Scalable and Accurate Knowledge Discovery in Real-World Databases

Dissertation

zur Erlangung des Grades eines

DOKTORS DER NATURWISSENSCHAFTEN

der Universität Dortmund
am Fachbereich Informatik
von

Martin Scholz

Dortmund

2007

Tag der mündlichen Prüfung:	25.4.2007
Dekan:	Prof. Dr. Peter Buchholz
Gutachter/Gutachterinnen:	Prof. Dr. Katharina Morik Prof. Dr. Gabriele Kern-Isberner Prof. Dr. Stefan Wrobel

Danksagung

Ich möchte mich an dieser Stelle bei den vielen Menschen bedanken, die zu dieser Arbeit, jede(r) auf eine eigene Weise, einen Beitrag geleistet haben.
Dies gilt in erster Linie für das gesamte LS8-Team der letzten Jahre. Ich danke allen herzlich für ein kollegiales Arbeitsumfeld, viele anregende Diskussionen, und nicht zuletzt, für eine schöne Zeit am Lehrstuhl. Katharina Morik danke ich für ein Umfeld, in dem es mir leicht fiel, mich mit spannenden wissenschaftlichen Fragestellungen auseinanderzusetzen, sowie für visionäre Ideen und stets konstruktive Kritik, die diese Arbeit fortlaufend begleitet haben. Timm sei für seinen kontinuierlichen Englischunterricht in Form unermüdlichen und instruktiven Korrekturlesens gedankt, ohne den diese Arbeit vermutlich nicht lesbar wäre. Herzlichen Dank auch an Ingo, für den stets enthusiastischen Support bei jedweder Frage rund um die Wunderwelt von YALE. Ich danke allen WiMis für die wissenschaftlichen und unwissenschaftlichen Diskussionen "nebenbei", die mir stets eine willkommene Quelle der Inspiration bzw. Regeneration waren. Mein Dank gilt auch allen NiWiMis und HiWis für den organisatorischen und technischen Support. Schließlich danke ich noch allen Aktivisten der LS8 Fußballbewegung für einen recht indirekten, schwer messbaren Beitrag zu dieser Arbeit.
Gabriele Kern-Isberner und Stefan Wrobel danke ich für die schnelle Bereitschaft meine Dissertation zu begutachten.
Mein Dank gilt auch den Mitarbeitern des SFB 475 für inspirierende Gespräche und eine ergänzende Sichtweise auf die Wissensentdeckung in Datenbanken.
Schließlich (aber sicherlich "not least") möchte ich noch meiner Familie für vielerlei Unterstützung in den letzten Jahren danken.

Contents

List of Figures

List of Tables

List of Algorithms

1. Introduction

1.1. Motivation

Knowledge Discovery in Databases (KDD) is a comparatively new scientific discipline, lying at the intersection of machine learning, statistics, and database theory. It aims to systematically discover relevant patterns that are hidden in large collections of data and are either interesting to human analysts or valuable for making predictions. Depending on the underlying business objectives, KDD tasks may accordingly either be addressed by descriptive or predictive techniques.

The main goal of descriptive data mining is to identify interpretable results that summarize a data set at hand, and point out interesting patterns in the data. A general descriptive data mining task that plays an important role in this work is supervised rule discovery. It aims to identify interesting patterns that describe user-specified properties of interest. The range of corresponding KDD applications is very diverse. One important domain is marketing. Important business goals in this domain include the identification of specific customer groups, for example customers that are likely to churn, or of segments of the population that contain particularly many or few perspective customers, which helps in designing targeted marketing strategies. An example of a challenging medical application is the identification of pathogenic factors.

The goal of predictive data mining is to induce models that allow to reliably derive relevant properties of observations that are not explicitly given in the data. This includes the prediction of future events and classification problems. Even the most prominent examples span many different domains. Information retrieval techniques, for example, aim to predict which documents match a user's information need, based on a query. Fraud detection is another important application. It aims to identify fraudulent behavior, for example fraudulent credit card transactions, often with real-time constraints and a vast amount of data. In the finance business, the separation of "good" from "bad" loans is a typical example of a predictive task.

As a consequence of the variety of applications, the field of KDD has recently gained much attention in both academia and industry. In the academic world, this trend is reflected by an increasing number of publications and a growing participation in annual conferences like the *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* and the *IEEE International Conference on Data Mining*. For both descriptive and predictive analysis tasks a plethora of well understood techniques that apply to the core analytical problems is available in the scientific literature. The industrial commitment leveraged a rapidly growing market of KDD software environments during the last few years. Even the major modern database management systems come nowadays shipped with a set of basic data mining algorithms, reflecting a growing customer demand.

