluating the appropriate attribute at each node, and following the branch corresponding to the attribute's value in the example. When the example reaches a leaf, the sufficient statistics are updated. Then, each possible condition based on attribute-values is evaluated. If there is enough statistical support in favor of one test over the others, the leaf is changed to a decision node. The new decision node will have as many descendant leaves as the number of possible values for the chosen attribute (therefore this tree is not necessarily binary). The decision nodes only maintain the information about the split-test installed within them.

input : S : A Sequence of examples
 X : A Set of nominal attributes
 $Y: Y = \{y_1, \dots, y_k\}$ set of class values
 $H(\cdot)$: Split evaluation function
 δ : 1 minus the desired probability of choosing the correct attribute.
 τ : Constant used to break ties.
output: HT : A decision tree
begin
 Let $HT \leftarrow$ Empty Leaf (Root)
 foreach $example (x, y_k) \in S$ **do**
 Traverse the tree HT from the root to a leaf l
 Update sufficient statistics at l
 if *all examples in l are not of the same class* **then**
 Compute $G_l(X_i)$ for all the attributes
 Let X_a be the attribute with highest H_l
 Let X_b be the attribute with second highest H_l
 Compute ϵ (Hoeffding bound)
 if $(H(X_a) - H(X_b) > \epsilon)$ **or** $\epsilon < \tau$ **then**
 Replace l with a splitting test based on attribute X_a
 Add a new empty leaf for each branch of the split
 end
 end
 end
end

Algorithm 6.1: The Hoeffding tree algorithm

The main innovation of the VFDT system is the use of Hoeffding bounds to decide how many examples must be observed before installing a split-test at a leaf. Suppose we have made n independent observations of a random variable r whose range is R . The Hoeffding bound states that, with probability $1 - \delta$, the true mean of r is in the range $\bar{r} \pm \epsilon$ where $\epsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n}}$ and $\bar{r}$ is the sample mean.

Let $H(\cdot)$ be the evaluation function of an attribute. For the information gain, the range R of $H(\cdot)$ is $\log_2(k)$, where k denotes the number of classes. Let x_a be the attribute with the highest $H(\cdot)$, x_b be the attribute with second-highest $H(\cdot)$ and $\Delta \bar{H} = \bar{H}(x_a) - \bar{H}(x_b)$, the difference between the two best attributes. Then if $\Delta \bar{H} > \epsilon$ with n examples observed in the leaf, the Hoeffding bound states that, with probability $1 - \delta$, x_a is really the attribute with the highest value in the evaluation function. In this case the leaf must be transformed into a decision node that splits on x_a .

The evaluation of the merit function for each example could be very expensive. It turns out that it is not efficient to compute $H(\cdot)$ every time an example arrives. VFDT only computes the attribute evaluation function $H(\cdot)$ when a minimum number of examples have been observed since the last evaluation.

This minimum number of examples is a user-defined parameter. When two or more attributes continuously have very similar values of $H(\cdot)$, even with a large number of examples, the Hoeffding bound will not decide between them. To solve this problem the VFDT uses a constant τ introduced by the user for a runoff, e.g., if $\overline{\Delta H} < \epsilon < \tau$ then the leaf is transformed into a decision node. The split test is based on the best attribute.

Later, the same authors presented the CVFDT algorithm [126], an extension to VFDT designed for time-changing data streams. CVFDT generates alternative decision trees at nodes where there is evidence that the splitting test is no longer appropriate. The system replaces the old tree with the new one when the latter becomes more accurate.

6.3.2 The case of Bayesian Network Classifiers

The k -Dependence Bayesian Classifiers (DBC) are a stratified family of decision models with increasing (smooth) complexity. In this framework all of the attributes depend on the class, and any attribute depends on k other attributes, at most. The value of k can vary from 0 to a maximum of the number of attributes. The 0-DBC corresponds to considering all variables as independent and is usually referred to as the *Naïve Bayes* classifier. At the other end of the spectrum, each variable is influenced by all the others. This family of models is better described as consisting of a direct acyclic graph that defines the dependencies between variables, and a set of parameters (the conditional probability tables) that codify the conditional dependencies. Increasing the number of dependencies among attributes requires the estimation of an increased number of parameters.

Assume that data is available to the learning system sequentially. The actual decision model must first make a prediction and then update the current model with new data. This philosophy about online learning frameworks has been exposed by Dawid in his predictive-sequential approach, referred to as *prequential* [73] for statistical validation of models. An efficient adaptive algorithm in a *prequential learning framework* must be able, above all, to improve its predictive accuracy over time while reducing the cost of adaptation. However, in many real-world situations it may be difficult to improve and adapt to existing changing environments. As we have mentioned previously, this problem is known as concept drift. In changing environments, learning algorithms should be provided with some control and adaptive mechanisms that quickly adjust the decision model to these changes.

The Naïve Bayes classifier (NB) is one of the most widely used classifiers in real-world online applications, mainly due to its effectiveness, simplicity and incremental nature. NB simplifies learning by assuming that attributes are independent given the class. However, in practice, the independence assumption is often violated, which can lead to poor predictive performance. We can improve NB if we tradeoff *bias reduction*, which leads to the addition of new attribute dependencies, and, consequently, to the estimation of more

parameters, with *variance reduction*, by accurately estimating the parameters. Different classes of Bayesian Network Classifiers (BNCs) [102] attempt to reduce the bias of the NB algorithm by adding attribute dependencies to the NB structure. Nevertheless, not always do the more complex BNCs outperform the NB. Increasing complexity decreases bias but increases the variance in the parameters. These issues are still more challenging in a *sequential framework*, where the training data increases with time. In this case, we should adjust the complexity of BNCs to suit the available data. The main problem is to handle the trade-off between the cost of updating the decision model and the gain in performance. Possible strategies for incorporating new data are *bias management* and *gradual adaptation*. The motivation for bias control, along with some results of its application, was first presented in [54]. Another issue that should be addressed is that of coping with concept drift. As new data is available over time the target function generating data can change. The same techniques that monitor the evolution of the error can be used to detect drift in the concepts to learn [55].

The model class of k -Dependence Bayesian Classifiers (k -DBC) [211] is very suitable to illustrate this approach. A k -DBC is a Bayesian Network, which contains the structure of the NB and allows each attribute to have a maximum of k attribute nodes as parents. By increasing k we can obtain classifiers that move smoothly along the spectrum of attribute dependencies. For instance, NB is a 0-DBC, and TAN [102] is a 1-DBC. Instead of using the learning algorithm proposed in [211] based on the computation of the conditional mutual information, it is possible to use a *hill-climbing* procedure due to the obvious simplicity of its computational implementation. The algorithm builds a k -DBC starting with an NB structure. Then it iteratively adds arcs between two attributes that result in the maximal improvements in a given score until there are no more improvements for that score or until it is not possible to add a new arc.

Figure 6.1 shows an example of the search space explored by the proposed algorithm. The initial state is a 0-DBC. For this model class only one structure can be explored. For a fixed k ($k > 0$) several different structures can be explored. As we increase the number of allowed dependencies the number of parameters needed to be estimated increases exponentially.

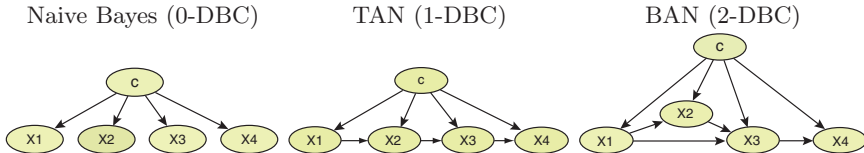


Fig. 6.1. Example of the space of increasing dependencies. Considering all variables are binary the number of parameters are 18, 30, and 38 respectively

To clearly illustrate the increased number of dependencies considered by each model, we present the factorization of the a posteriori probability of each model presented in Figure 6.1:

$$\begin{aligned} 0\text{-DBC: } & P(C)P(x_1|C)P(x_2|C)P(x_3|C)P(x_4|C) \\ 1\text{-DBC: } & P(C)P(x_1|C)P(x_2|x_1, C)P(x_3|x_2, C)P(x_4|x_3, C) \\ 2\text{-DBC: } & P(C)P(x_1|C)P(x_2|x_1, C)P(x_3|x_1, x_2, C)P(x_4|x_3, C) \end{aligned}$$

The Adaptive Prequential Learning Framework

The main assumption that drives the design of the **AdPreqFr4SL** [55, 53] is that observations do not arrive at the learning system at the same point in time. Typically the environment will change over time. Without loss of generality, one can assume that at each time-point data arrives in batches. The main goal is to predict the target classes of the next batch. Many adaptive systems make regular updates while new data arrives. The **AdPreqFr4SL**, instead, is provided with some control mechanisms that attempt to select the best adaptive actions based on the current learning goal. To this end, for each batch of examples the current hypothesis is used for prediction. The actual, correct class is then observed and performance *indicators* are assessed. The indicator values are used to estimate the current system's *state*. Finally, the model is adapted according to the estimated state.

Two performance indicators are monitored over time: the *batch error* Err_B (the proportion of misclassified examples in one batch) and the *model error* Err_S (the proportion of misclassified examples in the complete set of examples classified using the same structure). They are used, in turn, to estimate one of the following states:

SI - IS IMPROVING: performance is improving;
 SS - STOP IMPROVING: performance stops improving at a desirable rate;
 SA - CONCEPT DRIFT ALERT: first alert of concept drift;
 SD - CONCEPT DRIFT: presence of a gradual concept change;
 SCs- CONCEPT SHIFT: presence of an abrupt concept change;
 SP - STABLE PERFORMANCE: performance reaches a plateau.

The following subsections present the adaptive actions and control strategies adopted in the **AdPreqFr4SL** for handling the *cost-performance* trade-off and concept drift.

Cost-Performance Management.

The adaptation strategy for handling *cost-performance* is based upon two main policies:

- *bias management,*
- *gradual adaptation.*

The former policy starts with a 0-DBC, or NB, structure. The model complexity is scaled up by gradually increasing k and searching for new attribute dependencies in the resulting search space. The gradual adaptation policy works as follows (see Figure 6.2): In the INITIAL LEVEL a new model is built using a simple NB. In the FIRST LEVEL only the parameters are updated using new data [105]. In the SECOND LEVEL the structure is updated with new data. In the THIRD LEVEL, if it is still possible, k is increased by one, and the current structure is once again adapted.

The k -DBC is initialized to the simplest model: NB ($k = 0$). Whenever new data arrives, only the parameters of the NB are updated. When there is evidence indicating that the performance of the NB stops improving, the system starts adapting the structure. Only in this case (for $k = 0$) can the system move from the *first level* to the *third level*¹ of adaptation: increment k by 1 and start searching a 1-DBC using the *hill-climbing* search procedure with only arc additions. At this time-point, more data must be available to allow the

```

input  : A classifier  $h_C = (S, \Theta_S)$  belonging to the class of  $k$ -DBC's, A batch
           $B$  of  $m$  examples
          The level of adaptation
          The current  $k$  value
          The value kMax of the maximum allowable  $k$ 
output: An adaptive action over the classifier  $h_C$ 
begin
  if INITIAL level then
     $k \leftarrow 0$  /* build a new model using NB*/
    learnNaiveBayes(SHORT-MEMORY)
  end
  else if FIRST level then
    updateParameters( $h_C, B$ )
  end
  else if SECOND level then
    updateStructure( $h_C, B, \dots$ )
  end
  else if THIRD level then
    if  $k < \mathbf{kMax}$  then
       $k+ = 1$ 
    end
    updateStructure( $h_C, B, \dots$ )
  end
end

```

Algorithm 6.2: Adaptive actions for the class of k -DBC's in AdPreqFr4SL

¹ In the case of 0-DBC there is only one structure modeling dependencies between attributes. In all the other cases, for a fixed k ($k > 0$) there are several possible structures.

search procedure to find new 1-dependencies. Next, the algorithm continues to perform only parameter adaptation [105]. Thus, whenever a new structure is found, the algorithm continues working from the *first level* of adaptation, that is, by performing only parameter adaptation, until there is again evidence that the performance of the current hypothesis has stopped improving; and this moves the algorithm to the *second level*: update the current structure by searching for new attribute dependencies. At this stage and to correct previous errors, the search procedure is also allowed to perform *arc deletions*. Only if the resulting structure remains the same, does the algorithm move to the *third level* of adaptation by incrementing k by 1 and continuing searching for new dependencies, now in an augmented search space. To prevent k from increasing unnecessarily, the old value of k is recovered whenever the search procedure is not able to find new dependencies, thus keeping the original search space. Only if an abrupt concept drift is detected does the algorithm come back to the *initial level* and build a new NB using the examples from a short-term memory (see next section). This adaptation process continues until it is detected that it makes no sense to continue adapting the model. However, the algorithm will continue monitoring performance. If any significant change in the behavior is observed, then the algorithm will once again activate the adaptation procedures.

The *control policy* defines the criteria for tracking two situations:

- At what point in time does structure adaptation start?
- At what point in time does adaptation stop?

If it is detected that the performance of the current model no longer improves (the state **SS**), structure adaptation begins. If it is detected that the performance reaches a plateau (the state **SP**), adaptations to the model stop. To detect the states **SS** and **SP**, we plot the values of successive model errors, $y(t) = Err_S^{(t)}$, in time and connect them by a line, thus obtaining the *model-error learning curve* (**model-LC**). The state **SS** is met if *i*) the **model-LC** starts *behaving well* [47], i.e., the curve is *convex* and *monotonically non increasing* for a given number of points; and *ii*) its slope is *gentle*. Thus, whenever a new structure is used the adaptive algorithm will wait until the **model-LC** starts *behaving well* and shows only little improvements in the performance in order to trigger a new structure adaptation. Only when the structure does not change after adaptation is the **model-LC** once again analyzed in order to detect whether it has already reached its *plateau* (i.e., **SP** is signaled).

Figure 6.2 illustrates the behavior of the **model-LC** for one randomly generated sample of the *Adult dataset* using batches of 100 examples. To serve as a baseline, the graph also shows the error rates obtained with NB and with a 3-DBC (the class model with best performance) induced from scratch at each learning step. During all the learning process the structure changed only five times. The graphical behavior of the model error neatly corresponds to the detected conditions which lead to a structure adaptation action. The k value

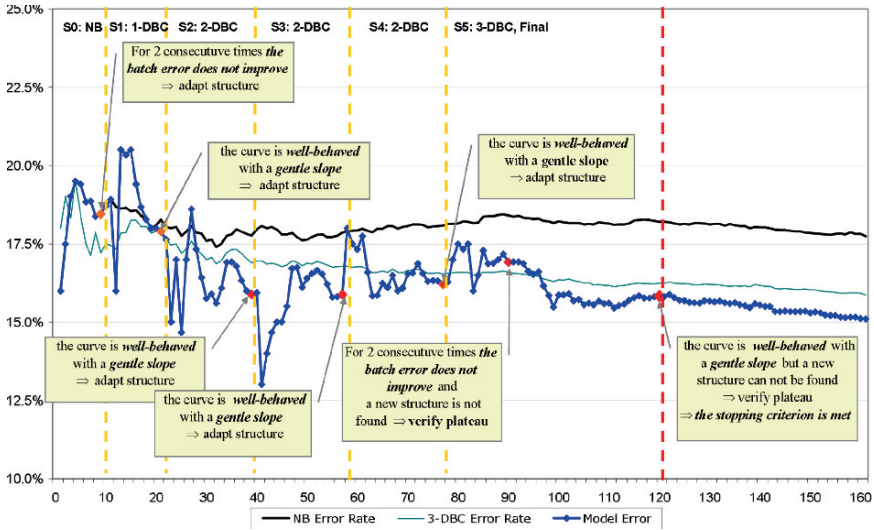


Fig. 6.2. Behavior of the model-LC for the adaptive algorithm. Vertical lines indicate the time-points at which the structure changed. On top, the resulting structures with their corresponding k -DBC class models are presented

slowly increases from 0 to 3 until the stopping criterion is met at $t = 120$ and the model is not further adapted with new data.

Using the P-Chart for Handling Concept Drift

Concept drift refers to unforeseen changes in the distribution underlying the data that can also lead to changes in the target concept over time [282]. Several available concept drift trackers employ different approaches that include some control strategies in order to decide whether adaptation is really necessary because a concept change has occurred. To this end, a process that monitors the value of some performance indicators must be implemented. If a concept drift is detected, some actions to adapt the model to these changes are taken, which usually lead to building a new model. Some concept drift trackers are also capable of recognizing the *extent* of concept drift. The term *concept drift* is more often associated with gradual changes whereas the term *concept shift* defines abrupt changes.

In [56] the authors present a method for handling concept drift based on a *Shewhart P-Chart* [113], an attribute control chart that monitors the proportion of a *dichotomous* count variable. This method for handling concept drift is integrated with the method for bias management described in the previous section into the unified framework AdPreqFr4SL. The basic idea consists of using the P-Chart for monitoring the batch error Err_B . The values $p(t) = Err_B^{(t)}$ are plotted on the chart over time and connected by a line.

Concept
drift
Statistical
process
control

The chart has a *center line* (CL), an *upper control limit* (UCL) and an *upper warning limit* (UWL). If the sample sizes are large (≥ 30) the *sample proportion* approaches the Normal distribution with parameters $\mu = p$; $\sigma = \sqrt{p(1-p)/n}$ (p is the population proportion). Therefore, the use of *three-sigma control* limits is a reasonable choice. Suppose that an estimate $\hat{p}$ is obtained from previous data. We can obtain the P-Chart's lines as follows: $CL = \hat{p}$; $UCL = \hat{p} + 3\sigma$; $UWL = \hat{p} + \alpha\sigma, 0 < \alpha < 3$. The usual value for α is 2. To better follow the natural behavior of the learning process, the target value $\hat{p}$ is set to the minimum value of the current model error Err_S , denoted by Err_{min} . Whenever a new structure S is found, Err_{min} is initialized to some large number. Then, at each time step, if $Err_S^{(t)} + SErr_S^{(t)} < Err_{min}$ then Err_{min} is set to $Err_S^{(t)}$, where $SErr_S^{(t)}$ is its standard deviation.

Thus, at each time t , $\hat{p}$ is set to Err_{min} and the P-Chart's lines are computed accordingly. Then, one can observe where the new proportion $p(t) = Err_B^{(t)}$ falls on the P-Chart. If $p(t)$ falls above the UCL, a *concept shift* is signaled. If $p(t)$ falls between the UCL and the UWL for the first time, then a *concept drift alert* is signaled. Otherwise, if this situation occurs for two or more consecutive times then a *concept drift* is detected. If $p(t)$ falls under UWL we assume that the learner is *in control* and then proceed to analyze the behavior of the model-LC as described in the previous section.

The *adaptive strategy* for handling concept drift mainly consists of manipulating a *short-term memory* (SHORT-MEMORY) to store those examples that are suspected to belong to a new concept. If a *concept shift* is detected then all the examples from the SHORT-MEMORY are used to build a new NB classifier. Afterwards, the SHORT-MEMORY is cleaned for future use. Whenever a *concept drift alert* or *concept drift* is signaled, the examples of the current batch are added to the SHORT-MEMORY. However, after signaling a *concept drift*, the new examples are not used to update the model in order to force a greater degradation of the performance. This way the P-Chart will be able to recognize a concept shift more quickly and re-build the model. Algorithm 6.3 contains the pseudo-code of the whole algorithm for learning k -DBC's in the AdPreqFr4SL framework. It summarizes all of the aforementioned strategies for handling *cost-performance* and *concept drift*.

Figures 6.3 and 6.4 illustrate the dynamics of the adaptive and control strategies. In the *first drift phase* (between $t = 37$ and $t = 43$) the P-Chart detected two concept shifts and a new NB was built using the examples of the current batch. In the *second drift phase* (between $t = 77$ and $t = 83$) almost all the points fell above the UWL but very close to the UCL. The P-Chart signaled concept drift and the adaptation process was temporarily stopped to force the Err_B to jump outside the UCL. Later, at $t = 83$, when a concept shift was detected, all the examples stored in the SHORT-MEMORY were used to build a new NB. For the remaining drift phases the detection method using P-Chart also worked as expected. In this scenario, the structure was rebuilt five times, at points in time that belong to the drift phases. Note that

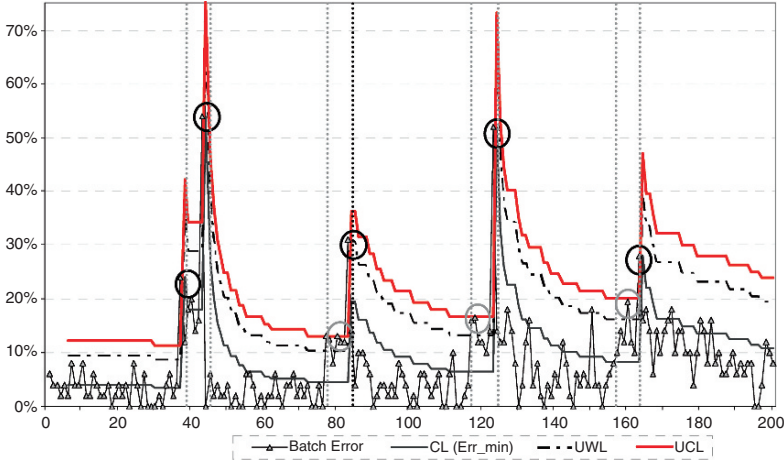


Fig. 6.3. The P-Chart for a generated CSS. Parallel light-grey dotted lines on the P-Chart indicate the beginning and the end of each drift phase

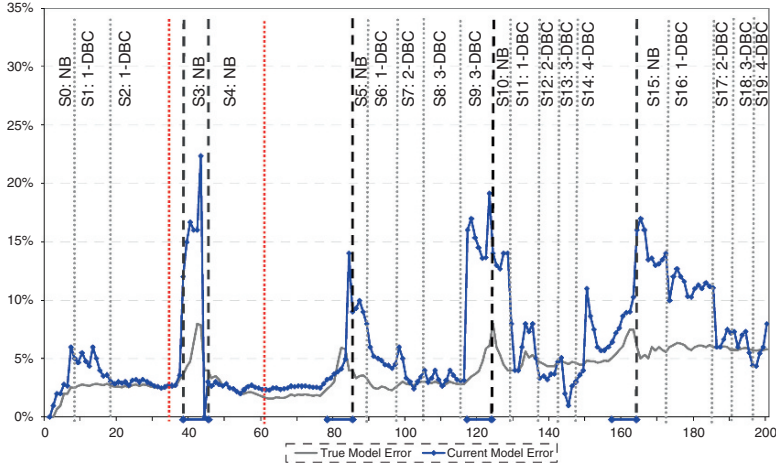


Fig. 6.4. The model error $Errs$ for a generated CSS. Vertical light-grey dotted lines and black dashed lines indicate the times at which the current structure was adapted or rebuilt, respectively. Vertical dark-grey dotted lines indicate the times at which the adaptation process was stopped. At the top, the resulting structures with their corresponding k -DBC class models are presented

the complexity of the induced k -DBCs increased from context to context: in the first context the resulting k -DBC is a 1-DBC, in the third, a 3-DBC, in the fourth, a 4-DBC, and in the last, a 4-DBC too (searching for more complex

structures can require more training data). Only in the second context was the NB structure not modified since the adaptation process was stopped early. However, the model error showed a good behavior in this context.

6.4 Lessons Learned and Open Issues

Throughout this chapter, the object under study has been the dynamics of the learning process. We discuss general strategies for reasoning about the evolution of the learning process itself. What makes today's learning problems different from earlier ones is the large volume and continuous flow of data. These characteristics impose new constraints on the design of learning algorithms. Large volumes of data require efficient bias management, while the continuous flow of data requires change detection algorithms to be embedded in the learning process.