It turns out, however, that most KDD problems are not easily solved by just applying those data mining tools. As the amounts of data to be analyzed as part of daily routines drastically increased over the last decade, new challenges emerged, because standard algorithms that were designed for data of main memory size are no longer applicable. At the same time, even more challenging data mining problems emerged continuously, like the analysis of huge gene sequences, the classification of millions of web sites and news feeds, and recommending countless

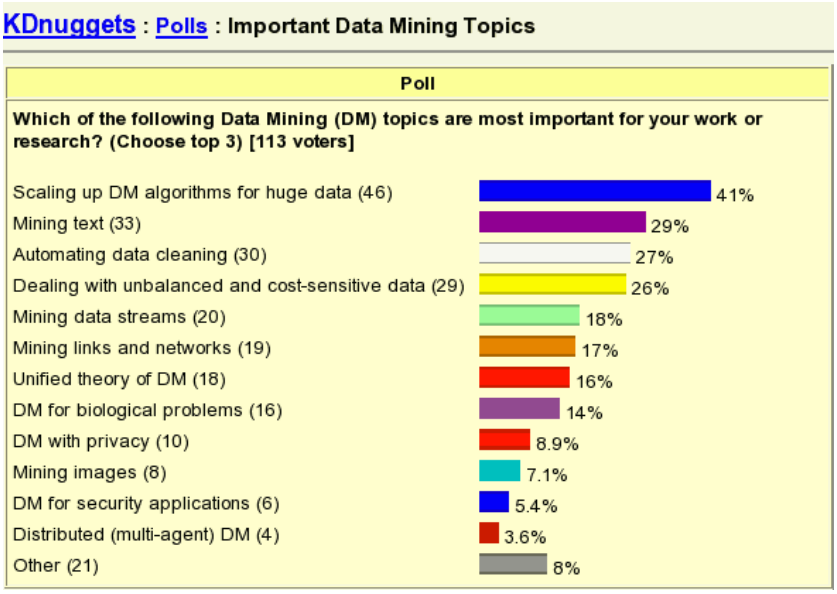


Figure 1.1.: The most important data mining topics due to a KDnuggets survey in 2005.

products to huge customer bases based on behavioral profiles. Comparing the orders of magnitude of the number of data records involved to the response time of systems tolerable in the specific contexts, the challenging problem of addressing such complex tasks with scalable KDD techniques seems inevitable.

Besides, for most KDD applications the data will not be stored in a single database table, but rather be organized in terms of a complex database schema that will most likely be distributed over a large number of geographically distant nodes. In particular larger companies will often store their data locally at each branch or in each major city, but analysis tasks may still refer to global properties, e.g., the global buying behavior of customers. If a single full table-scan takes several days, which is not uncommon in modern data warehouses, then transferring all the data to a single site clearly is no attractive option. Another important aspect that is not well supported by existing solutions is that in many cases new data becomes continuously available. Data mining results may quickly become outdated if they are not adapted to the newest available information. Continuously training from scratch is computationally very demanding, but only very few data mining techniques have successfully been adapted to naturally address this kind of dynamic input directly.

Among all the burdens mentioned above, the large amount of data to be analyzed is most critical for modern KDD applications. This was recently confirmed by a survey (cf. figure 1.1) of the popular forum KDnuggets[1]; scalability was named as the most important data mining topic. This thesis is mostly motivated by scalability aspects of data mining, while favoring generic solutions that can as well be used to tackle the other burdens named above. Hence, variants of the scalable solutions that will be proposed in this work will be discussed for data streams and distributed data. The final part of this work is dedicated to some practical issues of data preprocessing.

[1] http://www.kdnuggets.com/

1.2. Scalable knowledge discovery

Scalability aspects can roughly be characterized as being of a technical, or of a theoretical nature. As a constraint on the technical side, most data mining toolboxes require the data to be analyzed to fit into main memory. This allows for very efficient implementations of data mining algorithms that often drastically outperform solutions that, e.g., access the data via the interfaces of a database management system. However, the dominating constraint that truly hinders practitioners to scale up data mining algorithms to the size of large databases is the super-linear runtime complexity of the core algorithms themselves. For example, even the simple task of selecting a single best classification rule that e.g., conditions on only a single numerical attribute value and compares it to a threshold causes computational costs in $\Omega(n \log n)$ for sample size n. The reason is that the selection involves a step of sorting the data.

In sheer contrast to this observation, mastering data analysis tasks from very large databases requires algorithms with *sub-linear* complexity. It is understood that, in order to meet this constraint, only subsets of the available data may be processed.

One valuable line of research on scalability, most prominently hosted in the frequent itemset mining community[2], tries to minimize the runtime complexity of individual data mining algorithms by exploiting their specific properties, e.g., by designing specific data structures, or by investing much time into technical software optimization. Despite the continuous progress in this field, algorithms that are always guaranteed to find exact solutions clearly cannot scale sub-linearly.