The main research issue is the trade-off between the cost of update and the gain in performance that may be obtained. Learning algorithms exhibit different profiles. Algorithms with strong variance management are quite efficient for small training sets. Very simple models, using few free parameters, can be quite efficient in variance management, and effective in incremental and decremental operations (for example, naive Bayes), for which a natural choice is the sliding windows framework. The main problem with simple approaches, however, is the bound in generalization performance they can achieve, since they are limited by high bias. Large volumes of data require efficient bias management. Complex tasks requiring more complex models increase the search space and the cost for structural updating. These models require efficient control strategies for the trade-off between the gain in performance and the cost of updating.

input : A dataset $\mathcal{D}$ divided in batches of m examples
 The value k_{Max} of the maximum allowable k
 A scoring function $\text{Score}(S, \mathcal{D})$
 The number maxTimes of consecutive times that Err_B does not decrease after parameter adaptation

output: A classifier $h_C = (S, \Theta_S)$ belonging to the class of k -DBC's

begin

```

/*build a NB classifier, see Alg. 6.2*/
AdaptiveAction( $h_C$ , SHORT-MEMORY, INITIAL LEVEL)
foreach batch  $B$  of  $m$  examples of  $\mathcal{D}$  do
  predictions  $\leftarrow$  predict( $B, h_C$ )
  /*get feedback */
  observed  $\leftarrow$  getFeedback( $B$ )
  /* asses current indicators*/
   $p(t) \leftarrow \text{Err}_B^{(t)}, y(t) \leftarrow \text{Err}_S^{(t)}$ 
  Add ( $t, y(t)$ ) to model-LC
  /*concept drift detection using the P-Chart*/
  state  $\leftarrow$  getState( $p(t)$ , P-Chart)
  if state is CONCEPT SHIFT then
    Add  $B$  to SHORT-MEMORY
    /*build a NB classifier, see Alg. 6.2*/
    AdaptiveAction( $h_C$ , SHORT-MEMORY, INITIAL LEVEL)
    Clean SHORT-MEMORY
    else if state is CONCEPT DRIFT ALERT  $\vee$  CONCEPT DRIFT then
      Add  $B$  to SHORT-MEMORY
    else
      Clean SHORT-MEMORY
      /* state is IN CONTROL then observe the model-LC*/
      if model-LC is Convex-NonIncreasing-with-GentleSlope then
        state  $\leftarrow$  STOPS IMPROVING
      end
      else
        state  $\leftarrow$  IS IMPROVING
      end
    end
  end
end
if state IS IMPROVING  $\vee$  CONCEPT DRIFT ALERT then
  /* update parameters */
  AdaptiveAction( $h_C, B$ , FIRST LEVEL)
  if  $\text{consecCounter}(\text{Err}_B^{t_{\text{AFTER-ADAP}}} \geq \text{Err}_B^{t_{\text{BEF-ADAP}}}) = \text{maxTimes}$  then
    state  $\leftarrow$  STOP IMPROVING
  end
end
if state STOPS IMPROVING then
  if  $k > 0$  then
    /* update structure */
    AdaptiveAction( $k$ -DBC,  $B$ , SECOND LEVEL, ...)
  end
  if ( $\text{not change}(S) \wedge k < \text{Maxk}$ )  $\vee k = 0$  then
    /*increment  $k$ ; continue searching */
    AdaptiveAction( $h_C, B$ , THIRD LEVEL,  $k, \dots$ )
  end
  if  $\text{not change}(S)$  then
    /* verify the stopping criterion */
    if model-LC Has-Plateau then
      stopAdapting  $\leftarrow$  TRUE;
      state  $\leftarrow$  STABLE PERFORMANCE
    end
  end
end
end
end
return ( $h_C$ )
end

```

Algorithm 6.3: The algorithm for learning k -DBC's in AdPreqFr4SL

Transfer of Metaknowledge Across Tasks

7.1 Introduction

We have mentioned before that learning should not be viewed as an isolated task that starts from scratch with every new problem. Instead, a learning algorithm should exhibit the ability to adapt through a mechanism dedicated to transfer knowledge gathered from previous experience [258, 254, 206, 50]. The problem of transfer of metaknowledge is central to the field of *learning to learn* and is also known as *inductive transfer*. In this case, metaknowledge can be understood as a collection of patterns observed across tasks. One view of the nature of patterns across tasks is that of invariant transformations. For example, image recognition of a target object is simplified if the object is invariant under rotation, translation, scaling, etc. A learning system should be able to recognize a target object on an image even if previous images show the object in different sizes or from different angles. Hence, learning to learn studies how to improve learning by detecting, extracting, and exploiting metaknowledge in the form of invariant transformations across tasks.

Learning to
learn
Inductive
transfer

In this chapter we take a look at various attempts to transfer metaknowledge across tasks. In its most common form, the process of inductive transfer maintains the learning algorithm unchanged (Sections 7.2.1–7.2.4), but the literature also presents more complex scenarios where the learning architecture itself evolves with experience according to a set of rules (Section 7.2.5). We present recent developments on the theoretical aspects of learning to learn (Section 7.3). We end our chapter by looking at practical challenges in knowledge transfer (Section 7.4).

7.2 Learning to Learn

In learning to learn, we expect a continuous learner to extract knowledge across domains or tasks to accelerate the rate of learning convergence [276]. In inductive learning, this calls for the ability to incorporate metaknowledge

into the new learning task. We review a variety of different techniques on how to transfer metaknowledge across tasks with an emphasis on inductive learning; other work can be found in fields such as reinforcement learning [115, 76, 200, 6] (mentioned briefly in Section 7.4.3) and Bayesian networks [182].

Many experiments in inductive transfer have been reported within the neural network community (Section 7.2.1), but other architectures have also played an important role. Besides neural networks, this section includes kernel methods (Section 7.2.2), parametric Bayesian methods (Section 7.2.3), and other methods (Section 7.2.4), including latent models, feature mapping, and clustering.

7.2.1 Transfer in Neural Networks

A learning paradigm amenable to testing the feasibility of knowledge transfer is that of neural networks. A nonlinear multi-layer network is capable of expressing flexible decision boundaries over the input space [82]; it is a nonlinear statistical model that applies to both regression and classification [114]. In particular, for a neural network with one hidden layer, each output node computes the following function:

$$g_k(\mathbf{x}) = f\left(\sum_j w_{kj} f\left(\sum_i w_{ji}x_i + w_{j0}\right) + w_{k0}\right) \quad (7.1)$$

where $\mathbf{x}$ is the input parameter vector, $f(\cdot)$ is a nonlinear (i.e., sigmoid) function, and x_i is a component of vector $\mathbf{x}$. Index i runs along the components of vector $\mathbf{x}$, index j runs along a number of intermediate functions (i.e., nonlinear transformations of the input features), and index k refers to the k th output node. The output is a nonlinear transformation of the intermediate functions. The learning process is limited to finding appropriate values for all weights $\{w\}$ [114]. Neural networks have received much attention in the context of knowledge transfer because one can exploit the final set of weights of the *source* network (i.e., of the network obtained on a previous task) to initialize the set of weights corresponding to the *target* network (i.e., to the network corresponding to the current task).

Before we proceed to review previous work in this area, we introduce relevant terminology (following Pratt and Jennings [192]). We use the term *representational transfer* [14] to denote the case when the target and source network are trained at different times and the transfer takes place after the source network has already been trained; in this case there is an explicit form of knowledge transferred into the target network. In contrast, we use the term *functional transfer* to denote the case where two or more networks are trained simultaneously [226]; in this case the networks share (part of) their internal structure during learning. When the transfer of knowledge is explicit, as is the case with representational transfer, a further distinction is made. We denote as *literal* transfer the case when the source network is left intact (e.g.,

Source
network
Target
network

Represent-
ational
transfer

Functional
transfer

Literal
transfer

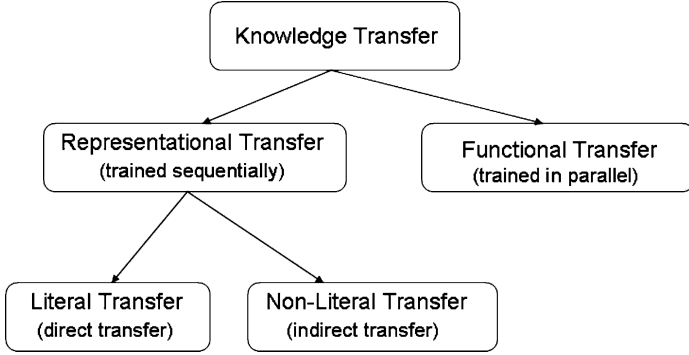


Fig. 7.1. Different forms of knowledge transfer in neural networks

when the final set of weights of the source network are directly used as initial weights for the target network). In addition, we denote as *non-literal* transfer the case when the source network is modified before knowledge is transferred to the target network; in this case some processing step is effected on the network before it is used to initialize the target network. Figure 7.1 illustrates different forms of knowledge transfer in neural networks.¹

A popular form of knowledge transfer is done within the functional transfer approach. *Multitask* learning takes place when the output nodes in the multi-layer network $\{g_k(\mathbf{x})\}$ represent more than one task (as proposed by Caruana [50, 51]). In such scenarios internal nodes are shared by different tasks dynamically during learning. As an illustration, consider the problem of learning to classify astronomical objects from images mapping the sky into multiple classes. One task may be in charge of classifying a star as main sequence, dwarf, red giant, neutron, pulsar, etc. Another task can focus on galaxy classification (e.g., spiral, barred spiral, elliptical, irregular, etc.). Rather than separating the problem into different tasks where each task is in charge of identifying one type of luminous object, one can combine the tasks together into a single parallel multi-task problem where the hidden layer shares patterns that are common to both classification tasks (see Figure 7.2). The reason learning often improves in accuracy and speed in this context is that training with many tasks in parallel on a single neural network induces information that accumulates in the training signals; if there exist properties common to several tasks, internal nodes can serve to represent common sub-concepts simultaneously.

In the representational transfer approach, most methods use a form of literal transfer, where some knowledge structure is transferred from the source

Non-literal
transfer

Multitask
learning

¹ Previous work did not limit literal or non-literal transfer to a form of representational transfer. We consider the hierarchical representation of Figure 1 more appropriate since functional transfer evades the idea of a sequential transfer of knowledge.

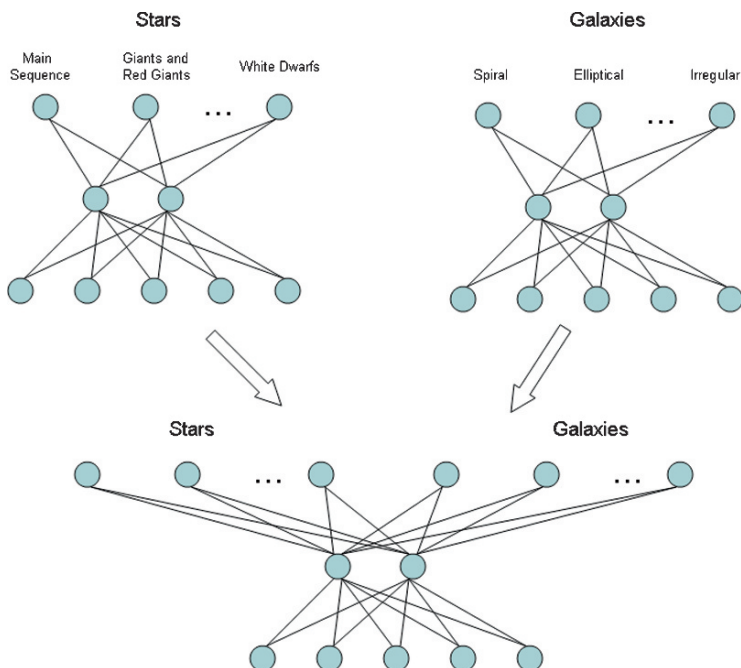


Fig. 7.2. One can combine tasks together into a single parallel multi-task problem; here, multiple luminous objects are identified in parallel using a common hidden layer

network to the target network. This has not always proved to be beneficial; in some cases the target network exhibits a degradation in performance. One simple explanation for this kind of learning behavior lies in the poor relation between previous tasks and the new task [223, 155].

In general, many hybrid variations have been tried around the central idea of sharing a hypothesis structure while learning, often by combining different forms of knowledge transfer. Examples include dividing the neural network into two parts: a common structure at the bottom of the network capturing a common task representation, and a set of upper structures each focused on learning a specific task [14]; adding extra nodes to the network representing contextual information [75]; using previous networks to produce virtual examples (also known as task rehearsal) while learning a new task [227]; or using entire previous networks as new nodes while building a new network [219].

An interesting example of an application of knowledge transfer in neural networks is the search for certain forms of invariance transformations. We mentioned before the importance of finding such transformations in the context of image recognition. As an illustration, suppose we have gathered images of a set of objects under different angles, brightness, location, etc. Let us assume our goal is to automatically learn to recognize an object in an image

using as experience images containing the same object (albeit captured in different conditions). One way to proceed is to train a neural network to learn an invariance function σ (as proposed by Thrun [255]). Function σ is trained with pairs of images generated under different conditions to identify when the images contain the same object. If function σ is approximated with no error, one could perfectly predict the type of object contained in one image by simply applying σ over the current image and previous images containing several prototype objects. In practice, however, finding σ can be intractable and information about the shape of the invariance function (e.g., function slopes) has proved effective to improve the accuracy of the learner.

7.2.2 Transfer in Kernel Methods

Kernel methods such as support vector machines (SVMs) have been extended to work on multi-task learning. Kernel methods look for a solution to the classification (or regression) problem using a discriminant function $g(\cdot)$ of the form:

$$g(\mathbf{x}) = \sum_i c_i k(\mathbf{x}_i, \mathbf{x}) \quad (7.2)$$

where $\{c_i\}$ is a set of real parameters, index i runs along the number of training examples, and k is a kernel function in a reproducing kernel Hilbert space [225].

Knowledge transfer can be effected using kernel methods by forcing the different hypotheses (corresponding to the different tasks) to share a common structure. As an illustration, consider the space of hypotheses made of hyperplanes, where every hypothesis is represented as $\mathbf{w} \cdot \mathbf{x}$ (i.e., as the inner product of $\mathbf{w}$ and $\mathbf{x}$). To employ the idea of having multiple tasks, we assume we have n datasets $\mathbf{T} = \{T_i\}$. Our goal is to produce n hypotheses $\{h_j\}$ from $\mathbf{T}$ under the assumption that the tasks are related. The idea of task relatedness can be incorporated by modifying the space of hypotheses so that the weight vector is made of two components:

$$\mathbf{w}_j = \mathbf{w}_0 + \mathbf{v}_j, \quad 1 \leq j \leq n \quad (7.3)$$

where we assume all models share a common model $\mathbf{w}_0$, and the vectors $\mathbf{v}_j$ Common model serve to model each particular task. In this case we are in effect forcing all hypotheses to share a common component while also allowing for deviations from the common model (as suggested by Evgeniou and Pontil [93]). These ideas can be used to reformulate the optimization problem in support vector machines as follows:

$$\min_{\mathbf{w}_0, \mathbf{v}_j, \xi_{ij}} \sum_i \sum_j \xi_{ij} + \frac{\lambda_1}{n} \sum_j \|\mathbf{v}_j\|^2 + \lambda_2 \|\mathbf{w}_0\|^2 \quad (7.4)$$

subject to the constraints:

$$y_{ij}(\mathbf{w}_0 + \mathbf{v}_j) \cdot \mathbf{x}_{ij} \geq 1 - \xi_{ij} \quad \text{and} \quad \xi_{ij} \geq 0 \quad (7.5)$$

where the ξ_{ij} are the slack variables that capture the empirical error of the models on the data. The second and third terms in equation 7.4 correspond to regularization terms, used to control the overfitting problem by penalizing for models that are too complex (see Section 7.3). By forcing all models to be small, the second term ensures that models do not differ too much from each other. The third term simply controls the complexity of the common model. Under this setting, λ_1 and λ_2 become very relevant parameters. In particular, if λ_1 tends to infinity the problem simplifies to single-task learning; if λ_2 goes to infinity the problem simplifies to solving the n tasks independently. In addition, the ratio $\frac{\lambda_1}{\lambda_2}$ can be used to force all models to be very similar (corresponding to a large ratio) or to consider all tasks as unrelated (corresponding to a small ratio). Metaknowledge can thus be interpreted here as a set of common assumptions about the data distribution for all tasks under analysis. The regularization terms introduce a trade-off between low-complexity models (equivalent to a large margin in SVMs) and how close the models are to a common model (i.e., to a common SVM model).

Several extensions have been proposed to the ideas above [92]. As an example, consider the particular learning scenario where each class is made of n dimensions (i.e., each class value is an n -dimensional vector). The problem becomes that of learning how kernels can be used to represent vector functions [167]. Under this framework, multi-task learning (Section 7.2.1) can be seen as an instance of learning a vector-valued function with specialized kernels and regularization functions that model the possible relationships between tasks. For example, if the regularization function of equation 7.4 is used, the kernel is in fact a combination of two kernels (controlled by λ_1 and λ_2): one kernel that treats the task functions as fully independent, and another kernel that forces the task functions to be similar.²

7.2.3 Transfer in Parametric Bayesian Models

In parametric Bayesian learning the goal is to compute the posterior probability of each class y given an input vector $\mathbf{x}$, $P(y|\mathbf{x})$. For a fixed class y , Bayes theorem results in the following formula:

$$g(\mathbf{x}) = P(y|\mathbf{x}) = \frac{P(\mathbf{x}|y)P(y)}{P(\mathbf{x})} \quad (7.6)$$

where $P(y)$ is the prior probability of class y , $P(\mathbf{x}|y)$ is called the likelihood of y with respect to $\mathbf{x}$ or the class-conditional probability, and $P(\mathbf{x})$ is the evidence factor [82].

Parameter Similarity. One approach to knowledge transfer is as follows (as suggested by Rosenstein et al. [208]). Assume we train a Bayesian learning

² It is also natural to try to learn such a kernel. This can be done, for example, by further minimizing the objective function over a certain class of kernels [7].

algorithm on a task A , resulting in a predictive model with parameter vector θ_A (parameter vector θ_A embeds the set of probabilities required to compute the posterior probabilities). For a new task B , we require that the new probability vector θ_B be similar to the previous one [208] (i.e., $\theta_A \sim \theta_B$). To accomplish this we assume that each component parameter of θ_A and θ_B stems from a hyper-prior distribution. The degree of similarity between parameter components can be controlled by forcing the hyper-prior distribution to have small variance (corresponding to similar tasks) or large variance (corresponding to dissimilar tasks).

Auxiliary Subproblems in Text Classification. To gain more insight into this kind of technique, let us look at another Bayesian approach (proposed by Raina et al. [201]). One interesting application of knowledge transfer is that of text document classification. Here a document is represented with a feature vector $\mathbf{x}$, where each component $x_i \in \{0, 1\}$ indicates if a word (from a fixed vocabulary) is present or not in the document. If Y is the class to which a document belongs, the learning goal is to estimate the posterior probability $P(Y = y|\mathbf{x})$. This can be rephrased using a parametric logistic regression model as follows:

$$P(y|\mathbf{x}) = \frac{1}{1 + \exp(-\theta \cdot \mathbf{x})} \quad (7.7)$$

where θ is a parameter vector containing a weight for each word in our vocabulary. It is common practice in learning to assume a multivariate Gaussian prior on θ of the form $\mathcal{N}(0, \sigma^2 I)$. This essentially assumes the same prior variance σ^2 for each word, with no covariance between words (i.e., words are independent). In practice, words that belong to the same topic tend to appear together. For example, a class of documents where the word “moon” appears often may also have words such as “space” or “astronaut”. Therefore one can instead assume a Gaussian prior $\mathcal{N}(0, \Sigma)$ where Σ is the (feature) covariance matrix that assumes certain dependency between words. We can attempt to approximate this matrix using information from auxiliary subproblems. The idea is to generate smaller problems (texts with few words) to construct a more informative set of priors. Applying logistic regression to each of these subproblems enables us to estimate the variance for each word and covariance between pairs of words. The auxiliary subproblems serve in effect as previous tasks where knowledge is being extracted and subsequently transferred.

Hyper-parameters and Neural Networks. Lastly, we discuss the case where one attempts to estimate hyper-parameters shared among tasks with model parameters corresponding to single tasks [117]. One example is to perform multi-task learning using neural networks combined with a Bayesian approach (as proposed by Bakker and Heskes [11]). Assume a neural network architecture where the links between input and hidden nodes share the same weights for all tasks, but the links between hidden and output nodes have different weights for different tasks (i.e., weights are task-dependent; see

Figure 7.2). Let $\mathbf{A}_i$ be a weight vector between hidden and output links (i.e., the weight vector corresponding to output node i). To estimate the values for each $\mathbf{A}_i$ we can first assume an a priori distribution that employs hyper-parameters common to all weight vectors. For example, we can assume a normal distribution for the weight values based on a predefined mean vector $\mathbf{m}$ and covariance matrix Σ :

$$\mathbf{A}_i \sim \mathcal{N}(\mathbf{A}_i | \mathbf{m}, \Sigma) \quad (7.8)$$

Alternatively we can extend the prior distribution above to a mixture of Gaussians. The next step is to find the set of weights that maximize a posterior probability (maximum a posteriori or MAP value):

$$\mathbf{A}^* = \arg \max_{\mathbf{A}_i} P(\mathbf{A}_i | T_i, \Lambda^*) \quad (7.9)$$

where T_i is the set of examples for task i , and Λ^* is a set of optimal hyper-parameters that maximize the likelihood of the data (the hyper-parameters include the mean $\mathbf{m}$ and covariance Σ). Λ^* is selected to maximize $P(T_i | \Lambda)$. The innovation lies in the fact that Λ^* is found by making use of all available data, $\{T_i\}$, thus allowing the learning of multiple tasks simultaneously. The underlying assumption (i.e., metaknowledge) is that the weights for each task have a similar prior Gaussian distribution.

7.2.4 Other Forms of Transfer

Inductive transfer can be attained in many additional forms, some of which are discussed briefly next.

Probabilistic Transfer and Latent Models. One additional example of inductive transfer is the use of a probabilistic framework under the concepts of latent variables and independent component analysis. For example, one can assume that the n parameters $\theta_1, \theta_2, \dots, \theta_n$ modeling the set of tasks can be represented as a combination of a set of (hypothetical) hidden source models [287]. The parameters are thus *related* by a combination of hidden source models that can be unveiled using independent component analysis. A similar approach uses linearly mixed Gaussian processes to model dependencies for the response (i.e., class) variables [252].

Transfer by Feature Mapping. One view of inductive transfer manipulates the input features. A straightforward method uses the predictions of hypotheses used on old datasets as features on new datasets. One example where this strategy has proved useful is in problems where the target concept changes over time (i.e., the concept drift problem), where predictions of classifiers on old temporal data are useful in predicting class labels for current data [97]. Using the predictions of classifiers as new features has also been reported in graphical models, particularly in conditional random fields [244]. Note this

Similar
prior
distribution

Predictions
as features

is different from the idea of stacked generalization [284], where predictions – used as features – do not originate from previous tasks.

Transfer by Clustering. One approach to learning to learn consists of designing a learning algorithm that groups similar tasks into clusters. A new task is assigned to the most related cluster; inductive transfer takes place when generalization exploits information about the cluster to which each task belongs [257]. Tasks as clusters

The idea of clustering similar tasks has also been pursued under a Bayesian approach. Essentially, each vector $\mathbf{A}_i$ of hidden to output weights (see the parametric Bayesian models above) is modeled as a mixture of Gaussians [11]:

$$\mathbf{A}_i \sim \sum_{\alpha} q_{\alpha} \mathcal{N}(\mathbf{A}_i | \mathbf{m}_{\alpha}, \Sigma_{\alpha}) \quad (7.10)$$

where q_{α} is the prior probability of a task assigned to cluster α , and $\mathbf{m}_{\alpha}$ and Σ_{α} are the mean and covariance of each Gaussian respectively. Here, each Gaussian is in fact describing a cluster of tasks.