Another approach to foster scalability, more common in practice, is to consider only a small fraction of a database that – in its original form – would be too costly to be analyzed by the chosen data mining algorithm. It is crucial to understand the properties of sampling techniques in specific analytical contexts when following this approach. We still want to be able to give guarantees regarding the quality of our data mining results when working only on a subset of the data. The difference to training from all the data should be marginal.

The main motivation of this work is to provide generic techniques that improve the scalability of data intensive KDD without perceptibly compromising model performance. This thesis will demonstrate that for many data mining tasks sampling is more than a temporary solution that fills the gap until algorithms of better scalability are available. It will be illustrated how a solid theoretical understanding that includes both the statistical foundations of sampling and the nature of the optimization problems solved by data mining techniques helps to avoid the caveats of commonly seen ad hoc sampling heuristics, i.e., techniques that do not allow to provide reasonable guarantees. This thesis establishes a sampling-centered view on learning, based on the insight that the available training data usually is a sample itself.

At the methodological level, this view allows to derive novel practically relevant algorithms, like preprocessing operators that allow to i) enhance the predictive power of existing learning schemes without modifying them, or to ii) explicitly mine patterns that optimize novelty in descriptive settings, where novelty is measured in terms of deviation from given prior knowledge or expectation. Unlike for handcrafted solutions that improve one particular data mining algorithm at a time, the sampling-centered approaches are inherently generic. Later parts of this thesis analyze the predictive power of the presented methods in detail, and investigate their applicability to a broader set of practically important settings, including drifting concepts and distributed data.

[2]For example, the FIMI website hosts a repository of fast implementations and benchmark datasets: `http://fimi.cs.helsinki.fi/`

1.3. A constructivist approach to learning

Data mining subsumes diverse learning tasks and a variety of techniques and algorithms to solve them. It can be expected that novel tasks will continuously emerge in the future, accompanied by specific techniques that address very characteristic aspects. On the analytical side, this work hence follows a more principled approach towards tackling data mining tasks. It is based on discovering similarities between tasks and methods at an abstract, yet operational level. The goal is to gain a thorough understanding of the principles underlying data mining problems by *decomposing* the diverse variety of data mining tasks into a small set of theoretically well-based building blocks. Identifying such components at a proper level of abstraction is a promising approach, because it allows to (re-)compose them in a flexible way to new principled tasks. As an intuitive motivation, a constructive way of reducing one *problem* to another one at an abstract level may prevent us from wasting efforts on the development of redundant *techniques*. This raises the question what the right theoretically well-based building blocks for data mining tasks are, and how they can be utilized as novel problems emerge.

Some questions that will naturally emerge in the context of this thesis and that will be analyzed using the approach sketched above include:

- What is the inherent difference between descriptive supervised rule discovery and classifier induction?
- Which effects do class skews have on utility functions that are used to evaluate models?
- Can stratification be utilized to improve the performance of ensemble techniques?
- What is the inherent difference between optimizing error rate and rankings?

Along the objectives outlined above, this thesis does not cover any individual full case studies; it rather aims to derive building blocks that can easily be compiled into a variety of different scalable, yet accurate knowledge discovery applications. The utility of the established theoretical view will be demonstrated by deriving novel, practically relevant algorithms that address the problems discussed in the last section in a very generic way. Empirical studies on benchmark datasets will be provided to substantiate all claims.

1.4. Outline

This thesis divides into three parts. Part I provides theoretical foundations along with related work (chapters 2 and 3), part II presents novel data mining methods (chapters 4-7), and part III presents a system designed to simplify data preprocessing for KDD (chapters 8 and 9).

Theoretical foundations

Before going into the technical details of machine learning and data mining, this thesis starts (chapter 2) with an overview of existing algorithms and fundamental principles which are central to later parts. The focuses of this thesis is the scalability of data mining applications. Since most learning algorithms cannot cope with huge amounts of data directly, it is common practice to work on sub-samples that fit into main memory and allow to find models in reasonable time. Chapter 3 discusses the foundations of sub-sampling techniques and practically relevant algorithms exploiting them. As will be discussed, uniform sub-sampling can be used to speed up most data mining procedures run on large data sets with a bounded probability to select poor models. The success of ensemble methods like boosting illustrates that sampling from non-uniform distributions may often be an attractive alternative. A short introduction to the family of Monte Carlo algorithms will be given. These algorithms constitute the most important tools when sampling with respect to altered distributions.