7.2.5 Meta-searching for Problem Solvers

We now move to a different research direction in learning to learn that explores complex scenarios where the software architecture itself evolves with experience. The main idea is to divide a program into different components that can be reused during different stages of the learning process. As an illustration, one can work within the space of (self-delimiting binary) programs to propose an optimal ordered problem solver (as suggested by Schmidhuber [218, 217, 216]). The goal is to solve a sequence of problems, arriving one after the other, as optimally as possible; ideally the system should be capable of exploiting previous solutions and incorporating them into the solution of the current problem. This can be done by allocating computing time to the search for previous solutions that, if useful, become new building blocks. If the current problem can be solved by copying or invoking previous pieces of code (i.e., building blocks), then the mechanism will accept those solutions with substantial savings in computational time. Meta-searching
Component reuse

When looking for a solution to a problem, a metalearning algorithm has the choice of generating new programs or reusing previously generated candidate programs (grown incrementally by relying on previous computable solutions). This mechanism embeds a trade-off between exploration (search for new programs) and exploitation (search for variant solutions). The rationale is that exploiting experience collected in previous search steps can solve the target problem much faster. Although the connection to other metalearning mechanisms addressed in this chapter is not explicit, applying such a methodology on a learning problem would be equivalent to storing learning modules or components while dynamically constructing new learning algorithms exhibiting high generalization performance. Stored candidate learning components

would represent a form of metaknowledge. Exploiting this information is akin to exploiting knowledge about learning, or knowledge for incremental self improvement. Practical implementations of these ideas can be found in the CAMLET system [246, 247, 245] (Chapters 4 and 8), and to a lesser extent in the Intelligent Discovery Assistant (Chapter 4). Both systems use complete learning algorithms as learning components. Experimental results using CAMLET show improved predictive accuracy when using a genetic algorithm to search for an optimal combination of learning components.

7.3 A Theoretical Framework

Several studies have provided a theoretical analysis of the learning-to-learn paradigm. The aim is to understand the conditions under which a metalearner can provide good generalizations when embedded in an environment made of related tasks. Although the idea of knowledge transfer is normally made implicit in the analysis, it is clear that the metalearner extracts and exploits knowledge from every task to perform well on future tasks. Theoretical studies fall within a Bayesian model [15, 117] and a probably approximately correct (PAC) model [16, 165]. The idea is to find not only the right hypothesis h in a hypothesis space $\mathcal{H}$, $h \in \mathcal{H}$, but in addition to find the right hypothesis space $\mathcal{H}$ in a family of hypothesis spaces $\mathbb{H}$, $\mathcal{H} \in \mathbb{H}$.

Let us look at these studies more closely. We focus on the problem of bounding the number of examples needed to produce good generalizations when the learner faces a stream of tasks (other studies provide a different perspective by looking at the amount of information required for each task to learn n tasks [15]).

Consider first that the goal of traditional learning is to find a hypothesis $h^* \in \mathcal{H}$ that minimizes a functional risk:

$$h^* = \arg \min_{h \in \mathcal{H}} R_\phi(h) \quad (7.11)$$

where

$$R_\phi(h) = \int_{\mathbf{x} \in \mathcal{X} \times \mathcal{Y}} L(h(\mathbf{x}), y) d\phi(\mathbf{x}, y) \quad (7.12)$$

The risk corresponds to the expected loss incurred by hypothesis h ; $L(h(\mathbf{x}), y)$ is a particular loss function (e.g., zero-one loss) and the integral runs across the input-output space. We assume a probability distribution ϕ (i.e., a learning task) over $\mathcal{X} \times \mathcal{Y}$ that indicates which examples are more likely to be seen for that particular task. Since we do not have access to all possible examples in the input-output space, we may choose to approximate the true risk with an empirical risk $\hat{R}_\phi(h)$. We do this by randomly sampling m examples according to ϕ to generate a training sample $T = \{(\mathbf{x}_j, y_j)\}_{j=1}^m$, where:

$$\hat{R}_\phi(h, T) = \frac{1}{m} \sum_{j=1}^m L(h(\mathbf{x}_j), y_j) \quad (7.13)$$

It has been formally shown that one can bound the true risk $R_\phi(h)$ as a function of the empirical risk $\hat{R}_\phi(h, T)$ if there exists a uniform bound for all $h \in \mathcal{H}$ on the probability of deviation between $R_\phi(h)$ and $\hat{R}_\phi(h, T)$ [272, 28]. Such bounds can be represented as a function of the Vapnik-Chervonenkis (VC) dimension of the hypothesis space $\mathcal{H}$, $\text{VC}(\mathcal{H})$. The VC dimension captures the degree of expressiveness or richness in delimiting flexible decision boundaries by the set of functions in $\mathcal{H}$; it provides an objective characterization of $\mathcal{H}$ [272]. Bounds for the deviation between $R_\phi(h)$ and $\hat{R}_\phi(h, T)$ take on the form

$$R_\phi(h) \leq \hat{R}_\phi(h, T) + g(m, \delta, \text{VC}(\mathcal{H})) \quad (7.14)$$

where function $g(\cdot)$ explicitly indicates an upper bound on the deviation between the true risk and the empirical risk; the inequality is satisfied for all $h \in \mathcal{H}$ with probability $1 - \delta$ (according to the choice of training set T).

7.3.1 The Learning-to-Learn Scenario

Let us now consider the novelty brought about by the learning-to-learn scenario (following Baxter [16]). Here we assume the learner is embedded in a set of related tasks that share certain commonalities. Let us go back to the problem where a metalearner is designed for recognition of astronomical objects; the idea is to classify objects (e.g., stars, galaxies, nebulae, planets) extracted from images mapping certain regions of the sky. One way to transfer learning experience from one astronomical center to another is by sharing a metalearner that carries a bias towards recognition of astronomical objects. In traditional learning, we assume a probability distribution ϕ_i that indicates which examples are more likely to be seen in such a task. Now we assume there is a metadistribution Φ over the space of all possible distributions ϕ_i . In essence Φ indicates which tasks are more likely to be found within the sequence of tasks faced by the metalearner (just as ϕ_i indicates which examples are more likely to be seen in such a task). In our example, Φ stands for a probability distribution that peaks over tasks corresponding to classification of astronomical objects. Given a family of hypothesis spaces $\mathbb{H}$, the goal of the metalearner is to find a hypothesis space $\mathcal{H}^* \in \mathbb{H}$ that minimizes a new functional risk:

Meta-distribution

$$\mathcal{H}^* = \arg \min_{\mathcal{H} \in \mathbb{H}} R_\Phi(\mathcal{H}) \quad (7.15)$$

where

$$R_\Phi(\mathcal{H}) = \int_{\phi_i \in \Phi} \inf_{h \in \mathcal{H}} R_{\phi_i}(h) d\Phi(\phi_i) \quad (7.16)$$

An expansion of the above formula gives

$$R_{\Phi}(\mathcal{H}) = \int_{\phi_i \in \Phi} \inf_{h \in \mathcal{H}} \int_{\mathbf{x} \in \mathcal{X} \times \mathcal{Y}} L(h(\mathbf{x}), y) d\phi_i(\mathbf{x}, y) d\Phi(\phi_i) \quad (7.17)$$

The new functional risk, $R_{\Phi}(\mathcal{H})$, represents the expected loss of the best possible hypothesis in each hypothesis space. The integral runs across all task distributions ϕ_i , which are themselves distributed according to a metadistribution Φ . In practice, since we ignore the form of Φ , we need to draw samples $T_1, T_2, \dots, T_n$ to infer how tasks are distributed in our environment.

To summarize, in the learning-to-learn scenario our input is made of n samples $\mathbf{T} = \{T_i\}_{i=1}^n$, where each sample T_i is composed of m examples $\{(\mathbf{x}_j^i, y_j^i)\}_{j=1}^m$. The goal of the metalearner is to *output a hypothesis space with a learning bias that generates accurate models for a new task*.

In conventional learning a learning algorithm A maps a training set T into a hypothesis:

$$A : \bigcup_{m>0} (\mathcal{X} \times \mathcal{Y})^m \rightarrow \mathcal{H} \quad (7.18)$$

In contrast, in learning to learn, a metalearner $\mathcal{A}$ is a function that maps a sequence of training sets into a hypothesis space:

$$\mathcal{A} : \bigcup (\mathcal{X} \times \mathcal{Y})^{(n,m)} \rightarrow \mathbb{H} \quad (7.19)$$

The advantage of working on a learning-to-learn scenario is that the learner accumulates experience after each new task. Such experience, here referred to as metaknowledge, is expected to result in more accurate models when the tasks share commonalities or patterns. The expectation is that as more tasks are observed, the number of examples required to attain accurate models (with high probability) decreases over time.

7.3.2 Bounds on Generalization Error for Metalearners

Finding bounds on the generalization error for metalearners follows the same logic as that adopted in conventional learning theory. The idea is to formally show that it is possible to bound the new functional risk $R_{\Phi}(\mathcal{H})$ as a function of the empirical risk $\hat{R}_{\Phi}(\mathcal{H})$. Given a set of n samples $\mathbf{T} = \{T_i\}$, the empirical risk is defined as the average of the best possible empirical error for each training sample T_i :

$$\hat{R}_{\Phi}(\mathcal{H}) = \frac{1}{n} \sum_{i=1}^n \inf_{h \in \mathcal{H}} \hat{R}_{\phi}(h, T_i) \quad (7.20)$$

The bound can be found if there exists a uniform bound for all $\mathcal{H} \in \mathbb{H}$ on the probability of deviation between $R_{\Phi}(\mathcal{H})$ and $\hat{R}_{\Phi}(\mathcal{H})$. In conventional learning theory these bounds are governed by the expressiveness of the family

of hypotheses $\mathcal{H}$. Similarly, in the learning-to-learn scenario, bounds on generalization error are governed by the size of function classes associated with the family space $\mathbb{H}$.

Specifically, one can guarantee that with probability $1 - \delta$ (according to the choice of samples $\mathbf{T}$), all $\mathcal{H} \in \mathbb{H}$ will satisfy the following inequality:

$$R_{\Phi}(\mathcal{H}) \leq \hat{R}_{\Phi}(\mathcal{H}) + \epsilon \quad (7.21)$$

This holds if the number of tasks n is such that

$$n \geq \max \left\{ \frac{256}{\epsilon^2} \log \frac{8\mathcal{C}(\frac{\epsilon}{32}, \Lambda_{\mathbb{H}})}{\delta}, \frac{64}{\epsilon^2} \right\} \quad (7.22)$$

and the number of examples m for each task is such that

$$m \geq \max \left\{ \frac{256}{n\epsilon^2} \log \frac{8\mathcal{C}(\frac{\epsilon}{32}, \Lambda_{\mathbb{H}}^{\mathbf{n}})}{\delta}, \frac{64}{\epsilon^2} \right\} \quad (7.23)$$

The theorem (proved by Baxter [16]) introduces two new properties characterizing the family of hypothesis spaces $\mathbb{H}$, $\mathcal{C}(\epsilon, \Lambda_{\mathbb{H}})$ and $\mathcal{C}(\epsilon, \Lambda_{\mathbb{H}}^{\mathbf{n}})$. These functions measure the capacity of $\mathbb{H}$ in a way similar to how the VC dimension measures the capacity of $\mathcal{H}$. To provide continuity to our chapter we defer explanation of these properties to Appendix A. The bounds stated above simply show that to learn both a good hypothesis space $\mathcal{H} \in \mathbb{H}$ and a good hypothesis $h \in \mathcal{H}$, one needs a minimum number of both the number of tasks and the number of examples on each task. It is known that if ϵ and δ are fixed [16], the number of examples m needed on each task to attain an accurate model is such that

$$m = O \left(\frac{1}{n} \log \mathcal{C}(\epsilon, \Lambda_{\mathbb{H}}^{\mathbf{n}}) \right) \quad (7.24)$$

This indicates that the required number of examples on each task decreases as the number of tasks increases, in accordance with our expectations of the benefits gained when the learning algorithm has the capability of exploiting previous experience.

7.3.3 Other Theoretical Work

New Bounds Using Theory of Algorithmic Stability

Recent work has shown alternative views to the theory behind the learning-to-learn paradigm (as developed by Maurer [165]). Results from Section 7.3.2 can be improved if one makes certain assumptions. To understand this we need to review the concept of algorithmic stability (introduced by Bousquet and Elisseeff [33]). A learning algorithm is said to be uniformly β -stable if taking away one example from the training set does not modify the loss of

Algorithmic
stability

the output hypothesis by more than β (for a fixed loss function). We update our definition of a metalearning algorithm as a function $\mathcal{A}(\mathbf{T})$ that outputs a hypothesis after looking at a sequence of samples $\mathbf{T} = \{T_i\}_{i=1}^n$. That is, we no longer talk about a hypothesis space, but of a single hypothesis that does well on all previous tasks. In that case, one can also think of a metalearning algorithm as being β' -stable if removing one sample from the set of samples $\mathbf{T}$ does not modify the loss of the output hypothesis by more than β' . Notice that parameter β' corresponds to the concept of stability across tasks, whereas parameter β is used to refer to stability across examples drawn from one task.

Given that $\mathcal{A}(\mathbf{T}) = h$ for a given set of samples $\mathbf{T}$, the new results show that for every environment Φ , with probability greater than $1 - \delta$ according to the selection of $\mathbf{T}$, the following inequality holds:

$$\forall \Phi \ R_{\Phi}(h) \leq \frac{1}{n} \sum_{i=1}^n \hat{R}_{\phi_i}(h, T_i) + 2\beta' + (4n\beta' + m) \sqrt{\frac{\ln(1/\delta)}{2n}} + 2\beta \quad (7.25)$$

where $\phi_i \in \Phi$ and $\hat{R}_{\phi_i}(h, T_i)$ is an estimation of the empirical loss of hypothesis h when the examples are drawn from sample T_i . The first term on the right-hand side of the inequality is then the average empirical loss of h on the set of tasks $\mathbf{T}$. It can be shown that the new bound is tighter than that of Section 7.3.2 (of course under the assumption of stability parameterized by β and β' on $\mathcal{A}(\mathbf{T}) = h$).

New Bounds Based on Task Similarity

We explain one more interesting theoretical study in learning to learn. It has been assumed so far that previous tasks are *related*, with no mechanism to quantify the degree of *relatedness* between different tasks. Such a mechanism can serve to indicate how much gain can be derived when learning a new task if it relates to our set of previous tasks. One approach to using task relatedness is to think about a set $\mathcal{F}$ of transformations $f : \mathcal{X} \rightarrow \mathcal{X}$ (as proposed by Ben-David [17]). The motivation is that many real-world problems contain multiple datasets that capture the same set of objects but from different perspectives. A good example is that of face recognition; when a face has been captured at different angles and varying brightness, a set of transformations can be used to recognize when two images belong to the same face. Formally, we say two samples are $\mathcal{F}$ -related if they are obtained from $\mathcal{F}$ -related distributions. Two distributions ϕ_1 and ϕ_2 are $\mathcal{F}$ -related if there exists a transformation $f \in \mathcal{F}$ that, after being applied to $\mathcal{X}$, makes the two distributions equivalent, i.e., for a sample $T \subseteq \mathcal{X} \times \mathcal{Y}$, $\phi_1(T) = \phi_2(f(T))$.

To provide tight error bounds between the true and empirical risk, we start with an initial space of hypothesis $\mathcal{H}$ and then separate this space into equivalence classes under $\mathcal{F}$. Two hypotheses h_i and h_j belong to the same class $[h]$ if there exists an $f \in \mathcal{F}$ such that $h_j = h_i \circ f$ (i.e., $\forall \mathbf{x}, h_j(\mathbf{x}) = h_i(f(\mathbf{x}))$). The advantage of this method consists precisely in separating a

hypothesis space $\mathcal{H}$ into equivalence classes $[h]_{\sim \mathcal{F}}$. The learning process is now simplified by reducing the complexity of the hypothesis space into a few classes. The goal here is to find upper bounds on the sample complexity of finding a class $[h]_{\sim \mathcal{F}}$ that is close to optimal for every single task. Equivalence classes

Following the above, it has been shown [17] that for any $\epsilon \geq 0, \delta \leq 1$ and $h \in \mathbb{H}$, if $\mathbf{T} = T_1, T_2, \dots, T_n$ is an $\mathcal{F}$ -similar sequence of training samples drawn respectively from distributions $\phi_1, \dots, \phi_n, \forall i |T_i| \geq m$, and³

$$m \geq \frac{88}{\epsilon^2} [2d_{\mathbb{H}}(n) \log \frac{22}{\epsilon} + \frac{1}{n} \log \frac{4}{\delta}], \quad (7.26)$$

then with probability at least $1 - \delta$ (over the choice of $\mathbf{T}$), for any $1 \leq j \leq n$,

$$| \inf_{[h]_{\sim \mathcal{F}} \in \mathcal{H}} R_{\phi_j}([h]_{\sim \mathcal{F}}) - \inf_{h_1, h_2, \dots, h_n \in [h]_{\sim \mathcal{F}}} \frac{1}{n} \sum_{i=1}^n \hat{R}(h_i, T_i) | \leq \epsilon \quad (7.27)$$

The advantage of this approach over previous methods is that the bounds are defined by searching for the equivalence class $[h]_{\sim \mathcal{F}}$ that is near optimal for each of the tasks (as opposed to methods that obtain an average bound over all tasks [16]).

7.3.4 Bias vs. Variance in Metalearning

As part of our theoretical study, we end by looking into the nature of the bias-variance dilemma in classification when immersed in a learning-to-learn scenario. Let us first recall what the bias-variance dilemma states in traditional learning [114, 110]. The dilemma is based on the fact that the prediction error (i.e., expected error loss on unseen examples) can be decomposed into a bias and a variance components.⁴ Ideally we would like to have classifiers with both low bias and low variance but these components are inversely related. On the one hand, simple classifiers encompass a small hypothesis space $\mathcal{H}$. Their small repertoire of functions produces high bias (since the hypothesis with lowest prediction error may lie far from the true target function) but low variance (since there is little dependence on local irregularities in the data). On the other hand, increasing the size of $\mathcal{H}$ reduces the bias but increases the variance. The large size of $\mathcal{H}$ normally allows for flexible decision boundaries (low bias) but the learning algorithm inevitably becomes sensitive to small variations in the data (high variance). Bias-variance dilemma

In the learning-to-learn framework, there is an equal need to find a balance in the size of the family of hypothesis spaces $\mathbb{H}$. A small $\mathbb{H}$ will exhibit low

³ The lower bound for the sample complexity m introduces a new term $d_{\mathbb{H}}(n)$ that can be understood as a generalized version of the VC dimension for a family of hypothesis spaces $\mathbb{H}$ (see [16, 17]).

⁴ A third component, the *irreducible error* or *Bayes error*, cannot be eliminated or traded.

variance and high bias; here, unless we can find a good hypothesis space $\mathcal{H} \in \mathbb{H}$ with a small risk $R_\Phi(\mathcal{H})$, the best $\mathcal{H}$ may be far from the true hypothesis space modeling the actual phenomenon under study. And just as in traditional learning, a large $\mathbb{H}$ will exhibit low bias but high variance, since the large number of available hypothesis spaces increases the chances of selecting one that simply accommodates the idiosyncracies of the sequence of empirical data $\mathbf{T} = \{T_i\}_{i=1}^n$.

Current research aims at understanding if learning the right family of hypothesis spaces $\mathbb{H}$ is inherently easier than learning the right space $\mathcal{H}$ in traditional learning. Some recent work suggests that learning $\mathbb{H}$ may be simpler than learning $\mathcal{H}$ [16].

7.4 Challenges in Knowledge Transfer

7.4.1 Representational Language of Explicit Metaknowledge

We end our chapter by looking into current challenges in knowledge transfer. One challenge involves devising learning architectures with an explicit representation of metaknowledge [248, 249]. Most metalearning systems make an implicit assumption about the transfer process by modifying the bias embedded by the hypothesis space; in most situations this form of implicit knowledge is not readily available for reuse. For example, we may change the bias by selecting a learning algorithm that draws linear boundaries over the input space instead of one that draws quadratic boundaries; here, no explicit knowledge is transferred specifying our preference for linear boundaries. Because of this limitation, transferring knowledge across domains becomes problematic and in need of new cognitive architectures [249].

7.4.2 High-Level Task Characterization

Another challenge is to understand why a learning algorithm performs well or not on certain datasets, and to use that knowledge to improve its performance. Recent work in metalearning points to the relation between dataset characteristics and learning performance as a critical research field. The central idea is that high-quality dataset characteristics or metafeatures provide enough information to differentiate the performance of a set of given learning algorithms [1, 169, 106, 37, 141, 41]. From a practical perspective, a proper characterization of datasets leads to an interesting goal: the construction of metalearning assistants. The main role of these assistants is to recommend a good predictive model given a new dataset, or to attempt to modify the learning mechanism before it is invoked again in a dataset drawn from a similar distribution. Moreover, this holds whether it refers to selecting a good predictive model, estimating model parameters, looking for heterogeneous models in the context of stacking [284, 59], or looking for the best combination of data mining processes (plan), as discussed elsewhere in this book (Chapters 1 and 5).

Meta-
features

Meta-
learning
assistants

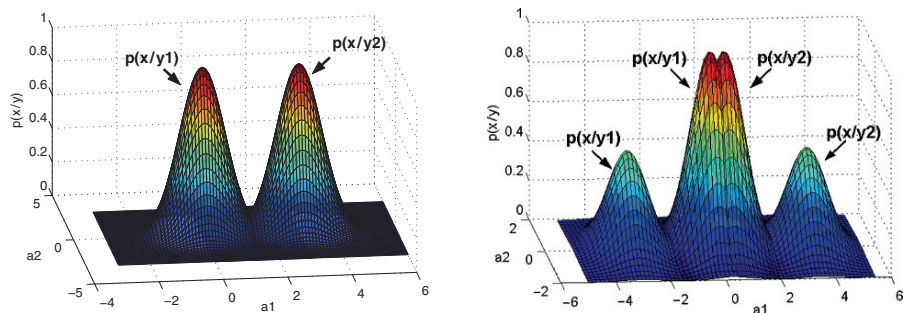


Fig. 7.3. Two different class-conditional distributions on two real-valued features and two classes; (left) a distribution with few structures that is easy to learn; (right) a rough distribution with multiple peaks and high overlap between classes that is hard to learn

The construction of metalearning assistants is contingent on the availability of new forms of data characterization that can be directly used to explain the connection between example distributions and learning strategies [206, 203, 204, 205]. As an illustration, one can look at an example distribution as a data landscape over the input space where elevations correspond to class-conditional probabilities. For example, Figure 7.3 shows two different forms of data landscapes constructed using a simple and multimodal Gaussian distributions over two real-valued features and two classes. Figure 7.3 (left) denotes a data landscape easy to learn; examples cluster around well-separated class-uniform peaks. Figure 7.3 (right) in contrast denotes a data landscape where Bayes error is high (i.e., where learning is inherently complicated). Though most problems are multidimensional, the example helps us visualize the different types of landscapes in need of a robust characterization.

Data
landscape

7.4.3 Inductive Transfer in Robotics

A vast amount of research has been reported on the applications of machine learning in robotics. Our attempt here is limited to exemplify the importance of inductive transfer in robotics applications while pointing to important challenges.⁵ We start by describing an interesting application of inductive transfer on competitive games involving teams of robots (e.g., Robocup Soccer [183]). In this scenario, transferring knowledge learned from one task into another task is crucial to acquire skills necessary to beat the opponent team. As an example, imagine a situation where a team of robots has been taught to keep a soccer ball away from the opponent team (as proposed by Stone and Sutton [242]). To achieve that goal, robots must learn to keep the ball, pass the ball to a close teammate, etc., always trying to remain at a safe distance from the

Transfer in
robotics

⁵ Some of the work described employs simulated agents rather than actual physical robots.

opponents. Now let us assume we wish to teach the same team of robots to play a different game where they must learn to score against a team of defending robots (as proposed by Maclin et al. [163]). Knowledge gained during the first activity can be transferred to the second one. Specifically, a robot can prefer to perform an action learned in the past over actions proposed during the current task because the past action has a significant higher merit value [267, 266]. For example, a robot might learn in the first task that it should pass to a teammate when an opponent is getting too close. This knowledge is useful in the second task; to be effective at scoring, the agent should combine knowledge on how to keep the ball away from the opponent team with accurate shooting.