Novel supervised learning methods

In chapter 4 the novel concept of *knowledge-based sampling* is presented. This strategy allows to incorporate prior knowledge into supervised data mining, and to turn pattern mining into a sequential process. An algorithm is presented that samples directly from a database using rejection sampling. It is very simple but still allows to "sample out" correlations exactly, which do not have to be qualified by probabilistic estimates. The low complexity of this algorithm allows to apply it to very large databases. A subsequently derived variant for sequential rule discovery is shown to yield small diverse sets of well interpretable rules that characterize a specified property of interest. In a predictive setting these rules may be interpreted as an ensemble of weak classifiers.

Chapter 5 analyzes the performance of a marginally altered algorithm focusing on predictive performance. The conceptual differences between the corresponding algorithm and the most commonly applied boosting algorithm ADABOOST are analyzed and interpreted in coverage spaces, an analysis tool similar to ROC spaces. It is shown that the new algorithm simplifies and improves ADABOOST at the same time. A novel proof is provided that illustrates the connection between accuracy and ranking optimization in this context.

In chapter 6 the novel technique is adapted to streaming data. The refined variant naturally adapts to concept drift and allows to quantify drifts in terms of the base learners. If distributions change slowly, then the technique decomposes the current distribution, which helps to quickly adapt ensembles to the changing components. Sudden changes are addressed by continuously re-estimating the performances of all ensemble members.

In chapter 7 the task of supervised rule discovery is analyzed for distributed databases. The complexity of the resulting learning tasks, formulated in very general terms to cover a broad variety of rule selection metrics, is compared to the complexity of learning the same rules from non-distributed data. Besides, a novel task that aims to characterize differences between databases will be discussed. The theoretical results motivate algorithms based on exhaustively searching the space of all rules. Two algorithms are derived that apply only safe pruning and hence yield

exact results, but still have moderate communication costs. Combinations with knowledge-based sampling are shown to be straightforward.

Support for data preprocessing

Besides being huge, real-world data sets that are analyzed in KDD applications typically have several other unpleasant characteristics. First, the data quality tends to be low, e.g. information is missing, typing errors and outliers compromise reliability, and semantically inconsistent entries do not allow to induce models satisfying the business demands. Second, the data usually cannot directly be fed into data mining algorithms, because most KDD applications make use of data that were originally collected for different purposes. This means that the representation of the data is highly unlikely to fit the demands of a data mining algorithm at hand. As an obvious example, data is often stored in relational databases, but most of the commonly applied data mining techniques require the data to be in attribute-value form, that is, they apply only to inputs taking the form of a single database table.

The last part of this thesis hence discusses the practical embedding of the data mining step into real-world KDD applications. Chapter 8 sketches the general notion of a KDD process, illustrates its iterative nature, and identifies preprocessing as the missing link between data mining and knowledge discovery. The chapter provides an overview of an integrated preprocessing environment called MININGMART; it focuses on setting up and re-using best-practice cases of preprocessing for very large databases.

In chapter 9 several details about MININGMART's meta-data driven software generation are discussed. The MININGMART meta model storing all the meta-data of preprocessing cases is operationalized by a module called the M4 *compiler*, large parts of which were designed and implemented by the author of this thesis. It is illustrated how different levels of abstraction are involved when running the compiler, that is, how very different types of information interact. Synergy effects between the preprocessing environment MININGMART, running on real-world databases to yield a representation that allows for data mining, and the main memory based learning environment YALE used in the data mining part of this thesis are pointed out.

2. Machine Learning – Some Basics

This chapter introduces the most basic concepts from the fields of machine learning and data mining that will be referred to throughout this thesis. It starts with the commonly used formal statistical framework in section 2.1, which applies to supervised and unsupervised learning tasks. In supervised learning, classified examples are supplied to an algorithm, which tries to find an appropriate generalization. Often the goal is to classify previously unseen examples. Assumptions about the data generating process help to define appropriate selection criteria for models, e.g. the error rate of models for samples. For descriptive tasks, similar assumptions allow to decompose a set of observations by assigning each observation to one of a set of different generating processes.

The formal framework used throughout the remainder of this thesis is introduced in section 2.1. Section 2.2 provides an overview of relevant learning tasks. For supervised learning the paradigm of *probably approximately correct (PAC)* learning allows to analyze learnability of "target concepts" from specific classes from a given set of models (hypothesis language) in this framework. This paradigm is briefly discussed in section 2.3. Along with the learning scenarios of rule induction and of discovering "interesting" rules some formal criteria for model selection are introduced in the subsequent section 2.4. Section 2.5 explains the differences between rule selection criteria using the *receiver operator characteristics* (ROC), a tool recently rediscovered by machine learning researchers. Furthermore, it discusses more general learning scenarios than those assumed in section 2.3. In subsequent chapters, learning algorithms often yield *sets* of different rules or other kinds of models. Section 2.6 discusses some general techniques that allow to combine their associated predictions.

2.1. Formal Framework

The overall goal in machine learning is to construct models from classified or unclassified example sets, that allow for a deeper understanding of the data, the data generating process, and/or to predict properties of previously unseen observations.

Different representations of examples lead to learning scenarios of different complexity. The most common representation is derived from propositional logics, leading to data in attribute-value form. In a relational database, attribute-value representations can be thought of as single tables consisting of boolean, nominal, or continuous attributes. Some machine learning algorithms may directly be applied to relational data, because they are capable of "following" foreign key references on demand. If the data is split into several relations of a relational database, then the learning scenario is referred to as relational or multi-relational. More expressive representations, like full first order logics, are not discussed in this thesis.

The set of examples can be considered as a subset of an *instance space* $\mathcal{X}$, the set of all possible examples. Starting with propositional data and representations based on attribute-value pairs, the instance space can be defined as the Cartesian product of all d available domains (attributes) $A_j, 1 \leq j \leq d$, where each domain is a set of possible attribute values. The instance space is hence defined as

$$\mathcal{X} := A_1 \times \ldots \times A_d.$$

In the case of supervised learning, there is an additional set of possible labels or continuous target values $\mathcal{Y}$. A set of n *classified examples* is denoted as

$$\mathcal{E}_n = \{(x_1, y_1), \ldots, (x_n, y_n)\}, \text{ where}$$
$$(x_i, y_i) \in \mathcal{X} \times \mathcal{Y} \text{ for } i \in \{1, \ldots, n\}.$$

Please note that, although examples have indices and are generally given in a specific order, this order has no effect on most of the algorithms studied in the following chapters. For the few algorithms that depend on the order it may generally be assumed in the studied contexts that the examples have been permutated randomly, with any two permutations of examples being equally probable. For this reason, the common but imprecise notion of example *sets* is used, even if referring to ordered sequences of examples.

In the machine learning literature the data for training and validation is usually assumed to follow a common underlying probability distribution with probability density function (pdf) $D : \mathcal{X} \to \mathbb{R}^+$. Examples are sampled *independently* from each other, and are *identically* distributed with respect to this function D. This assumption is referred to as *sampling i.i.d.* in the literature. Sampling n examples i.i.d. from D is equivalent to sampling a single instance from the product density function $D^n : \mathcal{X}^n \to \mathbb{R}^+$,

$$D^n(x_1, \ldots, x_n) := \prod_{i=1}^{n} D(x_i), \quad (\forall x \in \{1, \ldots, n\}) : x_i \in \mathcal{X},$$

because each single example is independently sampled from D.

One of the crucial prerequisites of the probably approximately correct learning paradigm (Valiant, 1984; Kearns & Vazirani, 1994) discussed in section 2.3 is that both the training data used for model selection, and the validation data used to assess the quality of models, are sampled i.i.d. from the *same* underlying distribution.

The case of multi-relational data is more complex, in particular because the notion of a *single example* is less clear. Each example may be spread over several relations, and may thus be represented by sets of tuples. For this reason explicit distributional assumptions are not as common in this field, or examples are defined using a single target relation, as in the case of propositional data. In the latter case, the target relation has associated relations that are considered when searching for intensional characterizations of subsets.

In the simple case of a *finite* instance space $\mathcal{X}$, or of a finite subset of $\mathcal{X}$ with positive weight under D, the probability to observe an (unclassified) example $x \in \mathcal{X}$ under D is denoted as $\Pr_{x \sim D}(x)$. The shorter notation $\Pr_D(x)$ is used, if the variable is clear from the context. If the underlying distribution is also obvious, then all subscripts are omitted.

Even if $\mathcal{X}$ is not finite, for typical data mining applications the formal requirements are still not very complex. The total weight of $\mathcal{X}$ may be assumed to be finite, and there are relevant subsets of $\mathcal{X}$ that have a strictly positive weight. The probability to observe an instance from a compact subset $W \subseteq \mathcal{X}$ is denoted as $\Pr_D[W]$. It is equivalent to

$$\Pr_D[W] = \int_{x \in W} D(x)\, dx = \int_D I[x \in W]\, dx,$$

where $I : \{\text{true}, \text{false}\} \to \{1, 0\}$ denotes the indicator function. This function evaluates to 1, iff its argument evaluates to true. If $\mathcal{X}$ is continuous, then the considered density functions are assumed to be well-behaved throughout this work, in the sense specified in the appendix of (Blumer et al., 1989). This property requires not only that for the probability distribution induced by the density

function all considered subsets of $\mathcal{X}$ are Borel sets, but also that specific differences between such sets are measurable. This should not narrow the applicability of the presented results in practice, and is not explicitly mentioned, henceforth.