Transfer in
reinforce-
ment
learning

Most work on knowledge transfer applied to robotics assumes a form of reinforcement learning as the central learning mechanism. In reinforcement learning the goal is to find an optimal policy mapping states (e.g., location of robots, angles between them, distances) to actions (e.g., hold the soccer ball, pass the ball) so as to maximize a long-term reward function. One of the first attempts to learn from previous experience is based on the problem of balancing a pole hinged to a cart that moves along a 1-D track. It has been shown that keeping the pole balanced becomes easier under varying conditions (e.g., smaller or heavier pole) when the learning task begins with a policy already acquired before using some initial conditions [221]. Many additional examples have been reported where knowledge transfer is performed using reinforcement learning (e.g., by decomposing a task into subtasks so as to facilitate the learning of new tasks [229], by letting one learner imitate another learner [194], or by using hierarchical reinforcement learning to transfer subroutines between tasks [6]).

One of the most important challenges during inductive transfer in robotics is that of automatically generating a transformation function to map action and state spaces from one task into another task (as observed by Taylor and Stone [250]). To understand this let us go back to the example of the soccer-playing robots; here it is reasonable to expect different tasks to exhibit different state parameters and actions. For example, keeping the soccer ball away from the opponent team would need a new representation if one were to increase the number of players on each team; additional players would increase the number of parameters which in turn would modify the state space. While it has been shown that it is possible to provide such transformations in particular domains, it remains an open problem to show how the transformation itself can be automatically acquired or learned [162, 251]. It would be equally desirable to learn how to automate the process of generating pieces of advice from one task to another [267].

One proposed solution to alleviate the common dependency on user information characterizing the robot controller and environment is to embed the robot learner in a lifelong scenario (as suggested by Thrun [253]). Due to the inherent complexity of many robot tasks where the environment is characterized by a high degree of uncertainty, one approach is to let the robot transfer

knowledge as it accumulates experience. Specifically a robot can learn the consequences of its actions for a particular environment by learning a mapping from a previous state and action to the present state. If the environment is the same, such an action-model function would be instrumental to learning invariants across different tasks. When the robot faces a new task and attempts to learn a control function mapping states to actions, action models can be used as background knowledge by enabling the robot to anticipate the consequences of executing a sequence of actions [256]. A current challenge in inductive transfer is to find efficient ways to make knowledge accumulated by a lifelong learner readily available when dealing with new tasks.

Appendix

Section 7.3.2 makes use of two properties characterizing the space of a family of hypothesis spaces $\mathbb{H}$, $\mathcal{C}(\epsilon, \Lambda_{\mathbb{H}})$ and $\mathcal{C}(\epsilon, \Lambda_{\mathbb{H}}^{\mathbf{n}})$. These functions quantify the capacity of the space of a family of hypothesis spaces $\mathbb{H}$. We now explain the nature of these properties in more detail:⁶

Definition 1. For each $\mathcal{H} \in \mathbb{H}$, define a new function $\lambda_{\mathcal{H}}(\phi_i)$ by

$$\lambda_{\mathcal{H}}(\phi_i) = \inf_{h \in \mathcal{H}} R_{\phi_i}(h) \quad (7.28)$$

where $\lambda : \Phi \rightarrow [0, 1]$. In other words, function λ specifies the minimum error loss achieved after looking at every $h \in \mathcal{H}$ under distribution ϕ_i .

Definition 2. For the family of hypothesis spaces $\mathbb{H}$, define a new set $\Lambda_{\mathbb{H}}$ by

$$\Lambda_{\mathbb{H}} = \{\lambda_{\mathcal{H}} : \mathcal{H} \in \mathbb{H}\} \quad (7.29)$$

The set $\Lambda_{\mathbb{H}}$ contains all *different* functions according to Def. 1 within the space of a family of hypotheses $\mathbb{H}$. We can compute the expected difference in the minimum error loss for any two functions $\lambda_1, \lambda_2 \in \Lambda_{\mathbb{H}}$ as follows.

Definition 3. For any two functions $\lambda_1, \lambda_2 \in \Lambda_{\mathbb{H}}$, and a distribution Φ on the space of possible input-output distributions, define

$$D_{\Phi}(\lambda_1, \lambda_2) = \int_{\phi_i} |\lambda_1(\phi_i) - \lambda_2(\phi_i)| d\Phi(\phi_i) \quad (7.30)$$

Function D can be seen as the expected distance between two functions λ_1, λ_2 . We now define the concept of an ϵ -cover as follows.

Definition 4. An ϵ -cover of $(\Lambda_{\mathbb{H}}, D_{\Phi})$ is a set $\{\lambda_1, \lambda_2, \dots, \lambda_n\}$ such that for all $\lambda \in \Lambda_{\mathbb{H}}$, $D_{\Phi}(\lambda, \lambda_i) \leq \epsilon$ ($1 \leq i \leq n$). Let $\mathcal{N}(\epsilon, \Lambda_{\mathbb{H}}, D_{\Phi})$ represent the size of the smallest ϵ -cover. We now define the capacity of $\Lambda_{\mathbb{H}}$ by

⁶ We follow Baxter's work [16] in different order and notation to simplify the explanation of the two properties characterizing $\mathbb{H}$.

$$\mathcal{C}(\epsilon, \Lambda_{\mathbb{H}}) = \sup_{\phi} \mathcal{N}(\epsilon, \Lambda_{\mathbb{H}}, D_{\phi}) \quad (7.31)$$

where the supremum runs over all probability distributions over $\mathcal{X} \times \mathcal{Y}$.

We can similarly define the second capacity $\mathcal{C}(\epsilon, \Lambda_{\mathbb{H}}^{\mathbf{n}})$. To begin, consider a sequence of n tasks that has been modeled with n hypotheses $\mathbf{h} = (h_1, h_2, \dots, h_n)$. We can compute the expected error loss across n tasks as follows:

$$\lambda_{\mathbf{h}}^n(\{\mathbf{x}_i, y_i\}) = \frac{1}{n} \sum_{i=1}^n L(h_i(\mathbf{x}_i), y_i) \quad (7.32)$$

Definition 5. For the space of a family of hypotheses $\mathbb{H}$, define a new set $\Lambda_{\mathbf{h}}^n$ by

$$\Lambda_{\mathbf{h}}^n = \{\lambda_{\mathbf{h}}^n : h_1, h_2, \dots, h_n \in \mathcal{H}\} \quad (7.33)$$

The set $\Lambda_{\mathbf{h}}^n$ is a loss function class and as before it indicates how many *different* classes of functions (capturing the average error loss for a sequence of n hypotheses) are contained within the hypothesis space $\mathcal{H}$; the difference is that now we are comparing sets of n loss functions.

Definition 6. For the space of a family of hypotheses $\mathbb{H}$, define

$$\Lambda_{\mathbb{H}}^{\mathbf{n}} = \bigcup_{\mathbf{h} \in \mathbb{H}} \Lambda_{\mathbf{h}}^n \quad (7.34)$$

where $\mathbf{h} \subseteq \mathcal{H}$. The second capacity $\mathcal{C}(\epsilon, \Lambda_{\mathbb{H}}^{\mathbf{n}})$ is defined similarly to the first one but using a new distance function:

$$D_{\phi}^n(\mathbf{h}, \mathbf{h}') = \int_{(\mathcal{X} \times \mathcal{Y})^n} |\lambda_{\mathbf{h}}^n(\{\mathbf{x}_i, y_i\}) - \lambda_{\mathbf{h}'}^n(\{\mathbf{x}_i, y_i\})| d\phi_1, d\phi_2, \dots, d\phi_n \quad (7.35)$$

This brings us to the second capacity function:

$$\mathcal{C}(\epsilon, \Lambda_{\mathbb{H}}^{\mathbf{n}}) = \sup_{\phi_i} \mathcal{N}(\epsilon, \Lambda_{\mathbb{H}}^{\mathbf{n}}, D_{\phi}^n) \quad (7.36)$$

where the supremum runs over all sequences of n probability distributions over $\mathcal{X} \times \mathcal{Y}$.

Composition of Complex Systems: Role of Domain-Specific Metaknowledge

8.1 Introduction

The aim of this chapter is to discuss the problem of employing learning methods in the design of complex systems. The term *complex systems* is used here to identify systems that cannot be learned in one step, but rather require several phases of learning. Our aim will be to show how domain-specific metaknowledge can be used to facilitate this task.

8.1.1 Dynamic Selection of Bias

To introduce this problem we need to come back to the problem of dynamic selection of bias discussed in Chapter 1. As was mentioned there, bias is, according to DesJardins and Gordon [112], *any factor that influences the definition or selection of inductive hypotheses*. Let us review how this concept was used in the task of selecting suitable Machine Learning (ML) or Data Mining (DM) algorithms for a given dataset (see Figure 1.1 in Chapter 1). Typically, a new dataset is given and then we seek to identify one or more suitable ML/DM algorithms for that task. As was mentioned, information in the metaknowledge base is used in the process. The identification of a suitable subset of ML algorithms from a larger set can be considered as *dynamic selection of bias*. By eliminating some ML/DM algorithms, we are, in effect, excluding some forms of inductive hypotheses from consideration.

Dynamic
selection of
bias

Let us now consider another possible interpretation of bias when applying ML/DM algorithms. Without loss of generality, let us simply focus on one ML algorithm to simplify the exposition. Let us further assume that the aim is to predict a categorical (or a numeric) value of some variable, but the rest of the data includes potentially a *very large number of attributes*. So a question arises about what should be done in this case.

A typical solution adopted is to gather the data first and then use some standard feature elimination method (see, e.g., [280]) to reduce the number of features as appropriate. However, this approach has the following shortcoming. Someone has to decide which attributes/features are potentially relevant

for the task at hand. For instance, the task can be to predict the value of some class variable, such as credit risk. In this case, we would want to consider, for instance, financial and/or personal data of the prospective customer. If a wrong decision is made, this can create difficulties for the learning system. If the relevant attributes are *not included*, a suboptimal hypothesis may be generated. If on the other hand the set of attributes is *too large* and includes unnecessary information, it may again be difficult for the system to generate the right hypothesis (the search space of inductive hypotheses may be too large). So, it is obviously advantageous to have methods that help us to determine the relevant attributes automatically.

Determining which attributes (or in general which concepts) should be used can be considered as the problem of dynamic selection of bias, as it satisfies the definition given earlier. Our aim in this chapter is to discuss this issue in more detail, clarify its relationship to metalearning and suggest how dynamic selection of bias can be handled.

Determining which concepts should be brought into play is influenced by the learning goal. This issue was noted by the Russian psychologist Wygotski [286] in 1934. He drew attention to the fact that concepts arise and develop if there is a specific need for them. Acquisition of concepts is thus a purposeful activity directed towards reaching a specific goal or a solution of a specific task.

This problem has been noted also by many people in AI and ML. Various researchers (e.g., Hunter and Ram [128, 127], Michalski [168], Ram and Leake [202], etc.) have argued that it is important to define explicit goals that guide learning. Learning is seen as search through a knowledge space guided by the learning goal. Learning goals determine which parts of prior knowledge are relevant, what knowledge is to be acquired and in what form, how it is to be evaluated and when to stop learning. The importance of planning in this process has also been identified [129].

As we will see, dynamic selection of bias is important when dealing with this issue. Whenever we are concerned with the problem of constructing complex systems, we need not only identify the attributes / features / concepts that are potentially relevant, but also one or more subproblems (concepts) that constitute the final solution. Typically, it is advantageous to define also some ordering in which (some of) the subproblems (concepts) should be acquired. This problem can be seen as the problem of *learning multiple interdependent concepts*. Let us now see how it can be related to the issue of bias discussed earlier. In effect, defining the ordering can be regarded as defining the appropriate procedural bias, as this ordering determines how the hypothesis space should be searched.

8.1.2 Representation of Multiple Learning Goals and Concepts

In the light of the above discussion, it is important to have a good representation for multiple learning goals, their interdependencies and related feature

spaces. This issue is discussed in the next section, where we introduce the notion of *goal/concept graphs*. These can be related to other similar concepts, including *goal dependency networks* (*GDN*), proposed by Stepp and Michalski [240], *ontologies*, other related mechanisms like *clause schemata* and *clause structure grammars* used in Inductive Logic Programming (ILP), which are discussed later in this chapter.

8.1.3 Relation Between Dynamic Selection of Bias and Metalearning

Let us examine the relationship between dynamic selection of bias (say via activation of certain concept graphs or ontologies) and metalearning. In the introductory chapter to this book we have stated that learning at the meta level is concerned with the accumulation of experience with the performance of multiple applications of a learning system. Suppose that we have examined one or more related problems and observed that in all of these problems we need to know the values of given attributes. This knowledge is a result of accumulated experience and can be useful when dealing with related problems in the future. Consider, for instance, the problem of credit rating. Once we have identified a good set of attributes, this knowledge can be useful in future similar credit rating tasks. So the knowledge about which attributes are (or are not) relevant when dealing with a particular set of tasks can be regarded as *metadata*. This knowledge also affects the outcome of learning (i.e., whether the concept generated as a result of learning will lead to correct predictions when applied to new unseen cases).

8.1.4 Examples of Some Complex Applications Studied

As we have mentioned earlier, the aim of this chapter is to discuss the problem of learning of *complex systems*, which by definition cannot be learned in one step. The methodology discussed here will be exemplified in several concrete applications, including:

- examples of induction of several interdependent rules (sometimes referred to as *multi-predicate learning*),
- problem of learning individual skills,
- learning to achieve multiple goals in a coordinated fashion,
- learning to attain coordinated behavior of a group of agents.

In all these example applications we will be using the goal/concept graphs to guide the process of learning. Goal/concept graphs represent meta-data/metaknowledge that is shared and exploited by the learning system. Obviously, a question arises about how this knowledge can be acquired. This point will be briefly reviewed in one of the later sections (Discussion).

Goal/
concept
graphs

8.2 Representing Multiple Concepts and Goals as a Graph

In this section we will discuss the issue of *representation of concepts and learning goals*. Typically concepts are defined in terms of subconcepts. These may again be defined in terms of further subconcepts and so on. The concepts can be organized in the form of a graph.

Figure 8.1 shows the concept graph associated with the definition of *uncle* and Figure 8.2 shows the concept graph associated with the definition of *quicksort*. Both definitions follow the conventions common in Inductive Logic Programming (ILP) (see, e.g., [83]). The corresponding definite clauses¹ for the example of family relationships are:

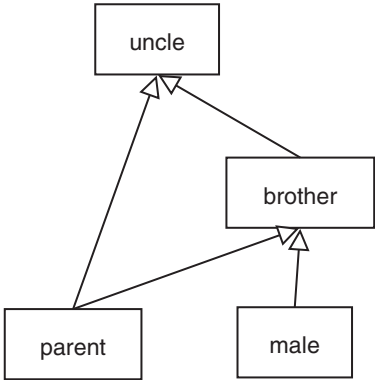


Fig. 8.1. The concept graph for some family relationships

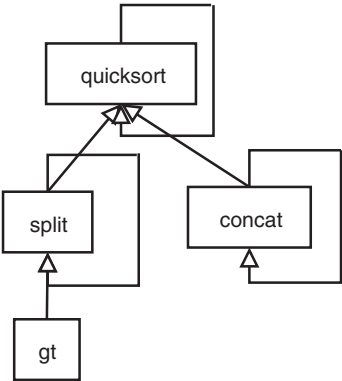


Fig. 8.2. Part of the concept graph showing the dependence of concepts for *quicksort*

¹ A clause is a *definite clause* if it contains exactly one positive literal (here, the literal that is before the symbol $\leftarrow$). See, e.g., [83] for more details.

```
uncle(X,Y) <- parent(Z,Y), brother(Z,Y).
brother(Z,Y) <- parent(V,Z), parent(V,Y), male(Z).
```

We note that each clause in the example above defines a different concept. The first one defines the concept of *uncle/2* (in terms of *parent/2* and *brother/2*), while the second one defines the concept of *brother/2*. The latter rule defines an auxiliary concept used in the concept of *uncle/2*.²

The concept graph in Figure 8.1 can be seen as an abstraction of the information contained in the clauses. The information concerning variable bindings has been abstracted out.

The notion of graphs is, of course, not new. Other researchers have used graphs to represent concepts and interdependencies among them. For example, Morik et al. [177] and Wrobel [285] have used a *rule graph* to represent a given knowledge base graphical form.³ Besides this, there was a possibility to generate an abstracted version in which several predicates could be grouped together and appear thus as a single node. In MOBAL [177] this kind of graph is referred to as *topology*.

Graphs of this kind are very often used merely to *illustrate* which concepts have been learned. Here the aim is to show that the graph can be used to control the process of learning, as we will see later.

8.2.1 Defined Concepts and Learning Goals

We assume that certain basic concepts are known to the system and do not need to be learned. We will call them here *primitive concepts*. For instance, the concepts of *parent/2* and *male/1* are regarded as primitive concepts here. We will distinguish them from other concepts. These have either been defined, learned or acquired in some way or else no definition exists. If no definition is available, these concepts may constitute *learning goals*. Typically, an agent may be aware that some concepts need to be defined before initiating the process of learning and acquiring the definition this way.

Learning
goals

Using subconcepts is very common. For instance, consider the well known definition of *quicksort/2* (similar to the definition in [36]). The concept graph is shown in Figure 8.2.

The corresponding definitions are:

```
quicksort([],[]) <-
quicksort([X|Tail],Sorted) <-
  split(X,Tail,Small,Big),
```

² The concept of *uncle/2* could be defined without recourse to the concept of *brother/2*. For instance, we could just use the primitive concepts (i.e., *parent/2* and *male/1*) in the definition.

³ Recursive occurrence of a predicate, i.e., as both a premise and a conclusion of a rule, will not be represented by an edge ([177], p. 156). As we will see later, this information is relevant in the process of controlling the learning.

```
quicksort(Small,SortedSmall), quicksort(Big,SortedBig),
concat(SortedSmall,[X|SortedBig],Sorted).
```

To be able to use this definition, we also need the definitions of some basic list processing operations, including *split/4*, which decomposes a given list into two parts (sublists *Small* and *Big*) and *concat/3*, which concatenates two lists. In this example, *gt/2*, which determines whether one element is greater than another, is regarded as primitive concept that does not need to be defined.

The process of learning both the target concepts and its subconcepts from examples is usually referred to as *multiple predicate learning*. Our aim in this chapter is to discuss how the concept/goal graph can be exploited to control the process of learning.

Let us ignore for the time being the fact that some concepts are recursive, like the concepts of *split/4* or *concat/3*. A recursive concept can be identified easily in the graph. It has a link to itself.

8.3 Using the Concept Graph to Control Learning

The concept graph places certain restrictions on the process of learning multiple definitions. One obvious possibility is to use a kind of *bottom-up method*. We start at the bottom layer of the graph. The primitive concepts like the concept *parent/2* in Figure 8.1 or the concept *gt/2* in Figure 8.2 do not need to be learned. Then we can move one layer up. In the family relationship example, we would define the concept of *brother/2* as the next learning goal. After acquiring the definition the process would be repeated for the concept of *uncle/2*. As the concept of *uncle/2* requires that the concept of *brother/2* be known, the corresponding definition (of *brother/2*) needs to be added to the background knowledge before attempting to learn the concept of *uncle/2*.

Similarly, in the case of *quicksort/2*, we could start by defining the learning goal as follows. First, learn the definitions of the auxiliary list processing operations *split/4* and *concat/3*. After the corresponding definitions have been added to the background knowledge, we learn the definition of *quicksort/2*.

Anyone who has worked on larger applications will note that the description given above is really an oversimplification of what is normally done. Many books and articles exist on the topic of software development and some are concerned with the problem of how Machine Learning and/or Data Mining methods can be incorporated into the process (for instance, the methodology CRISP-DM [61]). The bottom-up strategy described above can be seen as one step in the development cycle of this methodology.

8.3.1 Learning One Concept

Rules relative to a particular concept can be induced from the given data. According to Easterlin and Langley [84] the process of formation of new concepts involves:

- Given a set of object descriptions (usually presented incrementally)
- find sets of objects that can usefully be grouped together (aggregation) and
- find the intensional definition of these objects (characterization).
- Define a new name (predicate) for the concept and introduce it into the representation, so that it can be used in the definition of further concepts or for the description of future input objects.

Note that this definition assumes that the object descriptions are given. This may not be the case in general if we assume that the agent is situated in an environment. The agent may give specific attention to certain aspects of the environment (and ignore others), particularly if the agent is goal driven. If there is some indication that the objects are potentially related to its goals, the agent has a motivation to gather the corresponding objects (data) and learn from it.

However, the point to be made here is that concept formation in this setting involves a phase of *unsupervised learning* (aggregation) which is followed by learning from preclassified examples, often referred to as *supervised learning*.

The task of acquiring a concept is of course much simplified, if some of these steps have been carried out (e.g. by the user). In learning to classify examples, the given data consists of preclassified examples. Besides, the name of the concept to be acquired is normally also given.

There are many classification algorithms whose description can be found elsewhere [174]. Here we will review just one rule learning algorithm called *sequential covering method*. As it is a well-known algorithm described elsewhere ([174, 83]), we will review it only briefly here. The sequential covering algorithm requires that we provide at least the following three types of information:

Sequential
covering
method

- a set of training examples (the value of the target attribute is given too),
- the target attribute/concept to be learned (e.g., the concept *brother*),
- candidate literals/attributes that can be introduced as conditions in the rule (e.g., the attributes *parent/2* and *male/1*).

Generally the set of attributes is assumed to be given. Later we will discuss various means for controlling the choice of the attributes.

The process of construction of one rule (or clause) using the sequential covering algorithm proceeds as follows. Typically the method will try to learn one clause at a time, while in each step it will try to cover some positive examples.

Each rule (or clause) is generated by including the target concept (e.g., *brother(Z, Y)*) in the clause head. The body is initialized to the empty set of literals. Then the algorithm attempts to add, in each step, one or more literals to the body. This aim is usually to improve a certain measure, such as the difference between the positive and negative examples covered or the

information. After the rule has been generated, the corresponding positive examples covered are marked and the process is repeated for other unmarked examples.

8.3.2 Formulating the Learning Problem as Inverse Deduction

In the literature on Inductive Logic Programming it is habitual to express the problem of induction as the inverse problem of deduction. Given some data D and background knowledge B , learning can be described as the process of generating the hypothesis h that explains D . Let us express this more formally.

Let us assume that D is a set of examples of the form $\langle x_i, f(x_i) \rangle$ where x_i denotes the description of the given example i represented as a set of facts and $f(x_i)$ represents the target value (classification) for that example. Then learning can be expressed as the problem of discovering a hypothesis h , such that for each training instance the classification $f(x_i)$ follows deductively from the hypothesis h together with the description of the example x_i , and the background knowledge B :

$$(\forall \langle x_i, f(x_i) \rangle \in D) (B \wedge h \wedge x_i) \vdash f(x_i)$$

This scheme can be generalized to multiple predicate learning as follows. Let D_j represent the data relative to the learning problem j (e.g., learning the target concept of *brother/2*). Thus each example will be represented in the form $\langle x_{j,i}, f(x_{j,i}) \rangle$. The background knowledge and the hypothesis require a similar index. So the problem above can be formulated as shown below. Instead of indexing the problem using integers, we will use a more convenient term *br* (short for *brother*), representing the target concept, as index here.

$$(\forall \langle x_{br,i}, f(x_{br,i}) \rangle \in D) (B_{br} \wedge h_{br} \wedge x_{br,i}) \vdash f(x_{br,i})$$

$$\begin{aligned} x_{br,1} &: \text{parent(bernard, jara), parent(bernard, ludmila),} \\ &\quad \text{male(bernard)} \\ f(x_{br,1}) &: \text{brother(jara, ludmila)} \\ B_{br} &: \text{empty} \end{aligned}$$

The following hypothesis h_{br} satisfies the description above:

$$h_{br} : \text{brother}(Z, Y) \leftarrow \text{parent}(V, Z), \text{parent}(V, Y), \text{male}(Z).$$

The problem of learning the definition of *uncle* can be formulated similarly:

$$(\forall \langle x_{uncle,i}, f(x_{uncle,i}) \rangle \in D) (B_{uncle} \wedge h_{uncle} \wedge x_{uncle,i}) \vdash f(x_{uncle,i})$$

$$\begin{aligned} x_{uncle,1} &: \text{parent(bernard, jara), parent(bernard, ludmila),} \\ &\quad \text{male(bernard), parent(ludmila, pavel)} \\ f(x_{uncle,1}) &: \text{uncle(jara, pavel)} \\ B_{uncle} &: \text{brother}(Z, Y) \leftarrow \text{parent}(V, Z), \text{parent}(V, Y), \text{male}(Z). \end{aligned}$$

The following hypothesis satisfies the description above:

$$h_{uncle} : uncle(X, Y) \leftarrow parent(Z, Y), brother(Z, Y).$$

We note that the two problems are not independent. In the first phase we induce a rule for the concept of *brother* (in general we may have several rules). This definition is added to the background knowledge in the next phase, when trying to induce a rule for the concept higher up in the concept graph.