2.2. Learning Tasks

In the machine learning literature a variety of different tasks have been studied. Traditionally, the considered learning tasks are referred to as either *supervised* or *unsupervised*. For the former kind of tasks there are known classes of observations which are represented by a *target attribute* $\mathcal{Y}$, assigning a *class label* to each observation. The family of unsupervised problems contains all kinds of tasks for which no classes are given a priori, and for which the identification of regularities in the data, e.g. patterns, classes, or an hierarchical organization of observations, is up to the learner.

2.2.1. Classification

The most intensively studied task in machine learning is *classification*. The goal is to fit a classifier (function) $h : \mathcal{X} \to \mathcal{Y}$ to a given set of training data, aiming at an accurate classification of unclassified observations in the future. This supervised learning problem can be addressed in different frameworks. *Logical learning approaches* typically aim at the identification of a set of valid rules or other kinds of discrete models. Each model, like a rule stated in a restricted form of first order logic, makes a prediction for a subset of the universe of discourse. It is correct, if and only if all of its predictions fit the data. For many domains the identification of perfectly matching models is unrealistic, which motivates a relaxation of this framework. The most successful relaxation assumes that the data is the result of a stationary stochastic process. In this setting, the goal is to fit a model to the data, that has a low risk (probability) to err. The training data can be considered to be a sample drawn from a distribution underlying the universe of discourse, typically referred to as an *instance space* $\mathcal{X}$ in this case. This space contains all possible observations that may be sampled with respect to the density function $D : \mathcal{X} \to \mathbb{R}^+$. In this setting, there is usually a risk that the learner is provided with a poor sample, which inevitably may lead to a poor model. Details on this learning framework are discussed in section 2.3.

2.2.2. Regression

A straightforward generalization of the task of classification does no longer require the target quantity (or label) $\mathcal{Y}$ to be a nominal attribute, but also allows for continuous targets, e.g. $\mathcal{Y} = \mathbb{R}$. In this case, the problem is to fit a function $h : \mathcal{X} \to \mathbb{R}$ to the training data, which deviates as least as possible from the true target values of future observations $x \in \mathcal{X}$. Unlike for classification, a prediction is no longer just correct or wrong, but there is a continuous degree of deviation of predictions from true values. For an example (x, y) this degree of deviation is captured by a so-called *loss function*

$$L(h(x), y) \mapsto \text{loss} \in \mathbb{R}^+,$$

mapping each tuple of a predicted target value $h(x)$ and true value y to a single positive loss that penalizes errors of the model h. This learning problem is referred to as *regression*. The *empirical* risk R_{emp} of a model (hypothesis) h is the total loss when evaluating on a training set $\mathcal{E}$:

$$R_{emp}(h, \mathcal{E}) := \sum_{(x,y)\in\mathcal{E}} L(h(x), y).$$

Similar to probably approximately correct learning (cf. section 2.3), this task usually assumes a fixed but unknown probability density function D underlying the space $\mathcal{X} \times \mathcal{Y}$. This function specifies the density of each observable $(x, y) \in \mathcal{X} \times \mathcal{Y}$, and it is also used to define the *true* risk

$$R_D(h) := \int_D L(h(x), y)\, dx\, dy,$$

which is to be minimized by learning algorithms when selecting a model h. On the one hand, classification is subsumed as a specific case of regression when choosing the $0/1$ *loss function*. This function penalizes each misclassification by assigning a loss of 1, while it defines the loss of correct predictions to be 0. On the other hand, if the costs of misclassifications vary, or if the goal is to fit a classifier that estimates the conditional probabilities of each class $y \in \mathcal{Y}$ for each observation $x \in \mathcal{X}$, then the task of classification requires loss functions that are more complex than $0/1$ loss. In this case, the task of classification shares several aspects of regression. Some corresponding loss functions and utility functions are discussed in section 2.4.

2.2.3. Subgroup discovery

Subgroup discovery (Klösgen, 2002) is a supervised learning task that is discussed at several points in this work. It aims to detect well interpretable and interesting rules.

Formal framework

In the formal framework of subgroup discovery there is a *property of interest*; it is basically identical to nominal class labels in the field of classifier induction. Often the property of interest is boolean, for example "customer responds to mailing campaign" or "driver was recently involved in a car accident". For simplicity, it is also referred to as a class label and denoted as $\mathcal{Y}$. The property of interest can hence be thought of as an attribute generated by a target function $f : \mathcal{X} \to \mathcal{Y}$, where f assigns a label to each unclassified instance $x \in \mathcal{X}$. The function f is assumed to be fixed but unknown to the learner, which aims to find a good approximation. The *functional* dependency of $\mathcal{Y}$ on $\mathcal{X}$ is not required, and basically only introduced to simplify formal aspects. The same concepts apply for probabilistic dependencies.