8.3.3 Controlling the Domain-Dependent Language Bias

Let us come back to the process of construction of rules using the sequential covering algorithm described earlier. In a simple version of the algorithm the candidate literals that can be added by the system are specified by the user beforehand. As we (and the system) do not know the definition yet, this may often be rather problematic, particularly if we rely just on guessing.

If the set of possible literals is too restricted, the system is unable to arrive at the right definition. If it is very large, the search space will be very large too. So the system may again have difficulty returning the correct solution simply because there are too many candidate definitions to be tried out. This is the reason why some people have proposed to control the language bias using various means, including *determinations* proposed by Davies and Russel in 1987 [72], *relational clichés* [228], *clause schemata* [154], *metapredicates* that define a translation between a *metafact* and a domain-level rule [177] and *topologies* [177], representing abstracted graphs of rules. Other researchers have proposed various approaches based on grammars, including, for instance, the proposal of Cohen [65]. Some of these grammar-based approaches restrict the concepts that can be introduced (e.g., [133]). Others impose restrictions on the variables also, such as the *DLAB* formalism [74].

Domain-
dependent
language
bias

Control of
language
bias
Meta-
predicates
Metafacts

Clause Structure Grammar

In this section we will focus on the *clause structure grammar* of Jorge and Brazdil [133]. We will explain the basic issue using examples.

Suppose we are interested in synthesizing an algorithm for processing structured objects, such as lists. Before doing this, we would want to capture (and exploit) the following idea: *If you want to process a structured object using some procedure (P), decompose it into parts, then invoke the same procedure recursively and then join the partial solutions.* We can conceive a grammar to capture this. In this domain we need three different groups of literals. The first group decomposes certain arguments in the clause head into subterms (e.g., using the predicate *dest/2* separating a list into its *head* and *tail*). The second group enables us to introduce the recursive call. The third group consists of composition literals that enable us to construct the output arguments from other arguments (using the literal *append/3*). In addition we

Clause
structure
grammar

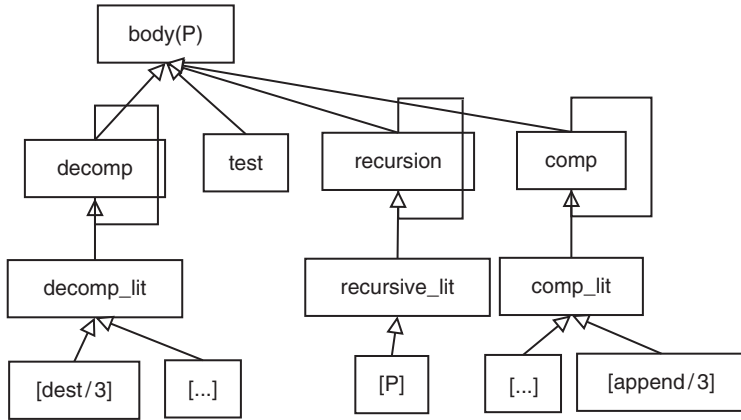


Fig. 8.3. Example of clause structure grammar represented in the form of a concept graph

may also need test literals. So in this domain the general structure of a clause (possibly recursive) is specified as follows:

$$\text{body}(P) \rightarrow \text{decomp}, \text{test}, \text{recursion}(P), \text{comp}$$

where the argument P carries the name of the predicate of the head literal (e.g., *append/3* if we are synthesizing *append*). The decomposition group is defined as a sequence of decomposition literals:

$$\begin{aligned} \text{decomp} &\rightarrow \text{decomp_lit}. \\ \text{decomp} &\rightarrow \text{decomp_lit}, \text{decomp}. \end{aligned}$$

The recursion group is defined as:

$$\begin{aligned} \text{recursion}(P) &\rightarrow \text{recursive_lit}(P). \\ \text{recursion}(P) &\rightarrow \text{recursive_lit}(P), \text{recursion}(P). \\ \text{recursive_lit}(P) &\rightarrow [P]. \end{aligned}$$

The individual predicates are introduced using rules like this one:

$$\text{decomp_lit} \rightarrow [\text{dest}/3].$$

The grammar can be represented in the form of a graph, such as the one shown in Figure 8.3.

8.3.4 Activation of Domain-Specific Metaknowledge/Ontology

Clause structure grammar can be regarded as a kind of domain-specific meta-knowledge useful in induction. This type of knowledge captures what has been acquired in the course of elaborating solutions of inductive problems in a particular domain. What interests us is that this knowledge can be reusable in

new settings. The term *domain-specific metaknowledge* implies that we conceive really different schemes for different types of problems. So, for instance, if the aim is to conceive a new procedure for list processing, we would activate the appropriate domain-specific metaknowledge. In other words, we would activate an ontology of possible concepts that the learning system could deploy when conceiving the solution. Domain-specific meta-knowledge

If the problem were different, say, if the aim were to elaborate a procedure for solving equations with several variables, we would again have to activate the appropriate domain-specific metaknowledge (ontology). This may be rather different from the previous example. Ontology

Given new problem, a question arises as to which domain-specific metaknowledge should be brought into play. The answer to this lies, we believe, in the problem itself. Suppose we have conceived several different ontologies permitting us to deal with many diverse problems. Suppose we have a new problem and need decide what to do. One way to address this is to determine the class of the problem and then activate the appropriate domain-specific metaknowledge (ontology).

This process can be seen as the problem of activating Minsky's frames [172]. However, when the proposal on frames was written, the area of Machine Learning was not yet very advanced and so the issue of learning to invoke frames was not really discussed. Determining the type of problem encountered can be regarded as a problem of *classification*. Classifying a problem could be regarded as something comparable to the task of classifying a given document. The techniques in this area are well advanced now and so this issue does not seem to constitute a serious problem.

However, it needs to be added that the research in this area has not yet advanced to a mature state. As we have pointed out earlier, various researchers did propose various schemes to control the language bias. The experiments were usually rather limited. They confirmed that this type of methodology could be useful and various promising results were reported, but so far, no results of large-scale comparative studies have been reported.

8.3.5 Learning Recursive Definitions: Iterative Process of Exploiting Concept Graphs

Let us analyze the issue of learning recursive concepts in more detail and show how this can be done with the help of a given concept graph. Learning recursive definitions is an important issue, as many concepts are best represented this way. Learning recursive concepts

Various approaches have been proposed in the literature, including the systems RTL [12], SKILit [133] and CRUSTACEAN [2] among others. Here we will focus on one method mentioned above (SKILit) that is able to synthesize the correct definitions with a relatively high probability on the basis of few randomly chosen examples. The method does not assume any a priori knowledge of the solution except the domain-specific knowledge captured in

Table 8.1. Iterative induction illustrated on an example

<i>positive examples</i>	<i>T1: first step theory (useful properties)</i>	<i>T2: second step theory T1 is background knowledge</i>
member(3, [4,1,3])	member(A, [C, A E])	member(A,[C, A E])
member(2, [1,2,3])		member(A,[C D]) ← member(A,D).

the form of concept graphs. The method can thus be seen as a natural extension of the method described earlier that exploits the given concept graph to control learning.

As described earlier, the method starts learning at the bottom layer of the graph. However, we note that the recursive concepts contain a link to themselves (as in Figure 8.2). Mutually recursive concepts involve larger cycles in the graph. So, a question arises about how to adapt the method described to learn such concepts. Our objective in this section is to describe just this.

The solution consists of repeating the learning cycle more than once, wherever a link exists to itself in the concept graph. In each step a tentative theory is produced and reused, as background knowledge, in the next cycle of the induction process. If the stopping criterion is satisfied, the process terminates.

Let us examine how the system generates a definition of *member/2* on the basis of only two positive examples shown in Table 8.1. Let us assume that the appropriate negative examples and background knowledge have also been given.

In the first iteration theory *T1* is induced. This theory generalizes one of the positive examples given. If we use *T1* together with the background knowledge and call the system again, we obtain theory *T2*. This definition is correct, although it is more specific than the usual one. The system stops in the following step since no new clauses appear in *T3*.

The clause in *T1* represents a useful property of the *member/2* relation. The term *properties* is used here to refer to (apparently) valid statements that do not necessarily appear in the final target definition. The system succeeded in generating the correct definition thanks to this property. To generate a recursive clause covering the example *member(3, [4, 1, 3])*, the system needs the fact *member(3, [1, 3])* corresponding to the recursive call. Although it does not appear among the examples given, it is implied by the property generated earlier (*member(A, [C, A|E])*). In other words, theory *T1* introduced a crucial fact which made possible the induction of recursion.

Building a Sequence of Theories with Iterative Induction

In general, the method of iterative induction proceeds as follows (see Algorithm 8.1 describing procedure SKILit). The system starts with theory *T0*, which is empty. Besides the positive examples, the system uses a set of

Iterative
induction

negative examples and the background theory (definitions of auxiliary predicates). In the first iteration the system invokes the basic induction system to create theory $T1$.⁴ The clauses in $T1$ generalize some positive examples and are typically non-recursive. In general it is hard to introduce recursion at this level due to the lack of crucial positive examples in the data. Thus, it is likely that the clauses in $T1$ are defined with predicates from the background knowledge only.

In the second iteration theory $T2$ is induced. Recursion is more likely to appear here, since the crucial examples that were missing may be covered by $T1$. Likewise, more facts are covered by $T2$, which means that new, interesting clauses may appear in subsequent iterations. The process stops when one of the iterations does not introduce new clauses. Throughout the process, the clauses covering negative examples are discarded.

Procedure SKILit

```
input:  E+, E- (positive and negative examples),
        BK (background knowledge)
output: T (a theory) i:= 0 T0 = {} repeat
        Ti+1 := SKIL(E+, E-, Ti, BK)
        i:= i+1
until Ti+1 does not contain new clauses
return Ti+1
```

Each theory in a given iteration is generated with the help of system SKIL. It employs a covering strategy as many other systems do. The interested reader can find more details elsewhere [133].

Example: Generation of a Definition of Insertion Sort

Let us examine how the method of iterative induction helped to synthesize the definition of insertion sort (*isort*/2). First, let us see which predicates were made available to the system as domain-specific metaknowledge.

Initially, these include some basic list handling predicates including *dest*/3, which decomposes a list into the head and the rest, *const*/3, which constructs a new list by adding an element to a given list, and *null*/1, which returns *true* if the given list is null. Apart from this, the system was also given other list handling predicates that are needed for this task, including *split*/4 and *concat*/3, discussed in one of the earlier sections. In addition, the authors also used *=*/2, which checks for equality and the predicate *<*/2 (less than) here. Although it is not used in the final definition of *isort*/2, this predicate is necessary for the generation of useful properties. Some properties generated in the first step of iterative induction are shown below:

⁴ The authors used system SKIL here, but in principle other systems (e.g., ALEPH) could have been used too.

$$\begin{aligned} \text{isort}([A, B], [A, B]) &\leftarrow A < B. \\ \text{isort}([A, B], [B, A]) &\leftarrow B < A \end{aligned}$$

The properties induced by the system represent a correct (but specific) program for sorting 2-element lists. The first clause establishes that if the list is already sorted (i.e., the first element is smaller than the second one), the order should be maintained. The second clause takes care of swapping the two elements when necessary. It is easy to see that both properties generalize many concrete examples. Thanks to these discovered properties, the system was able to generate the correct definition in the next step.

The method described was able to synthesize the correct definitions (with relatively high probability) of various predicates on the basis of a few examples. The definitions included predicates like *append/3* (join two lists), *delete/3* (delete an element from the list), *rv/2* (reverse the given list), *member/2* and *last_of/2* (identify the last element of the given list), among others. The examples were chosen randomly from a predefined set, without assuming a priori knowledge of the solution. The accuracy was evaluated on an independent test set. Overall the accuracies were of the order of 90% or more when only five positive examples were given.

8.4 Exploiting Concept Graphs in Other Applications

In this section we will analyze several other application domains and demonstrate that the basic methodology described earlier can be exploited. In all these examples we will discuss the role of the domain-specific metaknowledge (concept graphs) and show how it can be used to facilitate the process of learning.

8.4.1 Learning Individual Skills

In this section we will analyze the issue of learning individual skills. We will be concerned with the issue of how to control this process to make it more effective. We will address in particular the role of language bias. But first let us see what is meant by the term *skill*.

One useful presupposition is that agents acquire low-level skills before acquiring (and exhibiting) more complex behaviors. Having certain skill involves executing a certain action, or a sequence of actions, in the right manner.

If we are controlling a device such as a simulated plane, this may involve deciding what to do with regard to a particular control variable whose value we can change. For instance, we may decide to increase the thrust.

Let us consider another example from simulated robotic soccer. Suppose the aim is to learn to intercept a moving ball, as described by Stone [241]. The defender needs to consider where the ball is and determine the appropriate actions (e.g., turn and dash in the right direction) and the parameters of the actions (e.g. how much to turn).

Learning skills is more effective when it is done off-line. Learning off-line means that we focus on learning a particular skill without considering how it is used afterwards. Learning is relatively easy if the action is fixed and the aim is to determine the right value of a particular control variable. Learning off-line can involve either behavioral cloning or active experimentation.

In the framework of *behavioral cloning*, or *learning by imitation*, the assumption is that a skilled human is available, capable of performing the given task. This scheme was used by Sammut et al. [213] and Camacho et al. [49] for the task of learning to fly a simulated plane. The actions of the human pilot concerning *thrust* (or *flaps*, etc.) were recorded together with all state variables. Each case then represented a training example that was used to train an appropriate classification or regression model (e.g., a regression tree or a neural net) to determine the right value of a particular control variable.

Instead of having a skilled person show what to do and when, the system can use experimentation instead (as in [241]). Let us consider the case of learning to catch a ball, which in the particular simulated robot setting referred to before amounts to learning to determine the appropriate *turn angle* in a given situation. Initially the right value of the turn angle is not known and therefore various values are selected by a random process and tried out.

As with earlier examples, the learning problem can be constrained by the language bias, that is, by determining the concepts that should be taken into account and their interrelationships. We will use two examples to illustrate this. The first one is concerned with learning to control a simulated aircraft and the second one is from robotic soccer.

8.4.2 Learning to Control a Simulated Aircraft

Our first example is concerned with the problem of learning to control a simulated aircraft [49]. This involves learning to control a set of control devices (shortly, *controls*) including, for instance, the *ailerons*, *elevators* and *thrust*.⁵ The defender needs to consider where the ball is, determine the *turn angle*, *turn* and *dash forward* trying to catch the ball. So learning a skill involves considering the given state and determining which actions to perform (if there is a choice) and determining the parameters of the actions (such as the right *level of thrust*, or the right *turn angle*). The general scheme for learning one of the controls is illustrated in Figure 8.4.

Early approaches tried to learn to associate a particular action with a particular state. As Camacho et al. [49] and others have shown, this approach does not generalize too well. They have proposed a two-phase scheme. In the first phase the given state and goals are analyzed to determine whether

Learning to
control a
device

⁵ *Ailerons* are movable parts at the end of the wing that enable us to control left-right inclination of the plane. *Elevators* are movable parts alongside the wing that enable us to control the inclination of the nose of the plane. They also affect the left-right inclination.

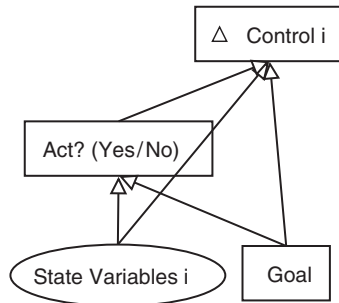


Fig. 8.4. Concept graph for learning whether to change controls

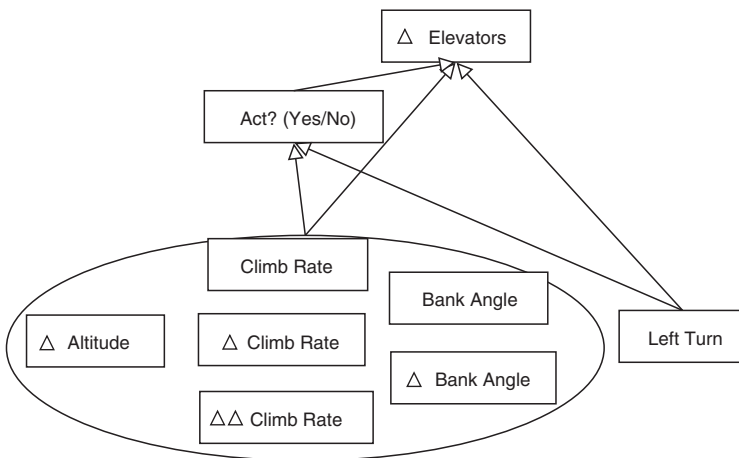


Fig. 8.5. Concept graph for learning to control *elevators*

it is *necessary to change* any of the controls. If the test does not demand any change, the system can just keep going as before. If however, the test demands that something should be changed, the system tries to determine what to do. This involves considering different controls and determining *by how much* these should be adjusted. In Figure 8.4 this change is represented by $\Delta Control_i$. Note that the figure includes the *goal* of the agent.⁶

In the rest of this section we will focus on the problem of learning how to operate one of the controls only and postpone the discussion of how to deal with several controls till later.

Figure 8.5 shows how the schema above is applied to the problem of learning *elevator control*. Here the change to be applied to *elevator control*

⁶ The goal of the agent (e.g., take a plane to a certain location) should not be confused with the learning goal(s) (e.g., learn how to adjust the elevator control of the simulated aircraft).

is represented by $\Delta Elevators$. Learning this concept is conditioned by the decision about whether it is necessary to alter this control in the first place. Besides this, it is also conditioned by the current goal, that is, *left turn*. The decision about whether to act or not depends on various *state variables*, such as $\Delta Altitude$ and $ClimbRate$. Figure 8.5 shows various state variables considered in this application (they are surrounded by an ellipse).

Let us now consider how metalearning can help in the conceptualization of this problem. For us, humans, it is clear that if some control (*elevators* in our case) affects basically a *vertical position*, then the state description should include those variables related to this concept. In our case these are the variables $\Delta Altitude$, $ClimbRate$ etc. So a recognition of a certain type of task (e.g., vertical control) should bring into play an appropriate ontology. If such a mapping has been established on one problem, it can be recovered and reused in another problem. That is, past domain-specific metaknowledge can be recovered and re-used in similar settings.

In the example discussed here all variables deemed relevant were simply defined by the user. That is, the user identified the relevant domain-specific metaknowledge that is pertinent to the problem at hand and introduced it manually into the system. Our aim in this chapter is to describe methods that could do this for us.

8.4.3 Learning a Skill in Simulated Soccer

Let us now analyze the second example, from robotic soccer [241]. The aim is for the defender to learn to intercept the ball. The defender needs to consider where the ball is, determine the *turn angle* and *turn* and *dash forward* trying to catch the ball. So learning a skill involves considering the given state and determining which actions to perform and the parameters of the actions.

Learning a skill

The rest of the behavior does not need to be learned. The simulated player will just *turn* and *dash forward* trying to intercept the ball. Not every attempt is successful. In general the agent needs to recognize whether or not the action was successful. To speed up learning the authors used a special centralized “omniscient agent” to provide this information. This agent classifies each trial as success or failure, depending on whether the ball was stopped or got past the defender. Each trial then serves as an example to train a model.

Let us represent the concepts again in the form of a concept graph. The result is shown in Figure 8.6. The meanings of the abbreviations used are as follows:

- *DefenseAction*: classification of the defense action (success or failure),
- *TurnAng_t*: angle that the defender should turn at time t (it is assumed that the player should dash forward after turning),
- *BallDist_t*: distance from the defender to the ball at time t ,
- *BallAng_t*: angle determining where the ball is relative to the defender at time t .

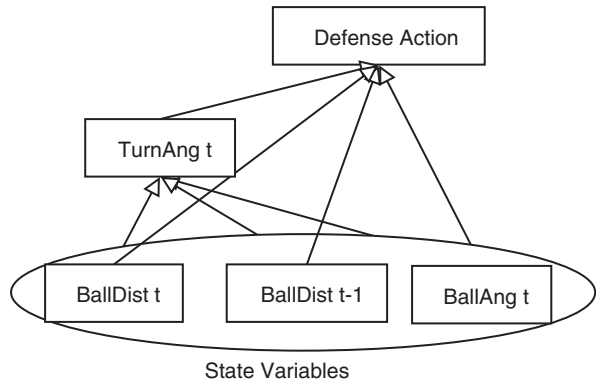


Fig. 8.6. Concept graph associated with learning to intercept a ball

The values of ball distance at time t ($BallDist_t$) and at time $t - 1$ ($BallDist_{t-1}$) permit us in effect to calculate the velocity of the ball. This concept is not explicitly represented in the figure. The author reports results of a series of experiments in which the shooter's position was varied. In a particular setting that was investigated the system needed about 500 examples to acquire quite good competence. In the experiments reported a Neural Net was used. The ball was intercepted in about 90% of the cases.⁷

The learning problems just described involved learning a single control parameter (turn angle). Other problems may involve learning several parameters of an action or even several coordinated actions. That is, in the context of trying to intercept a ball, we may endow the system with the ability to control not only the turn angle, but also the speed of dashing. But let us delay the discussion of such learning to coordinate actions till later.

8.4.4 Using Acquired Skills in Learning More Complex Behavior

Earlier we have presented the notion of concept graph and have shown how this can be used to control the process of learning. Typically, lower-level concepts would be learned before learning the concepts higher up. Stone [241] calls this strategy *layered learning* and applies it to learning skills followed by learning more complex actions. The acquired skills are considered as primitive actions. After these have been learned the system proceeds to learn more complex actions.

Let us analyze an example used in [241] which is concerned with passing a ball to another agent. The receiving agent must intercept the ball. This task

⁷ The performance was not 100% because the system was programmed to simulate various imperfections that exist in the real world. For instance, the system did not provide perfect information concerning position and angle. Also, the actions did not always produce the desired effects. This was intentional.

Learning
complex
behavior

Layered
learning

is identical to the problem discussed earlier and hence the learned ability can be reused. As there are several players in the field, the passer must decide to whom the ball should be passed. Here, the identifier of the player can be regarded as the parameter that needs to be learned.

The passer announces his intention to pass and the teammates reply when they are ready to receive. The passer chooses a receiver randomly during training and announces to whom it is passing. The receiver and four nearest opponents attempt to get the ball using the learned interception skill. The training example is classified as success if the receiver manages to advance the ball towards the opponent's goal, and failure otherwise. Many features are collected and stored with each training instance, permitting us to improve the decisions of the passer. The features are of two types:

- $\text{dist}(x,y)$: the distance between players x and y ,
- $\text{ang}(x,y)$: the angle to player y from player x 's perspective.

These concepts are then applied to various players. For instance, distances and angles to other teammates are given. Similarly, the system uses distances and angles to players from the opponent team. Besides these, the system uses derived features such as *relative angle*, defined as:

$$\text{rel_angle}(\text{passer}, k, \text{receiver}) = |\text{ang}(\text{passer}, k) - \text{ang}(k, \text{receiver})|$$

Some attributes are obtained by summarizing a set of values using the functions *min* (or *max*). This is useful, for instance, when passing a ball. We may want to identify a receiver that can safely receive the ball. One way of finding this is by establishing the distances to surrounding opponent players and identifying the nearest one. The nearest distance represents a kind of aggregate feature.