In contrast to classification and regression, the rules found by subgroup discovery are mainly used for *descriptive* data analysis tasks. Nevertheless, such rules are also useful in predictive settings.

The factors considered to make rules interesting depend on the user and application at hand. Among the subjective factors often named in this context are unexpectedness, novelty, and actionability. A rule is unexpected, if it makes predictions that deviate from a user's expectation. This aspect is similar to novelty. A rule is novel, if it is not yet known to a user. Finally, not all rules offer the option to take some kind of beneficial actions. Actionability generally depends on the user's abilities and on the context, which suggests to use an explicit model accounting for these aspects.

In practice different heuristics are used for discovering interesting rules. Measures for rule interestingness are formally stated as *utility* or *quality functions*, a specific type of rule selection metric that can be considered to be a parameter of the learning task itself. Let $\mathcal{H}$ denote a set of syntactically valid rules (or any broader class of models, respectively), and let $(\mathcal{X} \times \mathcal{Y})^{\mathbb{N}}$ denote the set of all finite sequences of examples from $\mathcal{X} \times \mathcal{Y}$. Then a utility function $\hat{Q} : \mathcal{H} \times (\mathcal{X} \times \mathcal{Y})^{\mathbb{N}} \to \mathbb{R}$ maps each tuple $(r, \mathcal{E})$ of a rule $r \in \mathcal{H}$ and example set $\mathcal{E}$ to a real-valued utility score. A typical subgroup discovery task is to identify a set $H^* \subset \mathcal{H}$ of k best rules with

respect to any given utility function $\hat{Q}$; in formal terms:

$$(\forall r \in H^*)(\forall r' \in \mathcal{H} \setminus H^*) : \hat{Q}(r, \mathcal{E}) \geq \hat{Q}(r', \mathcal{E}). \tag{2.1}$$

For subgroup discovery, classification rules (cf. Def. 13, p. 21) are the main representation language. The interestingness of rules and the requirements rule metrics should meet have been discussed by various authors, e.g. by Piatetsky-Shapiro (1991), Klösgen (1996), Silberschatz and Tuzhilin (1996), and Lavrac et al. (1999). Section 2.4 provides an overview of the most relevant evaluation criteria.

Existing approaches

Eqn. (2.1) above formulates subgroup discovery as an optimization problem. Three different strategies of searching for interesting rules have been proposed in the literature on subgroup discovery, exhaustive, probabilistic, and heuristic search.

Exhaustive EXPLORA by Klösgen (1996) and MIDOS by Wrobel (1997) are examples for tackling subgroup discovery by exhaustively evaluating the set of rule candidates. The rules are ordered by generality, which often allows to prune large parts of the search space. Only safe pruning based on optimistic estimates is applied. An algorithm recently proposed by Atzmüller and Puppe (2006) for mining subgroups from propositional data follows a two-step approach; it builds up an FP-growth data structure (Han et al., 2000) adapted to supervised settings in the first step, which can then be used to efficiently extract a set of best subgroups in the second. The advantage of all these exhaustive search strategies is that they allow to find the k best subgroups reliably.

Probabilistic Finding subgroups on uniform sub-samples of the original data is a straightforward method to speed up the search process. As shown by Scheffer and Wrobel (2002), most of the utility functions commonly used for subgroup discovery are well suited to be combined with adaptive sampling. This sampling technique reads examples sequentially, and continuously updates upper bounds for the sample errors based on the data read so far. That way probabilistic guarantees not to miss any of the *approximately* k best subgroups can be given much quicker than when following exhaustive approaches. This line of research is discussed in subsection 3.3.2.

Heuristic Heuristic search strategies are fast, but do not come with any guarantee of finding the most interesting patterns. One recent example implementing a heuristic search is a variant of CN2. By adapting its rule selection metric to a subgroup discovery utility function, the well known CN2 classifier has been turned into CN2-SD (Lavrac et al., 2004b). As a second modification, the sequential cover approach of CN2 has been replaced by a heuristic strategy to *reweight* examples. This algorithm will be discussed in more detail in section 4.3.

When allowing for broader model classes, the task of classifier induction is subsumed by subgroup discovery; predictive accuracy is just one specific instance of a utility function. Hybrid learning tasks that lie between classical subgroup discovery and classification will be discussed in chapter 4.