Let us consider again where the basic (and derived) features come from. As we are dealing with control in two-dimensional space, it is natural that an ontology employing *distances* and *angles* be used.

The authors have chosen a decision tree model because of its ability to leave out irrelevant features. The decision tree constructed can be used to improve the decision making. The system can consider different options when passing the ball. The decision tree can be used not only to estimate whether this will succeed or not, but also to select the best option. This is due to the fact that decision trees can provide not only the most probable class, but also the confidence estimate of this classification.

The authors report that overall the system achieved a success rate of 65%, which is better than 51% success rate achieved when a receiver is chosen randomly. The performance can be increased further (up to 79%) if the passer is given other options, besides just passing the ball. That is, if there are no conditions to pass, the agent may decide to continue to dribble the ball.

8.4.5 Learning Coordinated Actions

Learning
coordi-
nated
actions

Earlier we have discussed learning a simple action (skill) or a more complex action which employs already learned behaviour. Unfortunately not all learning can be explained using this scheme. There are situations where we need to control two or more processes in a coordinated manner.

Let us consider some situations when this occurs. Let us consider for instance how we start driving a car (assume it is a car without automatic gear change). Suppose we are just about to move. We need to keep releasing the clutch and pressing the accelerator. Both actions need to be carried out in a coordinated manner, as otherwise the car will stall.

Let us consider another example. Reconsider the problem of control of a simulated plane. As we have pointed out earlier, many maneuvers involve more than one control. For instance, if the aim is to turn left, we need to determine whether to adjust not only the *elevators* (as in Section 8.4.2), but also the *ailerons* and the *thrust*.

Let us analyze a situation discussed by [49] which involves two controls, control *i* and control *j*, which affect one another. The general situation is illustrated in Figure 8.7. The interdependence of the two controls is illustrated by a link interconnecting them.

Earlier we have discussed a strategy for learning multiple concepts. A question that we will address here is how to adapt this strategy for learning interdependent goals.

If we were to ignore the effect of control *j* (*ailerons*), learning the change of control *i* (*elevators*) at time *t* would involve the state variables at time *t-1*. In addition we would need to consider the goals and whether there is a need to act (values at time *t-1*). As our aim is to capture the effect of other controls, we need to add the relevant information to the model. Here we need to add information about the change of control *j* (*ailerons*) used at time *t-1*.

A similar approach is adopted when learning the change of control *j* (*ailerons*). The information used involves the state variables at time *t-1*,

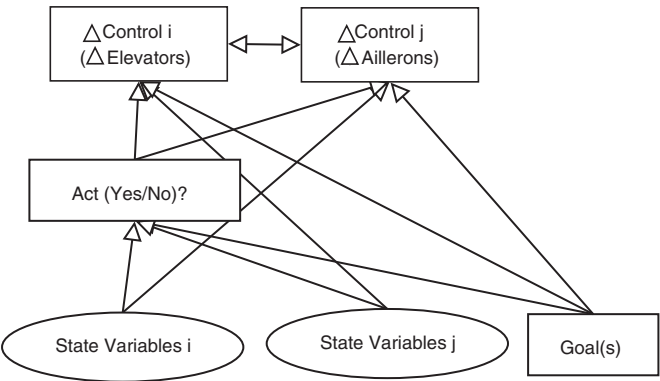


Fig. 8.7. Concept graph with two interdependent controls

the current goal and the current value of the change of control i (*elevators*) at time $t-1$.

The method outlined was validated using extensive experiments [49]. It was shown that if we follow the strategy outlined, we can acquire the ability to deal with several controls at the same time in a coordinated manner. The approach follows the basic methodology of bottom-up learning (or layered learning) discussed earlier. Due to interdependance of concepts, the approach can be regarded as a variant of iterative induction/closed-loop learning. Some concepts learned are used as input in the next phase of learning.

8.5 Summary and Future Challenges

8.5.1 Summary

The aim of this chapter was to complement the discussion in previous chapters of this book. Chapters 1 through 3 are concerned with the issue of how to select a suitable ML/DM algorithm for a given dataset. Our aim was to show that metaknowledge concerning algorithms and their performance on different problems can be very useful in this process. Chapter 4 extended this by showing how metalearning can be extended to Data Mining and KDD. It is clear that the issue is not really *how to select a particular ML/DM algorithm*, but rather *how to determine the operations that achieve a certain goal*. That is, the problem can be seen as a problem of *planning to achieve a given goal*. Again, our point is that metaknowledge about past solutions can be useful, as it can suggest which solutions can be reused. For instance, the system can recognize that a particular chain of operations is useful in a new setting. Chapter 7 was dedicated to the related issue of transfer of (meta)knowledge across tasks.

The aim of this chapter was also to introduce another aspect. We have drawn attention to the fact that if several problems share a similar conceptual structure that has been elaborated for one of the problems, it can be reused in other similar settings. We have referred to this structure as *domain-specific metaknowledge*. The conceptual structure was represented in the form of a concept/goal graph which, as was shown, can be used to control the learning process. We have provided examples from various domains illustrating how this was done. These included the problem of learning recursive definitions and interdependant concepts, which are considered out of scope of some systems, but are nevertheless very useful.

In this presentation we have omitted the discussion of how the concept/goal graphs could be created, modified or extended. This is intentional, as the issue of how to introduce new concepts can be found elsewhere. Besides, it would distract us from the main theme of this chapter, which is how domain-specific metaknowledge can be reused and in that way complement the material presented in previous chapters.

8.5.2 Future Challenges

In this section we mention some directions for future work. The list is by no means exhaustive.

How to Adapt the Metalearning Approach to Other Domains

It is interesting to consider how the metalearning approach could be adapted or extended to problems in other domains, such as operations research. This may involve not only selection of a suitable algorithm (e.g., some particular GA method) but also its parameterization. One research issue related to this is which measures should be used for characterizing different tasks and indexing different solutions.

Which Experiments Should Be Conducted When Constructing a Metaknowledge Base

The method presented in this book presupposes the existence of a metaknowledge base which captures, in effect, results of previous experiments. This is feasible only if the number of algorithms is relatively limited. So a question arises as to what should be done in general. It would seem that some solution that strikes a good balance between *exploration* and *exploitation* adopted in reinforcement learning (RL) could be useful here. If we draw an analogy between *algorithms* and *operators* in RL, it follows that more attention should be given to more promising algorithms when exploring the space.

How to Exploit Existing Ontologies in Learning

A great deal of effort is dedicated to the construction of ontologies in different domains. These are useful in communication across platforms and systems, which is often the motivation for constructing them. However, they are also useful in learning. As was pointed out in this chapter, the type of the problem encountered could be used to determine the initial ontology to be adopted for gathering the data. It is conceivable that it may be necessary not only to recover an existing ontology, but also to adapt or extend it to new settings. If such an ontology did not exist initially, it would have to be constructed for a particular problem at hand. Ontologies useful in one task could also be of use in another similar task and hence facilitate the transfer of knowledge across tasks, as discussed in Chapter 7.

Ontologies are also useful in the process of revision or update of existing models, as is well known. Let us consider, for instance, a rule learning system. If some rule is found too general (that is, if it covers negative examples), it can be specialized. A given ontology can be used to suggest how a particular condition can be specialized. This involves identifying the item in the hierarchy

and retrieving the more specific terms below. A similar method can be used if it is necessary to generalize a rule.

Although ontologies have already been used in previous work in ML, more work is needed to determine the details of the proposal concerning reuse of a conceptual structure in new settings. It is also necessary to see whether this could, in fact, save effort when tackling real-world problems.

How to Exploit the Work on Planning

Quite a large body of work exists on planning. A question is whether some of the techniques could be adapted or exploited for the planning to achieve a complex learning goal. As in physical domains, a complex learning goal can be decomposed into subgoals, and normally the individual subgoals are achieved by applying appropriate operators in an appropriate order (often the order is only partially constrained).

Although some systems presented in Chapter 4 are able to come up with rather complex solutions involving several operators, there is still a long way to go; a great deal of the existing work on planning has not yet been properly integrated and exploited. In our view this is one of the very promising directions for further research.

The task is by no means simple, for several reasons. First, we are normally interested not only in achieving good performance, but also in controlling the costs. The benefit/cost function typically involves at least two variables. Second, the outcome of each learning action is somewhat uncertain and so the planning system has to be able to model uncertainty. Third, we have only partial knowledge about the given state (determining which learning action is possible at that point) and hence it may be necessary to use actions concerned with *information gathering* which need to be properly integrated with learning.

Exploiting the existing knowledge on planning in learning, however, has potential advantages, as it may help to answer various research questions. One of these is, can we determine that a particular plan to learn will probably not be successful? If it were possible to determine that, then this knowledge could be used to trigger a *shift of bias*, and further phase of replanning and relearning. The new run can involve, for instance, other types of domain-specific metaknowledge, or other ML/DM algorithms that were not considered in the earlier run.

Planning to learn

Shift of bias

How to Control the Process of (Re)learning in Dynamic Environments

Dynamic environments represent yet another challenge as we cannot assume that the data is fixed. As the environment changes, it provides the system with a continuous stream of data. So even if we had a perfectly working model at some stage, there is no guarantee that the model will continue to

be satisfactory in the future. Typically, we would want to build a model that achieves a good performance and maintains it. Some solutions to this problem were discussed in Chapter 6.

Note that the task here can be seen as a *problem of control*. The system should not only achieve good performance after the initial phase of learning, but also keep it up over time. If a drop in performance is detected, the system should initiate a corrective action. Note that the system needs to carry out specific information gathering actions to be able to decide that. Despite the fact that some work has already been done in this area, the challenge is how to do this effectively. That is, the issue is how information gathering should be properly integrated with replanning and relearning.

References

1. D. W. Aha. Generalizing from case studies: A case study. In D. Sleeman and P. Edwards, editors, *Proceedings of the Ninth International Workshop on Machine Learning (ML92)*, pages 1–10. Morgan Kaufmann, 1992.
2. D. W. Aha, S. Lapointe, C. X. Ling, and S. Matwin. Inverting implication with small training set. In F. Bergadano and L. De Raedt, editors, *Machine Learning: ECML-94, European Conference on Machine Learning, Catania, Italy*, volume 784 of *Lecture Notes in Artificial Intelligence*, pages 31–48. Springer, 1994.
3. E. Alpaydin. *Introduction to Machine Learning*. MIT Press, 2004.
4. E. Alpaydin and C. Kaynak. Cascading classifiers. *Kybernetika*, 34:369–374, 1998.
5. P. Andersen and N. C. Petersen. A procedure for ranking efficient units in data envelopment analysis. *Management Science*, 39(10):1261–1264, 1993.
6. D. Andre and S. J. Russell. State Abstraction for Programmable Reinforcement Learning Agents. In *Eighteenth National Conference on Artificial Intelligence*, pages 119–125. AAAI Press, 2002.
7. A. Argyriou, T. Evgeniou, and M. Pontil. Multi-Task Feature Learning. In *Advances in Neural Information Processing Systems*, 2006.
8. A. Asuncion and D.J. Newman. UCI machine learning repository, 2007.
9. C. G. Atkeson, A. W. Moore, and S. Schaal. Locally weighted learning. *Artificial Intelligence Review*, 11(1-5):11–73, 1997.
10. B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In Phokion G. Kolaitis, editor, *Proceedings of the 21st Symposium on Principles of Database Systems*, pages 1–16. ACM Press, 2002.
11. B. Bakker and T. Heskes. Task Clustering and Gating for Bayesian Multitask Learning. *Journal of Machine Learning Research*, 4:83–999, 2003.
12. C. Baroglio, A. Giordana, and L. Saitta. Learning mutually dependent relations. *Journal of Intelligent Information Systems*, 1:159–176, 1992.
13. M. Basseville and I. Nikiforov. *Detection of Abrupt Changes: Theory and Applications*. Prentice Hall Inc., 1993.
14. J. Baxter. Learning Internal Representations. In *Advances in Neural Information Processing Systems, NIPS*. MIT Press, Cambridge MA, 1996.
15. J. Baxter. Theoretical models of learning to learn. In S. Thrun and L. Pratt, editors, *Learning to Learn*, chapter 4, pages 71–94. Springer-Verlag, 1998.

16. J. Baxter. A Model of Inductive Learning Bias. *Journal of Artificial Intelligence Research*, 12:149–198, 2000.
17. S. Ben-David and R. Schuller. Exploiting Task Relatedness for Multiple Task Learning. In *Sixteenth Annual Conference on Learning Theory*, pages 567–580, 2003.
18. K. P. Bennet and C. Campbell. Support vector machines: Hype or hallelujah. *SIGKDD Explorations*, 2(2):1–13, 2000.
19. H. Bensusan. God doesn't always shave with Occam's razor - learning when and how to prune. In *ECML '98: Proceedings of the 10th European Conference on Machine Learning*, pages 119–124, London, UK, 1998. Springer-Verlag.
20. H. Bensusan and C. Giraud-Carrier. Discovering task neighbourhoods through landmark learning performances. In D. A. Zighed, J. Komorowski, and J. Zytkow, editors, *Proceedings of the Fourth European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD2000)*, pages 325–330. Springer, 2000.
21. H. Bensusan and A. Kalousis. Estimating the predictive accuracy of a classifier. In P. Flach and L. De Raedt, editors, *Proceedings of the 12th European Conference on Machine Learning*, pages 25–36. Springer, 2001.
22. Hilan Bensusan, Christophe Giraud-Carrier, and Claire Kennedy. A higher-order approach to meta-learning. In *Proceedings of the ECML'2000 workshop on Meta-Learning: Building Automatic Advice Strategies for Model Selection and Method Combination*, pages 109–117. ECML'2000, June 2000.
23. A. Bernstein and F. Provost. An intelligent assistant for the knowledge discovery process. In *Proceedings of the IJCAI-01 Workshop on Wrappers for Performance Enhancement in KDD*, 2001.
24. A. Bernstein, F. Provost, and S. Hill. Towards Intelligent Assistance for a Data Mining Process. *IEEE Transactions on Knowledge and Data Engineering*, 17(4):503–518, 2005.
25. H. Berrer, I. Paterson, and J. Keller. Evaluation of machine-learning algorithm ranking advisors. In P. Brazdil and A. Jorge, editors, *Proceedings of the PKDD2000 Workshop on Data Mining, Decision Support, Meta-Learning and ILP: Forum for Practical Problem Presentation and Prospective Solutions*, pages 1–13, 2000.
26. C. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.
27. H. Blockeel, L. De Raedt, and J. Ramon. Top-down induction of clustering trees. In *ICML '98: Proceedings of the Fifteenth International Conference on Machine Learning*, pages 55–63, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.
28. A. Blumer, D. Haussler, and M. K. Warmuth. Learnability and the Vapnik Chervonenkis Dimension. *Journal of the ACM*, 36(1):929–965, 1989.
29. J. A. Botía, M. Garijo, J. R. Velasco, and A. F. Skarmeta. A generic data mining system: Basic design and implementation guidelines. In *Proceedings of the KDD-98 Workshop on Distributed Data Mining*, 1998.
30. J. A. Botía, A. F. Gómez-Skarmeta, M. Garijo, and J. R. Velasco. A proposal for meta-learning through a multi-agent system. In *Proceedings of the Agents Workshop on Infrastructure for Multi-Agent Systems*, pages 226–233, 2000.
31. J. A. Botía, A. F. Gómez-Skarmeta, M. Valdés, and A. Padilla. METALA: A meta-learning architecture. In *Proceedings of the International Conference, 7th Fuzzy Days on Computational Intelligence, Theory and Applications, LNCS 2206*, pages 688–698, 2001.

32. J. A. Botía, J. M. Hernansaez, and A. F. Gómez-Skarmeta. METALA: A distributed system for web usage mining. In *Proceedings of the Seventh International Work-Conference on Artificial and Natural Neural Networks (IWANN-03)*, LNCS 2687, pages 703–710, 2003.
33. O. Bousquet and A. Elisseeff. Stability and Generalization. *Journal of Machine Learning Research*, 2:499–526, 2002.
34. R. J. Brachman and T. Anand. The process of knowledge discovery in databases. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, chapter 2, pages 37–57. AAAI Press/The MIT Press, 1996.
35. D. Brain and G. Webb. The need for low bias algorithms in classification learning from large data sets. In T. Elomaa, H. Mannila, and H. Toivonen, editors, *Principles of Data Mining and Knowledge Discovery PKDD-02*, LNAI 2431, pages 62–73. Springer Verlag, 2002.
36. I. Bratko. *Prolog Programming for Artificial Intelligence*, 3rd edition. Addison-Wesley, 2001.
37. P. Brazdil. Data Transformation and Model Selection by Experimentation and Meta-Learning. In *Proceedings of the ECML-98 Workshop on Upgrading Learning to Meta-Level: Model Selection and Data Transformation Learning to Learn*, pages 11–17. 1998.
38. P. Brazdil, J. Gama, and B. Henery. Characterizing the applicability of classification algorithms using meta-level learning. In F. Bergadano and L. De Raedt, editors, *Proceedings of the European Conference on Machine Learning (ECML94)*, pages 83–102. Springer-Verlag, 1994.
39. P. Brazdil and R. J. Henery. Analysis of results. In D. Michie, D. J. Spiegelhalter, and C. C. Taylor, editors, *Machine Learning, Neural and Statistical Classification*, chapter 10, pages 175–212. Ellis Horwood, 1994.
40. P. Brazdil and C. Soares. A comparison of ranking methods for classification algorithm selection. In R. L. de Mántaras and E. Plaza, editors, *Machine Learning: Proceedings of the 11th European Conference on Machine Learning ECML2000*, pages 63–74. Springer, 2000.
41. P. Brazdil, C. Soares, and J. Pinto da Costa. Ranking learning algorithms: Using IBL and meta-learning on accuracy and time results. *Machine Learning*, 50(3):251–277, 2003.
42. P. Brazdil, C. Soares, and R. Pereira. Reducing rankings of classifiers by eliminating redundant cases. In P. Brazdil and A. Jorge, editors, *Proceedings of the 10th Portuguese Conference on Artificial Intelligence (EPIA2001)*. Springer, 2001.
43. L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.
44. K. Brinker and E. Hüllermeier. Case-based multilabel ranking. In *IJCAI*, pages 702–707, 2007.
45. C. E. Brodley. Recursive automatic bias selection for classifier construction. *Machine Learning*, 20:63–94, 1995.
46. G. Brown. Ensemble learning – on-line bibliography. <http://www.cs.bham.ac.uk/~gxb/ensemblebib.php>.
47. B. Brumen, I. Golob, H. Jaakkola, T. Welzer, and I. Rozman. Early assessment of classification performance. *Australasian CS Week Frontiers*, pages 91–96, 2004.
48. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2:121–167, 1998.

49. R. Camacho and P. Brazdil. Improving the robustness and encoding complexity of behavioural clones. In L. De Raedt and P. Flach, editors, *Proceedings of the 12th European Conference on Machine Learning (ECML '01)*, LNAI 2167, pages 37–48, Freiburg, Germany, September 2001. Springer.
50. R. Caruana. Multitask Learning. *Machine Learning, Second Special Issue on Inductive Transfer*, 28(1):41–75, 1991.
51. R. Caruana. Multitask Learning: A Knowledge-Based Source of Inductive Bias. In *Tenth International Conference on Machine Learning*, pages 41–48, 1993.
52. R. Caruana, A. Niculescu-Mizil, G. Crew, and A. Ksikes. Ensemble selection from libraries of models. In *Proceedings of the Twenty-first International Conference on Machine Learning (ICML'04)*, pages 137–144, 2004.
53. G. Castillo. *Adaptive Learning Algorithms for Bayesian Network Classifiers*. PhD thesis, University of Aveiro, Portugal, 2006.
54. G. Castillo and J. Gama. Bias management of bayesian network classifiers. In *Discovery Science, 8th International Conference, DS 2005*, LNAI 3735, pages 70–83. Springer-Verlag, 2005.
55. G. Castillo and J. Gama. An adaptive prequential learning framework for bayesian network classifiers. In *Knowledge Discovery in Databases: PKDD 2006, 10th European Conference on Principles and Practice of Knowledge Discovery in Databases*, LNAI 4213, pages 67–78. Springer-Verlag, 2006.
56. G. Castillo, J. Gama, and P. Medas. Adaptation to drifting concepts. In *Progress in Artificial Intelligence*, LNCS 2902, pages 279–293. Springer-Verlag, 2003.
57. G. Cauwenberghs and T. Poggio. Incremental and decremental support vector machine learning. In *Proceedings of the 13th Neural Information Processing Systems*, 2000.
58. P. Chan and S. Stolfo. Toward parallel and distributed learning by meta-learning. In *Working Notes of the AAAI-93 Workshop on Knowledge Discovery in Databases*, pages 227–240, 1993.
59. P. Chan and S. Stolfo. On the accuracy of meta-learning for scalable data mining. *Journal of Intelligent Information Systems*, 8:5–28, 1997.
60. O. Chapelle, V. Vapnik, O. Bousquet, and S. Mukherjee. Choosing multiple parameters for support vector machines. *Machine Learning*, 46(1):131–159, 2002. Available from <http://www.kernel-machines.org>.
61. P. Chapman, J. Clinton, R. Kerber, T. Khabaza, T. Reinartz, C. Shearer, and R. Wirth. CRISP-DM 1.0: Step-by-step data mining guide. Technical report, SPSS, Inc., 2000.
62. M. Charest and S. Delisle. Ontology-guided intelligent data mining assistance: Combining declarative and procedural knowledge. In *Proceedings of the Tenth IASTED International Conference on Artificial Intelligence and Soft Computing*, pages 9–14, 2006.
63. A. Charnes, W. Cooper, and E. Rhodes. Measuring the efficiency of decision making units. *European Journal of Operational Research*, 2(6):429–444, 1978.
64. C. Chatfield. *The Analysis of Time Series: An Introduction*. Chapman & Hall/CRC, 6th edition, 2003.
65. W. W. Cohen. Grammatically biased learning: Learning logic programs using an explicit antecedent description language. *Artificial Intelligence*, 68(2):303–366, 1994.

66. W. W. Cohen. Fast effective rule induction. In A. Prieditis and S. Russell, editors, *Proceedings of the Twelfth International Conference on Machine Learning*, pages 115–123. Morgan Kaufmann, 1995.
67. W. D. Cook, B. Golany, M. Penn, and T. Raviv. Creating a consensus ranking of proposals from reviewers' partial ordinal rankings. *Computers & Operations Research*, 34(4):954–965, April 2007.
68. W. D. Cook, M. Kress, and L. W. Seiford. A general framework for distance-based consensus in ordinal ranking models. *European Journal of Operational Research*, 96(2):392–397, 1996.
69. S. Craw, D. Sleeman, N. Granger, M. Rissakis, and S. Sharma. Consultant: Providing advice for the machine learning toolbox. In *Research and Development in Expert Systems IX (Proceedings of Expert Systems'92)*, pages 5–23. SGES Publications, 1992.
70. N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*. Cambridge University Press, 2000.
71. N. Cristianini, J. Shawe-Taylor, and C. Campbell. Dynamically adapting kernels in support vector machines. In M. Kearns, S. Solla, and D. Cohn, editors, *Advances in Neural Information Processing Systems*, volume 11, pages 204–210. MIT Press, 1998. Available from <http://www.kernel-machines.org>.
72. T. R. Davies and S. J. Russell. A logical approach to reasoning by analogy. In J. P. McDermott, editor, *Proceedings of the 10th International Joint Conference on Artificial Intelligence, IJCAI 1987*, pages 264–270, Freiburg, Germany, August 1987. Morgan Kaufmann.
73. A. P. Dawid. Statistical theory: The prequential approach. *Journal of the Royal Statistical Society A*, 147:278–292, 1984.
74. L. De Raedt and L. Dehaspe. Clausal discovery. *Machine Learning*, 26:99–146, 1997.
75. V. R. de Sa. Learning classification with unlabeled data. In *Advances in Neural Information Processing Systems*, pages 112–119, 1994.
76. T. Dietterich, D. Busquets, R. Lopez de Mantaras, and C. Sierra. Action Refinement in Reinforcement Learning by Probability Smoothing. In *19th International Conference on Machine Learning*, pages 107–114, 2002.
77. T. G. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting and randomization. *Machine Learning*, 40(2):139–157, 1998.
78. T. G. Dietterich and E. B. Kong. Machine learning bias, statistical bias, and statistical variance of decision tree algorithms. Technical report, Department of Computer Science, Oregon State University, 1995.
79. Data Mining Advisor. <http://www.metal-kdd.org>.
80. P. Domingos and M. Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning*, 29(2-3):103–130, 1997.
81. P. M. dos Santos, T. B. Ludermir, and R. B. C. Prudêncio. Selection of time series forecasting models based on performance information. In *Proceedings of the Fourth International Conference on Hybrid Intelligent Systems (HIS'04)*, pages 366–371, 2004.
82. R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley-Interscience, 2000.
83. S. Džeroski and N. Lavrač. *Relational Data Mining*. Springer, October 2001.

84. J. D. Easterlin and P. Langley. A framework for concept formation. In *Seventh Annual Conference of the Cognitive Science Society*, pages 267–271, Irvine CA, USA, 1985.
85. B. Efron. Estimating the error of a prediction rule: Improvement on cross-validation. *Journal of the American Statistical Association*, 78(382):316–330, 1983.
86. R. Engels. Planning tasks for knowledge discovery in databases; performing task-oriented user-guidance. In *Proceedings of the Second ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 170–175, 1996.
87. R. Engels, G. Lindner, and R. Studer. A guided tour through the data mining jungle. In *Proceedings of the Third ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 163–166, 1997.
88. R. Engels and C. Theusinger. Using a Data Metric for Offering Preprocessing Advice in Data-Mining Applications. In *Proceedings of the Thirteenth European Conference on Artificial Intelligence*, 1998.
89. T. Euler. Publishing operational models of data mining case studies. In *Proceedings of the ICDM Workshop on Data Mining Case Studies*, pages 99–106, 2005.
90. T. Euler, K. Morik, and M. Scholz. MiningMart: Sharing Successful KDD Processes. In *LLWA 2003 – Tagungsband der GI-Workshop-Woche Lehren – Lernen – Wissen – Adaptivitat*, pages 121–122, 2003.
91. T. Euler and M. Scholz. Using ontologies in a KDD workbench. In *Proceedings of the ECML/PKDD Workshop on Knowledge Discovery and Ontologies*, pages 103–108, 2004.
92. T. Evgeniou, C. Micchelli, and M. Pontil. Learning Multiple Tasks with Kernel Methods. *Journal of Machine Learning Research*, 6:615–637, 2005.
93. T. Evgeniou and M. Pontil. Regularized multi-task learning. In *Tenth Conference on Knowledge Discovery and Data Mining*, 2004.
94. S. E. Fahlman. The recurrent cascade-correlation architecture. *Advances in Neural Information Processing Systems*, 3:190–196, 1991.
95. C. Ferri, P. Flach, and J. Hernandez-Orallo. Delegating classifiers. In *Proceedings of the Twenty-first International Conference on Machine Learning (ICML'04)*, pages 289–296, 2004.
96. F. Fogelman-Soulié. Data mining in the real world: What do we need and what do we have? In R. Ghani and C. Soares, editors, *Proceedings of the Workshop on Data Mining for Business Applications*, pages 44–48, 2006.
97. G. Forman. Analysis of concept drift and temporal inductive transfer for Reuters 2000. In *Advances in Neural Information Processing Systems*, 2005.
98. E. Frank and I. H. Witten. Generating accurate rule sets without global optimization. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 144–151, 1998.
99. A. Freitas and S. Livingston. *Mining Very Large Databases with Parallel Processing*. Kluwer Academic Publ., 1998.
100. Y. Freund and R. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Proceedings of the European Conference on Computational Learning Theory*, pages 23–37, 1996.
101. Y. Freund and R. Schapire. Experiments with a new boosting algorithm. In *Proceedings of the Thirteenth International Conference on Machine Learning*, pages 148–156, 1996.

102. N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian network classifiers. *Machine Learning*, 29:131–161, 1997.
103. J. Fürnkranz. Separate-and-conquer rule learning. *Artificial Intelligence Review*, 13:3–54, 1999.
104. J. Fürnkranz and J. Petrak. An evaluation of landmarking variants. In C. Giraud-Carrier, N. Lavrač, and S. Moyle, editors, *Working Notes of the ECML/PKDD2000 Workshop on Integrating Aspects of Data Mining, Decision Support and Meta-Learning*, pages 57–68, 2001.
105. J. Gama. Iterative bayes. *Theoretical Computer Science*, 292(2):417–430, 2003.
106. J. Gama and P. Brazdil. Characterization of classification algorithms. In C. Pinto-Ferreira and N. J. Mamede, editors, *Progress in Artificial Intelligence, Proceedings of the Seventh Portuguese Conference on Artificial Intelligence*, pages 189–200. Springer-Verlag, 1995.
107. J. Gama and P. Brazdil. Linear tree. *Intelligent Data Analysis*, 3:1–22, 1999.
108. J. Gama and Brazdil P. Cascade generalization. *Machine Learning*, 41(3):315–343, 2000.
109. J. Gehrke. Report on the SIGKDD 2001 conference panel “New Research Directions in KDD”. *SIGKDD Explorations*, 3(2), 2002.
110. S. Geman, E. Bienenstock, and R. Doursat. Neural Networks and the Bias/Variance Dilemma. *Neural Computation*, pages 1–58, 1992.
111. R. Ghani and C. Soares. Data mining for business applications: KDD-2006 workshop. *SIGKDD Explorations*, 8(2):79–81, 2006.
112. D.F. Gordon and M. desJardins. Evaluation and selection of biases in machine learning. *Machine Learning*, 20(1/2):5–22, 1995.
113. E. Grant and R. Leavenworth. *Statistical Quality Control*. McGraw-Hill, 1996.
114. T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, 2001.
115. B. Hengst. Discovering Hierarchy in Reinforcement Learning with HEXQ. In *19th International Conference on Machine Learning*, pages 243–250, 2002.
116. J. M. Hernansaez, J. A. Botía, and A. F. Gómez-Skarmeta. A J2EE technology based distributed software architecture for Web usage mining. In *Proceedings of the Fifth International Conference on Internet Computing*, pages 97–101, 2004.
117. T. Heskes. Empirical Bayes for Learning to Learn. In *17th International Conference on Machine Learning*, pages 367–374. Morgan Kaufmann, San Francisco, CA, 2000.
118. S. Hettich and S.D. Bay. The UCI KDD archive, 1999. <http://kdd.ics.uci.edu>.
119. M. Hilario and A. Kalousis. Quantifying the resilience of inductive classification algorithms. In D. A. Zighed, J. Komorowski, and J. Zytkow, editors, *Proceedings of the Fourth European Conference on Principles of Data Mining and Knowledge Discovery*, pages 106–115. Springer-Verlag, 2000.
120. M. Hilario and A. Kalousis. Fusion of meta-knowledge and meta-data for case-based model selection. In A. Siebes and L. De Raedt, editors, *Proceedings of the Fifth European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD01)*. Springer, 2001.
121. T. B. Ho, T. D. Nguyen, and D. D. Nguyen. Visualization support for user-centered KDD process. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 519–524, 2002.

122. T. B. Ho, T. D. Nguyen, H. Shimodaira, and M. Kimura. A knowledge discovery system with support for model selection and visualization. *Applied Intelligence*, 19:125–141, 2003.
123. J. Huang and C. X. Ling. Using AUC and accuracy in evaluating learning algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 17(3):299–310, 2005.
124. G. Hulten and P. Domingos. Mining high-speed data streams. In *Proceedings of the ACM Sixth International Conference on Knowledge Discovery and Data Mining*, pages 71–80. ACM Press, 2000.
125. G. Hulten and P. Domingos. Catching up with the data: research issues in mining data streams. In *Proc. of Workshop on Research Issues in Data Mining and Knowledge Discovery*, 2001.
126. G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *Proceedings of the seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 97–106. ACM Press, 2001.
127. L. Hunter and A. Ram. Goals for learning and understanding. *Applied Intelligence*, 2(1):47–73, July 1992.
128. L. Hunter and A. Ram. The use of explicit goals for knowledge to guide inference and learning. In *Proceedings of the Eighth International Workshop on Machine Learning (ML’91)*, pages 265–269, San Mateo, CA, USA, July 1992. Morgan Kaufmann.
129. L. Hunter and A. Ram. Planning to learn. In A. Ram and D. B. Leake, editors, *Goal-Driven Learning*. MIT Press, 1995.
130. T. Jaakkola, M. Diekhans, and D. Haussler. Using the Fisher kernel method to detect remote protein homologies. In T. Lengauer, R. Schneider, P. Bork, D. Brutlag, J. Glasgow, M. H. Mewes, and R. Zimmer, editors, *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology*, pages 149–158. AAAI Press, 1999. Available from <http://www.kernel-machines.org>.
131. R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptive mixture of local experts. *Neural Computation*, 3(1):79–87, 1991.
132. M. I. Jordan and R. A. Jacobs. Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, 6:181–214, 1994.
133. A. M. Jorge and P. Brazdil. Architecture for iterative learning of recursive definitions. In L. De Raedt, editor, *Advances in Inductive Logic Programming*, volume 32 of *Frontiers in Artificial Intelligence and applications*. IOS Press, 1996.
134. A. Kalousis. *Algorithm Selection via Meta-Learning*. PhD thesis, University of Geneva, Department of Computer Science, 2002.
135. A. Kalousis, J. Gama, and M. Hilario. On data and algorithms: Understanding inductive performance. *Machine Learning*, 54(3):275–312, 2004.
136. A. Kalousis and M. Hilario. Model selection via meta-learning: A comparative study. In *Proceedings of the 12th International IEEE Conference on Tools with AI*. IEEE Press, 2000.
137. A. Kalousis and M. Hilario. Feature selection for meta-learning. In D. W. Cheung, G. Williams, and Q. Li, editors, *Proc. of the Fifth Pacific-Asia Conf. on Knowledge Discovery and Data Mining*. Springer, 2001.
138. A. Kalousis and M. Hilario. Representational issues in meta-learning. In *Proceedings of the 20th International Conference on Machine Learning (ICML03)*, pages 313–320, 2003.

139. A. Kalousis and T. Theoharis. NOEMON: Design, implementation and performance results of an intelligent assistant for classifier selection. *Intelligent Data Analysis*, 3(5):319–337, November 1999.
140. C. Kaynak and E. Alpaydin. Multistage cascading of multiple classifiers: One man’s noise is another man’s data. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 455–462, 2000.
141. J. Keller, I. Paterson, and H. Berrer. An integrated concept for multi-criteria ranking of data-mining algorithms. In J. Keller and C. Giraud-Carrier, editors, *Proceedings of the ECML Workshop on Meta-Learning: Building Automatic Advice Strategies for Model Selection and Method Combination*, pages 73–85, 2000.
142. E. Keogh and T. Folias. The UCR time series data mining archive. <http://www.cs.ucr.edu/~eamonn/TSDMA/index.html>, 2002. Riverside CA. University of California – Computer Science & Engineering Department.
143. J. Kittler, M. Hatef, R. P. W. Duin, and J. Matas. On combining classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20:226–239, 1998.
144. R. Klinkenberg. Learning drifting concepts: Example selection vs. example weighting. *Intelligent Data Analysis*, 2004.
145. R. Klinkenberg and T. Joachims. Detecting concept drift with support vector machines. In P. Langley, editor, *Proceedings of ICML-00, 17th International Conference on Machine Learning, Stanford, US, 2000*, pages 487–494. Morgan Kaufmann Publishers, 2000.
146. R. Klinkenberg and I. Renz. Adaptive information filtering: Learning in the presence of concept drifts. *Learning for Text Categorization*, pages 33–40, 1998.
147. Y. Kodratoff, D. Sleeman, M. Uszynski, K. Causse, and S. Craw. Building a machine learning toolbox. In L. Steels and B. Lepape, editors, *Enhancing the Knowledge Engineering Process*, pages 81–108. Elsevier Science Publishers, 1992.
148. R. Kohavi, L. Mason, R. Parekh, and Z. Zheng. Lessons and challenges from mining retail e-commerce data. *Machine Learning*, 57(1-2):83–113, 2004.
149. R. Kohavi and D. Wolpert. Bias plus variance decomposition for zero-one loss functions. In *Proceedings International Conference on Machine Learning*. Morgan Kaufmann, 1996.
150. C. Köpf and I. Iglezakis. Combination of task description strategies and case base properties for meta-learning. In M. Bohanec, B. Kavšek, N. Lavrač, and D. Mladenić, editors, *Proceedings of the Second International Workshop on Integration and Collaboration Aspects of Data Mining, Decision Support and Meta-Learning (IDDM-2002)*, pages 65–76. Helsinki University Printing House, 2002.
151. C. Köpf, C. Taylor, and J. Keller. Meta-analysis: From data characterization for meta-learning to meta-regression. In P. Brazdil and A. Jorge, editors, *Proceedings of the PKDD2000 Workshop on Data Mining, Decision Support, Meta-Learning and ILP: Forum for Practical Problem Presentation and Prospective Solutions*, pages 15–26, 2000.
152. C. Köpf, C. Taylor, and J. Keller. Multi-criteria meta-learning in regression – positions, developments and future directions. In C. Giraud-Carrier, N. Lavrač, S. Moyle, and B. Kavšek, editors, *ECML/PKDD Workshop on Integrating Aspects of Data Mining, Decision Support and Meta-Learning: Positions, Developments and Future Directions*, pages 67–76, 2001.

153. M. Koppel and S. P. Engelson. Integrating multiple classifiers by finding their areas of expertise. In *Proceedings of the AAAI-96 Workshop on Integrating Multiple Learned Models*, 1997.
154. S. Kramer and G. Widmer. Inducing classification and regression trees in first order logic. In S. Džeroski and N. Lavrač, editors, *Relational Data Mining*, pages 140–159. Springer, October 2001.
155. J. K. Kruscke. Dimensional Relevance Shifts in Category Learning. *Connection Science*, 8(2):225–248, 1996.
156. P. Kuba, P. Brazdil, C. Soares, and A. Woznica. Exploiting sampling and meta-learning for parameter setting support vector machines. In F. J. Garijo, J. C. Riquelme, and M. Toro, editors, *Proceedings of the Workshop de Minería de Datos y Aprendizaje associated with IBERAMIA 2002*, pages 217–225, 2002.
157. C. Lanquillon. *Enhancing Text Classification to Improve Information Filtering*. PhD thesis, University of Magdeburg, Germany, 2001.
158. R. Leite and P. Brazdil. Improving progressive sampling via meta-learning on learning curves. In J.-F. Boulicaut, F. Esposito, F. Giannotti, and D. Pedreschi, editors, *Proc. of the 15th European Conf. on Machine Learning (ECML2004)*, LNAI 3201, pages 250–261. Springer-Verlag, 2004.
159. R. Leite and P. Brazdil. Predicting relative performance of classifiers from samples. In *ICML '05: Proceedings of the 22nd International Conference on Machine Learning*, pages 497–503, NY, USA, 2005. ACM Press.
160. R. Leite and P. Brazdil. An iterative process for building learning curves and predicting relative performance of classifiers. In *Proceedings of the 13th Portuguese Conference on Artificial Intelligence (EPIA2007)*, pages 87–98, 2007.
161. G. Lindner and R. Studer. AST: Support for algorithm selection with a CBR approach. In C. Giraud-Carrier and B. Pfahringer, editors, *Recent Advances in Meta-Learning and Future Work*, pages 38–47. J. Stefan Institute, 1999. Available at <http://ftp.cs.bris.ac.uk/cgc/ICML99/lindner.ps.Z>.
162. Y. Liu and P. Stone. Value-function-based Transfer for Reinforcement Learning Using Structure Mapping. In *Proceedings of AAAI, Conference on Artificial Intelligence*, 2006.
163. R. Maclin, J. W. Shavlik, L. Torrey, T. Walker, and E. W. Wild. Giving Advice about Preferred Actions to Reinforcement Learners Via Knowledge-Based Kernel Regression. In *Proceedings of AAAI, Conference on Artificial Intelligence*, pages 819–824, 2005.
164. M. Maloof and R. Michalski. Selecting examples for partial memory learning. *Machine Learning*, 41:27–52, 2000.
165. A. Maurer. Algorithmic Stability and Meta-Learning. *Journal of Machine Learning Research*, 6:967–994, 2005.
166. METAL: A meta-learning assistant for providing user support in machine learning and data mining. ESPRIT Framework IV LTR Reactive Project Nr. 26.357, 1998–2001. <http://www.metal-kdd.org>.
167. C. A. Micchelli and M. Pontil. Kernels for Multi-Task Learning. In *Advances in Neural Information Processing Systems, Workshop on Inductive Transfer*, 2004.
168. R. Michalski. Inferential theory of learning: Developing foundations for multi-strategy learning. In R. Michalski and G. Tecuci, editors, *Machine Learning: A Multistrategy Approach, Volume IV*, chapter 1, pages 3–62. Morgan Kaufmann, February 1994.

169. D. Michie, D. J. Spiegelhalter, and C. C. Taylor. *Machine Learning, Neural and Statistical Classification*. Ellis Horwood, 1994.
170. MiningMart: Enabling end-user datawarehouse mining. IST Project Nr. 11993, 2000–2003.
171. MiningMart Internet case base. <http://mmart.cs.uni-dortmund.de/end-user/caseBase.html>.
172. M. Minsky. A framework for representing knowledge. In P. H. Winston, editor, *The Psychology of Computer Vision*, pages 211–277. McGraw-Hill, 1975.
173. T. Mitchell. Generalization as Search. *Artificial Intelligence*, 18(2):203–226, 1982.
174. T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.
175. Machine learning toolbox. ESPRIT Framework II Research Project Nr. 2154, 1990–1993.
176. K. Morik and M. Scholz. The MiningMart approach to knowledge discovery in databases. In N. Zhong and J. Liu, editors, *Intelligent Technologies for Information Analysis*, chapter 3, pages 47–65. Springer, 2004. Available from <http://www-ai.cs.uni-dortmund.de/MMWEB>.
177. K. Morik, S. Wrobel, J. Kietz, and W. Emde. *Knowledge Acquisition and Machine Learning: Theory, Methods and Applications*. Academic Press, 1993.
178. K.-R. Müller, S. Mika, G. Rätsch, K. Tsuda, and B. Schölkopf. An introduction to kernel-based learning algorithms. *IEEE Transactions on Neural Networks*, 12(2):181–201, 2001. Available from <http://www.kernel-machines.org>.
179. G. Nakhaeizadeh and A. Schnabl. Development of multi-criteria metrics for evaluation of data mining algorithms. In *Proceedings of the Fourth International Conference on Knowledge Discovery in Databases & Data Mining*, pages 37–42. AAAI Press, 1997.
180. G. Nakhaeizadeh and A. Schnabl. Towards the personalization of algorithms evaluation in data mining. In R. Agrawal and P. Stolorz, editors, *Proceedings of the Third International Conference on Knowledge Discovery & Data Mining*, pages 289–293. AAAI Press, 1998.
181. H. R. Neave and P. L. Worthington. *Distribution-Free Tests*. Routledge, 1992.
182. A. Niculescu-Mizil and R. Caruana. Learning the Structure of Related Tasks. In *Workshop at NIPS (Neural Information Processing Systems)*, 2005.
183. I. Noda, H. Matsubara, K. Hiraki, and I. Frank. Soccer Server: A Tool for Research on Multiagents Systems. *Journal of Applied Artificial Intelligence*, 12:233–250, 1998.
184. D. Opitz and R. Maclin. Popular ensemble methods: An empirical study. *Journal of Artificial Intelligence Research*, 11:169–198, 1999.
185. J. Ortega. *Making the Most of What You’ve Got: Using Models and Data to Improve Prediction Accuracy*. PhD thesis, Vanderbilt University, 1996.
186. J. Ortega, M. Koppel, and S. Argamon. Arbitrating among competing classifiers using learned referees. *Knowledge and Information Systems Journal*, 3(4):470–490, 2001.
187. M. Pavan and R. Todeschini. New indices for analysing partial ranking diagrams. *Analytica Chimica Acta*, 515(1):167–181, 2004.
188. Y. Peng, P. Flach, P. Brazdil, and C. Soares. Decision Tree-Based Characterization for Meta-Learning. In *Proceedings of the ECML/PKDD’02 Workshop on Integration and Collaboration Aspects of Data Mining, Decision Support and Meta-Learning*, pages 111–122, 2002.

189. Y. Peng, P. Flach, P. Brazdil, and C. Soares. Improved dataset characterisation for meta-learning. In *Discovery Science*, pages 141–152, 2002.
190. B. Pfahringer, H. Bensusan, and C. Giraud-Carrier. Meta-learning by Landmarking Various Learning Algorithms. In P. Langley, editor, *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 743–750, 2000.
191. J. Phillips and B. G. Buchanan. Ontology-guided knowledge discovery in databases. In *Proceedings of the First International Conference on Knowledge Capture*, pages 123–130, 2001.
192. L. Pratt and B. Jennings. A Survey of Connectionist Network Reuse Through Transfer. In S. Thrun and L. Pratt, editors, *Learning to Learn*, chapter 2, pages 19–44. Kluwer Academic Publishers, MA., 1998.
193. L. Pratt and S. Thrun. Second Special Issue on Inductive Transfer. *Machine Learning*, 28:41–75, 1997.
194. B. Price and C. Boutilier. Accelerating Reinforcement Learning Through Implicit Imitation. *Journal of Artificial Intelligence Research*, 19:569–629, 2003.
195. Project Statlog. Comparative testing and evaluation of statistical and logical learning algorithms for large-scale applications in classification, prediction and control. ESPRIT Framework II Research Project Nr. 5170, 1991-1994.
196. R. Prudêncio and T. Ludermir. Meta-learning approaches to selecting time series models. *Neurocomputing*, 61:121–137, 2004.
197. J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Francisco, CA, 1993.
198. R. Quinlan. *C5.0: An Informal Tutorial*. RuleQuest, 1998. <http://www.rulequest.com/see5-unix.html>.
199. R. Quinlan and R. Cameron-Jones. FOIL: A midterm report. In P. Brazdil, editor, *Proc. of the Sixth European Conf. on Machine Learning*, volume 667 of *LNAI*, pages 3–20. Springer-Verlag, 1993.
200. E. J. Rafols, M. B. Ring, R. S. Sutton, and B. Tanner. Using Predictive Representations to Improve Generalization in Reinforcement Learning. In L. P. Kaelbling and A. Saffiotti, editors, *19th International Joint Conference on Artificial Intelligence*, pages 835–840, 2005.
201. R. Raina, A. Y. Ng, and D. Koller. Transfer Learning by Constructing Informative Priors. In *Workshop at NIPS (Neural Information Processing Systems)*, 2005.
202. A. Ram and D. B. Leake, editors. *Goal Driven Learning*. MIT Press, 2005.
203. L. Rendell. Learning Hard Concepts. In *Proceedings of the Third European Working Session on Learning*, pages 177–200, 1988.
204. L. Rendell and H. Cho. Empirical Learning as a Function of Concept Character. *Machine Learning*, 5(3):267–298, 1990.
205. L. Rendell and H. Ragavan. Improving the Design of Induction Methods by Analyzing Algorithm Functionality and Data-Based Concept Complexity. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 952–958, 1993.
206. L. Rendell, R. Seshu, and D. Tcheng. Layered Concept-Learning and Dynamically-Variable Bias Management. In *Proceedings of the International Joint Conference of Artificial Intelligence*, pages 308–314, 1987.
207. L. Rendell, R. Seshu, and D. Tcheng. More robust concept learning using dynamically-variable bias. In P. Langley, editor, *Proc. of the Fourth Int. Workshop on Machine Learning*, pages 66–78. Morgan Kaufmann, 1987.

208. M. T. Rosenstein, Z. Marx, and L. P. Kaelbling. To Transfer or Not To Transfer. In *Workshop at NIPS (Neural Information Processing Systems)*, 2005.
209. S. Rosset, C. Perlich, and B. Zadrozny. Ranking-based evaluation of regression models. In *Proceedings of the 5th IEEE International Conference on Data Mining (ICDM 2005)*, pages 370–377, 2005.
210. M. Saar-Tsechansky and F. Provost. Handling missing values when applying classification models. *Journal of Machine Learning Research*, 8:1623–1657, 2007.
211. M. Sahami. Learning limited dependence bayesian classifiers. In *Proceedings of KDD-96*, 10, pages 335–338. AAAI Press, 1996.
212. L. Saitta and F. Neri. Learning in the “real world”. *Machine Learning*, 30(2/3):133–163, 1998.
213. C. Sammut, S. Hurst, D. Kedzier, and D. Michie. Learning to fly. In D. H. Sleeman and P. Edwards, editors, *Proceedings of the Ninth International Workshop on Machine Learning (ML’92)*, pages 385–393, Aberdeen, Scotland, UK, July 1992. Morgan Kaufmann.
214. C. Schaffer. Selecting a classification method by cross-validation. *Machine Learning*, 13(1):135–143, 1993.
215. R. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990.
216. J. Schmidhuber. Shifting Inductive Bias with Success-Story Algorithm, Adaptive Levin Search, and Incremental Self-Improvement. *Machine Learning*, 28:105–130, 1997.
217. J. Schmidhuber. Bias-Optimal Incremental Problem Solving. In K. Obermayer S. Becker, S. Thrun, editor, *Advances in Neural Information Processing Systems*, pages 1571–1578, 2003.
218. J. Schmidhuber. Optimal Ordered Problem Solver. *Machine Learning*, 54:211–254, 2004.
219. T. R. Schultz and F. Rivest. Knowledge-Based Cascade Correlation: An Algorithm for Using Knowledge to Speed Learning. In P. Langley, editor, *16th International Conference on Machine Learning*, pages 871–878, 2000.
220. P. D. Scott and E. Wilkins. Evaluating data mining procedures: techniques for generating artificial data sets. *Information & Software Technology*, 41(9):579–587, 1999.
221. O. Selfridge, R. S. Sutton, and A. G. Barto. Training and Tracking in Robotics. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 670–672, 1985.
222. S. Shalev-Shwartz and Y. Singer. Efficient learning of label ranking by soft projections onto polyhedra. *Journal of Machine Learning Research*, 7:1567–1599, 2006.
223. N. E. Sharkey and A. J. C. Sharkey. Adaptive Generalization. *Artificial Intelligence Review*, 7:313–328, 1993.
224. S. Sharma, D. Sleeman, N. Granger, and M. Rissakis. Specification of consultant-3. Deliverable 5.7 of ESPRIT Project MLT (Nr 2154), Ref: MLT/WP5/Abdn/D5.7, 1993.
225. J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.
226. D. L. Silver and R. E. Mercer. The Parallel Transfer of Task Knowledge Using Dynamic Learning Rates Based on a Measure of Relatedness. *Connection Science*, 8(2):277–294, 1996.

227. D. L. Silver and R. E. Mercer. The Task Rehearsal Method of Life-Long Learning: Overcoming Impoverished Data. In *17th International Conference of the Canadian Society for Computational Studies of Intelligence*, pages 217–232, 2002.
228. G. Silverstein and M. J. Pazzani. Relational clichés: Constraining induction during relational learning. In L. Birnbaum and G. Collins, editors, *Proceedings of the Eighth International Workshop on Machine Learning (ML'91)*, pages 203–207, San Francisco, CA, USA, 1991. Morgan Kaufmann.
229. S. Singh. Transfer of Learning by Composing Solutions of Elemental Sequential Tasks. *Machine Learning Journal*, 8(3):323–339, 1992.
230. D. Sleeman, M. Rissakis, S. Craw, N. Graner, and S. Sharma. Consultant-2: pre- and post-processing of machine learning applications. *Int. J. Human-Computer Studies*, 43:43–63, 1995.
231. A. J. Smola and B. Schölkopf. From regularization operators to support vector kernels. In *Advances in Neural Information Processing Systems*, 1998. Available from <http://www.kernel-machines.org>.
232. C. Soares. Is the UCI repository useful for data mining? In F. Moura-Pires and S. Abreu, editors, *Proceedings of the 11th Portuguese Conference on Artificial Intelligence (EPIA2003)*, volume 2902 of *LNAI*, pages 209–223. Springer-Verlag, 2003.
233. C. Soares. *Learning Rankings of Learning Algorithms*. PhD thesis, Department of Computer Science, Faculty of Sciences, University of Porto, 2004. Supervisors: P. Brazdil and J. P. da Costa.
234. C. Soares and P. Brazdil. Zoomed ranking: Selection of classification algorithms based on relevant performance information. In D. A. Zighed, J. Komorowski, and J. Zytkow, editors, *Proceedings of the Fourth European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD2000)*, pages 126–135. Springer, 2000.
235. C. Soares and P. Brazdil. Selecting parameters of SVM using meta-learning and kernel matrix-based meta-features. In *Proceedings of the ACM SAC*, 2006.
236. C. Soares, P. Brazdil, and P. Kuba. A meta-learning method to select the kernel width in support vector regression. *Machine Learning*, 54:195–209, 2004.
237. C. Soares, J. Petrak, and P. Brazdil. Sampling-based relative landmarks: Systematically test-driving algorithms before choosing. In P. Brazdil and A. Jorge, editors, *Proceedings of the 10th Portuguese Conference on Artificial Intelligence (EPIA2001)*, pages 88–94. Springer, 2001.
238. S. Y. Sohn. Meta analysis of classification algorithms for pattern recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(11):1137–1144, Nov. 1999.
239. C. Spearman. The proof and measurement of association between two things. *American Journal of Psychology*, 15:72–101, 1904.
240. R. S. Stepp and R. S. Michalski. How to structure structured objects. In *Proceedings of the International Workshop on Machine Learning*, Urbana, IL, USA, 1983.
241. P. Stone. *Layered Learning in Multiagent Systems: A Winning Approach to Robotic Soccer*. MIT Press, March 2000.
242. P. Stone and R. Sutton. Scaling Reinforcement Learning Toward Robocup Soccer. In *International Conference on Machine Learning*, pages 537–544, 2001.

243. P. Stone and M. Veloso. Layered Learning. In *Proceedings of the 11th European Conference on Machine Learning*, pages 369–381, 2000.
244. C. Sutton and A. McCallum. Composition of Conditional Random Fields for Transfer Learning. In *Human Language Technology, Empirical Methods in Natural Language Processing*, pages 748–754, 2005.
245. A. Suyama, N. Negishi, and T. Yamaguchi. CAMLET: A platform for automatic composition of inductive learning systems using ontologies. In *Pacific Rim International Conference on Artificial Intelligence*, pages 205–215, 1998.
246. A. Suyama, N. Negishi, and T. Yamaguchi. Composing inductive applications using ontologies for machine learning. In *Discovery Science*, pages 429–430, 1998.
247. A. Suyama and T. Yamaguchi. Specifying and learning inductive learning systems using ontologies. In *Working Notes from the 1998 AAAI Workshop on the Methodology of Applying Machine Learning: Problem Definition, Task Decomposition and Technique Selection*, pages 29–36, 1998.
248. S. Swarup, M. Mahmud, K. Lakkaraju, and S. Ray. *Cumulative Learning: Towards Designing Cognitive Architectures for Artificial Agents that Have a Lifetime*. Technical Report UIUCDCS-R-2005-2514, Department of Computer Science, University of Illinois at Urbana-Champaign, 2005.
249. S. Swarup and S. Ray. Cross-domain Knowledge Transfer Using Structured Representations. In *Workshop at NIPS (Neural Information Processing Systems)*, pages 111–222, 2005.
250. M. E. Taylor and P. Stone. Behavior Transfer for Value-Function-Based Reinforcement Learning. In *Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 53–59, 2005.
251. M. E. Taylor and P. Stone. Transfer via Inter-Task Mappings in Policy Search Reinforcement Learning. In *Conference on Autonomous Agents and Multi-Agent Systems*, 2007.
252. Y. Teh, M. Seeger, and M. Jordan. Semiparametric Latent Factor Models. In *Tenth International Workshop on Artificial Intelligence and Statistics*, pages 333–340, 2005.
253. S. Thrun. A Lifelong Learning Perspective for Mobile Robot Control. In *Proceedings of the IEEE/RSJ/GI Conference on Intelligent Robots and Systems*, pages 23–30, 1994.
254. S. Thrun. Lifelong Learning Algorithms. In S. Thrun and L. Pratt, editors, *Learning to Learn*, chapter 8, pages 181–209. Kluwer Academic Publishers, MA, 1998.
255. S. Thrun and T. Mitchell. Learning One More Thing. In *Proceedings of the International Joint Conference of Artificial Intelligence*, pages 1217–1223, 1995.
256. S. Thrun and T. Mitchell. Lifelong Robot Learning. *Robotics and Autonomous Systems*, 15:25–46, 1995.
257. S. Thrun and J. O’Sullivan. Clustering Learning Tasks and the Selective Cross-Task Transfer of Knowledge. In S. Thrun and L. Pratt, editors, *Learning to Learn*, pages 235–257. Kluwer Academic Publishers, MA., 1998.
258. S. Thrun and L. Pratt. Learning to Learn: Introduction and Overview. In S. Thrun and L. Pratt, editors, *Learning to Learn*, chapter 1, pages 3–17. Kluwer Academic Publishers, MA., 1998.
259. K. Ting and I. Witten. Stacked generalization: When does it work? In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pages 866–871, 1997.

260. K. M. Ting and B. T. Low. Model combination in the multiple-data-batches scenario. In *Proceedings of the Ninth European Conference on Machine Learning (ECML-97)*, pages 250–265, 1997.
261. L. Todorovski, H. Blockeel, and S. Džeroski. Ranking with predictive clustering trees. In T. Elomaa, H. Mannila, and H. Toivonen, editors, *Proc. of the 13th European Conf. on Machine Learning*, number 2430 in LNAI, pages 444–455. Springer-Verlag, 2002.
262. L. Todorovski, P. Brazdil, and C. Soares. Report on the experiments with feature selection in meta-level learning. In P. Brazdil and A. Jorge, editors, *Proceedings of the Data Mining, Decision Support, Meta-Learning and ILP Workshop at PKDD2000*, pages 27–39, 2000.
263. L. Todorovski and S. Džeroski. Experiments in meta-level learning with ILP. In J. Rauch and J. Zytkow, editors, *Proceedings of the Third European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD99)*, pages 98–106. Springer, 1999.
264. L. Todorovski and S. Džeroski. Combining classifiers with meta decision trees. *Machine Learning*, 50(3):223–250, 2003.
265. L. Torgo. *Inductive Learning of Tree-Based Regression Models*. PhD thesis, Dep. Ciências de Computadores, Fac. Ciências, Univ. Porto, 1999.
266. L. Torrey, J. W. Shavlik, T. Walker, and R. Maclin. Skill Acquisition via Transfer Learning and Advice Taking. In *Proceedings of the European Conference on Machine Learning (ECML)*, 2006.
267. L. Torrey, T. Walker, J. W. Shavlik, and R. Maclin. Using Advice to Transfer Knowledge Acquired in One Reinforcement Learning Task to Another. In *Proceedings of the European Conference on Machine Learning (ECML)*, pages 412–424, 2005.
268. K. Tsuda, G. Rätsch, S. Mika, and K. Müller. Learning to predict the leave-one-out error of kernel based classifiers. In *ICANN*, pages 331–338. Springer-Verlag, 2001.
269. A. Tsymbal, S. Puuronen, and V. Terziyan. A technique for advanced dynamic integration of multiple classifiers. In *Proceedings of the Finnish Conference on Artificial Intelligence (STeP'98)*, pages 71–79, 1998.
270. P. Utgoff and D. J. Straczuzi. Many Layered Learning. *Neural Computation*, 14:2497–2529, 2002.
271. J. Vanschoren and H. Blockeel. Towards understanding learning behavior. In *Proceedings of the Fifteenth Annual Machine Learning Conference of Belgium and the Netherlands*, 2006.
272. V. Vapnik. *The Nature of Statistical Learning Theory*. Springer Verlag, New York, 1995.
273. F. Verdenius and R. Engels. A process model for developing inductive applications. In *Proceedings of the Seventh Belgian-Dutch Conference on Machine Learning (Benelearn)*, 1997.
274. F. Verdenius and M. Van Someren. Applications of inductive learning techniques: A survey in the Netherlands. *AI Communications*, 10:3–20, 1997.
275. R. Vilalta. Understanding accuracy performance through concept characterization and algorithm analysis. In C. Giraud-Carrier and B. Pfahringer, editors, *Recent Advances in Meta-Learning and Future Work*, pages 3–9. J. Stefan Institute, 1999.
276. R. Vilalta and Y. Drissi. A perspective view and survey of meta-learning. *Artificial Intelligence Review*, 18(2):77–95, 2002.

277. R. Vilalta, C. Giraud-Carrier, P. Brazdil, and C. Soares. Using meta-learning to support data-mining. *International Journal of Computer Science Applications*, I(1):31–45, 2004.
278. S. R. Waterhouse and A. J. Robinson. Classification using hierarchical mixtures of experts. In *IEEE Workshop on Neural Networks for Signal Processing IV*, pages 177–186, 1994.
279. G. I. Webb. Decision tree grafting. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pages 846–851, 1997.
280. S. M. Weiss and N. Indurkha. *Predictive Data Mining: A Practical Guide*. Morgan Kaufmann, August 1997.
281. S. M. Weiss, N. Indurkha, T. Zhang, and F. J. Damerau. *Text Mining: Predictive Methods for Analyzing Unstructured Information*. Springer, 2005.
282. G. Widmer and M. Kubat. Learning in the presence of concept drift and hidden context. *Machine Learning*, 23:69–101, 1996.
283. R. Wirth, C. Shearer, U. Grimmer, T. P. Reinartz, J. Schlosser, C. Breitner, R. Engels, and G. Lindner. Towards process-oriented tool support for knowledge discovery in databases. In *Proceedings of the First European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 243–253, 1997.
284. D. H. Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–259, 1992.
285. S. Wrobel. *Concept Formation and Knowledge Revision*. Kluwer Academic Publishers, 1994.
286. L. S. Wygotski. *Thought and Language*. MIT Press, 1962.
287. J. Zhang, Z. Ghahramani, and Y. Yang. Learning Multiple Related Tasks using Latent Independent Component Analysis. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems 18*. MIT Press, Cambridge, MA, 2005.
288. N. Zhong, C. Liu, and S. Oshuga. A way of increasing both autonomy and versatility of a KDD system. In *Proceedings of the Tenth International Symposium on Foundations of Intelligent Systems, LNCS 1325*, pages 94–105. Springer, 1997.
289. N. Zhong, C. Liu, and S. Oshuga. Dynamically organizing KDD processes. *International Journal of Pattern Recognition and Artificial Intelligence*, 15(3):451–473, 2001.
290. N. Zhong and S. Oshuga. GLS – a methodology for discovering knowledge from databases. In *Proceedings of the Thirteenth International CODATA Conference*, pages A20–A30, 1992.

A

Terminology

base-learning (or **base-level learning**): The process of invoking a machine learning (ML) algorithm or a data mining (DM) process on a ML/DM application.

base-algorithm (or **base-level algorithm**): Algorithm used for base-learning.

metalearning (or **meta-level learning**): The process of invoking a learning algorithm to obtain knowledge concerning the behavior of machine learning (ML) and data mining (DM) processes.

meta-searching: One type of metalearning, in which a problem is divided into a set of sequential sub-problems.

meta-algorithm (or **meta-level algorithm**): Algorithm used for metalearning.

metalearner: Same as meta-algorithm.

metadata: Data that characterize datasets, algorithms and/or ML/DM processes.

metadataset: Database of metadata.

metadatabase: Same as metadataset.

metadecision: Output of a metalearning model.

meta-example: Record of a metadataset.

meta-instance: Same as meta-example.

metafact: One type of representation of metadata.

metadistribution: Distribution of meta-examples.

metafeature: Variable that characterizes a dataset, algorithm or a ML/DM process.

meta-attribute: Same as metafeature.

metatarget (or **target metafeature**): Variable that represents metadecisions.

metaknowledge: Knowledge concerning learning processes.

metainformation: Information concerning learning processes.

metamodel (or **meta-learning model**): Output of a meta-algorithm, encoding the metamodel.

metarule: One type of metamodel.

metapredicate: One type of metamodel.

B

Mathematical Symbols

Symbol	Description
$\mathbf{T} = \{(\mathbf{x}_1, \mathbf{p}_1), \dots, (\mathbf{x}_m, \mathbf{p}_m)\}$	Metadataset/metadatabase
m	Number of meta-examples
$\mathbf{x}_i = (x_{i,1}, x_{i,2}, \dots, x_{i,k})$	Metafeature vector of meta-example i
k	Number of metafeatures
$\mathbf{p}_i = \{p_1, \dots, p_n\}$	Estimates of the performance of base-algorithms associated with dataset i
$\mathbf{A} = \{a_1, \dots, a_n\}$	Set of base-algorithms i
n	Number of base-algorithms
$\mathbf{x} = (x_1, x_2, \dots, x_k)$	Feature vector
k	Number of features
y	Class label
$\mathbf{e} = (\mathbf{x}, y)$	Example (feature vector and class label)
$T = \{\mathbf{e}_i\} = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$	Training set, sample
$\mathbf{T} = \{T_1, T_2, \dots, T_n\}$	Set of training samples
$\mathcal{X}$	Input space
$\mathcal{Y}$	Output space
$h : \mathcal{X} \rightarrow \mathcal{Y}$	Hypothesis (receives example, outputs class label)
$\mathcal{H} = \{h\}$	Hypothesis space
$\mathbb{H} = \{\mathcal{H}\}$	Family of hypothesis spaces
L	Loss function
ϕ	Learning task (probability function over $\mathcal{X} \times \mathcal{Y}$)
Φ	Distribution over the space of all distributions ϕ_i
$\text{VC}(\mathcal{H})$	Vapnik–Chervonenkis dimension of $\mathcal{H}$
$A : \bigcup_{m>0} (\mathcal{X} \times \mathcal{Y})^m \rightarrow \mathcal{H}$	Learning algorithm (receives a training sample, outputs a hypothesis)
$\mathcal{A} : \bigcup (\mathcal{X} \times \mathcal{Y})^{(n,m)} \rightarrow \mathbb{H}$	Metalearning algorithm (receives training samples, outputs a hypothesis space)

Index

- Active learning, 8
- Algorithm recommendation, 13
- Algorithm-specific metafeatures, 46
- Algorithmic stability, 121
- Arbitrating, 86
- Auxiliary subproblems, 115

- Bagging, 74
- Base-learning, 2
- Bayesian network classifiers, 98
- Best algorithm in a set, 33
- Bias, 91
- Bias management, 100
- Bias selection, 95
- Bias-variance dilemma, 123
- Bias-variance error decomposition, 91
- Boosting, 76

- Cascade generalization, 79
- Cascading, 82
- Change detection, 93
- Characterization of algorithms, 43
- CITRUS, 66
- Clause structure grammar, 137
- Common model, 113
- Component reuse, 117
- Composite learning systems, 8
- Composition of complex systems, 9
- Concept drift, 93, 103
- Consultant, 63
- Control of language bias, 137
- Controlling learning, 8

- Data landscape, 125
- Data Mining Advisor, 63
- Data streams, 92
- Declarative bias, 4
- Default ranking, 21
- Definition of meta learning, 10
- Delegating, 84
- Domain-dependent language bias, 137
- Domain-specific metaknowledge, 139
- Dynamic selection of bias, 129

- Empirical metaknowledge, 53
- Equivalence classes, 123
- Estimation of performance, 36, 58
- Exploiting existing ontologies, 150

- Failures of algorithms, 58
- Functional transfer, 110

- Generation of datasets, 50
- Global Learning Scheme, 71
- Goal/concept graphs, 131

- Hyper-prior distribution, 115

- Inductive transfer, 109
- Intelligent Discovery Assistant, 67
- Iterative induction, 140

- KDD/DM process, 6

- Landmarkers, 6, 45
- Layered learning, 146
- Learning a skill, 145

- Learning bias, 3
- Learning complex behavior, 146
- Learning coordinated actions, 148
- Learning from data streams, 8
- Learning goals, 130, 133
- Learning individual skills, 142
- Learning rankings, 14
- Learning recursive concepts, 139
- Learning to control a device, 143
- Learning to learn, 2, 109
- Literal transfer, 110

- Manipulation of datasets, 51
- Meta-algorithm, 31
- Meta-attributes, 71
- Meta-decision trees, 88
- Meta-examples, 31, 48
- Meta-information, 86
- Meta-instance, 78
- Meta-searching, 117
- Metafeatures, 14, 124
- Metaknowledge, 3
- Metalearner, 9, 118
- Metalearning, 1, 13
- Metalearning assistants, 124
- Metadata, 13, 42
- Metadatabase, 31
- Metadistribution, 119
- Metafacts, 137
- Metafeatures, 31, 42
- Metaknowledge, 31
- METALA, 70
- Metamodel, 71
- Metapredicates, 137
- Metarules, 72
- Metatarget, 31, 33
- MiningMart, 65
- Model combination, 7
- Model-based metafeatures, 6, 44
- Multiple predicate learning, 134
- Multitarget prediction, 40
- Multitask learning, 111

- Non-literal transfer, 111

- Ontology, 139

- Parameter settings, 26
- Partial order of operations, 7
- Plan, 7
- Planning to learn, 151
- Predictions as features, 116
- Procedural bias, 4

- Ranking accuracy, 18
- Ranking aggregation, 16
- Ranking algorithms, 41
- Ranking trees, 42
- Rankings, 35
- Repositories of datasets, 50
- Representational transfer, 110

- Sequential analysis, 94
- Sequential covering method, 135
- Shift of bias, 151
- Similar prior distribution, 116
- Simple, statistical and information-theoretic metafeatures, 6, 43
- Source network, 110
- Stacking, 78
- Statistical process control, 103
- Subset of algorithms, 34

- Target network, 110
- Task relatedness, 122
- Task-dependent metafeatures, 45
- Tasks as clusters, 117
- Theoretical metaknowledge, 53
- Transfer in reinforcement learning, 126
- Transfer in robotics, 125
- Transfer of knowledge, 9

- Update of metadata, 57

- Very Fast Decision Tree, 96

Cognitive Technologies

H. Prendinger, M. Ishizuka (Eds.)

Life-Like Characters

Tools, Affective Functions, and Applications

IX, 477 pages. 2004

H. Helbig

Knowledge Representation and the Semantics of Natural Language

XVIII, 646 pages. 2006

P.M. Nugues

An Introduction to Language Processing with Perl and Prolog

An Outline of Theories, Implementation, and Application with Special Consideration of English, French, and German

XX, 513 pages. 2006

W. Wahlster (Ed.)

SmartKom: Foundations of Multimodal Dialogue Systems

XVIII, 644 pages. 2006

B. Goertzel, C. Pennachin (Eds.)

Artificial General Intelligence

XVI, 509 pages. 2007

O. Stock, M. Zancanaro (Eds.)

PEACH — Intelligent Interfaces for Museum Visits

XVIII, 316 pages. 2007

V. Torra, Y. Narukawa

Modeling Decisions: Information Fusion and Aggregation Operators

XIV, 284 pages. 2007

P. Manoonpong

Neural Preprocessing and Control of Reactive Walking Machines

Towards Versatile Artificial Perception–Action Systems

XVI, 185 pages. 2007

S. Patnaik

Robot Cognition and Navigation

An Experiment with Mobile Robots

XVI, 290 pages. 2007

M. Cord, P. Cunningham (Eds.)

Machine Learning Techniques for Multimedia

Case Studies on Organization and Retrieval

XVI, 290 pages. 2008

L. De Raedt

Logical and Relational Learning

XVI, 388 pages. 2008