2.2.4. Clustering

In several domains there is no a priori target attribute, but – similar to subgroup discovery – the goal of learning is to identify homogeneous subsets of reasonable size, showing different

variable distributions than those observed for the overall population. A corresponding machine learning task referred to as *clustering* has been derived from statistical cluster analysis. Classical approaches to clustering yield (disjoint) partitions $C_1, \ldots, C_k$ of the supplied example sets, so that $\biguplus_{i=1}^{k} C_i = \mathcal{E}$. Compared to classification, it is harder to assess the quality of clusterings; a priori there is no clear objective. To overcome this problem, a variety of formal objective functions have been proposed in the literature on clustering. They primarily aim to define similarity, and to trade-off between (i) the average similarity of instances sharing a cluster and (ii) the average difference between instances of different clusters. For a given distance measure $\Delta : \mathcal{X} \times \mathcal{X} \to \mathbb{R}^+$, with 0 denoting the highest similarity, and a given number k of clusters, a simple formulation of the clustering task is to partition $\mathcal{E}$ into disjoint subsets $C_1, \ldots, C_k$ in a way that minimizes the function

$$\sum_{i=1}^{k} \sum_{x_m, x_n \in C_i | m > n} \Delta(x_m, x_n).$$

Clustering is not directly addressed in this thesis. It is mainly mentioned for completeness, and because it shows some interesting similarities to subgroup discovery. Formally the main difference is, that clustering does neither require nor support a designated target attribute; it is an unsupervised learning task.

2.2.5. Frequent itemset and association rule mining

Another well-recognized unsupervised learning task is frequent itemset mining (Agrawal & Srikant, 1994). Most approaches for frequent itemset mining require all attributes to be boolean, i.e. $A_1 = \ldots = A_d = \{0, 1\}$, where 0 means absence and 1 represents presence of an event. For a given example set $\mathcal{E}$ the goal is to identify all subsets I of $\{A_1, \ldots, A_d\}$ (itemsets) for which the *support*

$$\sup(I, \mathcal{E}) := \frac{|\{\ (\forall A_j \in I) : A_j(e) = 1 \mid e \in \mathcal{E}\ \}|}{|\mathcal{E}|}$$

exceeds a user-given threshold min_sup. These frequent itemsets I can be used in a second step to generate the set of all *association rules*. Such rules need to exceed a user-given *precision* (or support) min_fr, which is defined as the fraction of examples that are classified correctly by such a rule. A more detailed definition is provided in section 2.4.3.

Association rule mining also shows some similarities to subgroup discovery. It yields sets of rules that might be considered interesting. Its unsupervised nature can be circumvented, so that only rules predicting values for a specific target attribute are reported, or it can be seen as a generalization that considers each attribute to be a potential target attribute. An intrinsic difference to subgroup discovery is, that association rule mining is a constraint-based search problem, rather than on optimization problem. The size of the resulting rule set is not known in advance, and association rule mining does not optimize subgroup utility functions. As a consequence, running an association rule mining algorithm to generate candidates for subgroup discovery will usually yield a large superset of the k best subgroups. Moreover, there is even a risk that some of the best subgroups will still not be contained in such a large set of candidate rules. In chapter 7 this issue will be discussed in more detail.

Finally it should be noted that, although there is a close connection between several learning tasks, there is no taxonomy of tasks in terms of true generalization. For example, regression may be considered to subsume the task of classification, but as the large number of publications on specific classification techniques illustrates, general regression techniques do not perform well

in classification domains. In turn, regression is sometimes addressed by classification techniques after a step of discretization, that is, after mapping the continuous responses to a discrete set of intervals. Subgroup discovery, clustering, and association rule mining have several properties in common, and are in theory, tackled using a similar catalog of methods. However, the task definitions differ, so the same catalog of methods is compiled into different algorithms, following different objectives, and having implementations with different strengths and weaknesses. One of the objectives of this work is to identify common theoretically well-based building blocks of data mining tasks that allow to generalize specific results, or to even address tasks by re-using algorithms that were tailored towards different tasks.

2.3. Probably Approximately Correct Learning

Most parts of this work address supervised learning tasks. The most successful theoretical framework for supervised machine learning has been formalized by Valiant (1984). The model of probably approximately correct (PAC) learning allows to investigate the complexity of classification problems. A learner is not required to identify the target concept underlying a classified example set exactly, as e.g., in the identification in the limit paradigm known from language identification (Gold, 1967). For a PAC learner it is sufficient to yield a good approximation of the target concept with high probability instead.

Only the most important definitions and some results of the PAC model are summarized in this section, since this field has been discussed elaborately by various authors. For example, Kearns and Vazirani (1994) and Fischer (1999) provide compact introductions.

The original version of the PAC model is described in 2.3.1. It depends on some assumptions that ease the analysis of learning algorithms, but are rather unrealistic from a practical point of view. A weaker definition of learnability, particularly useful in the context of *boosting* classifiers, is given in subsection 2.3.2. Additionally, another generalization of the learnability framework is presented, the so-called *agnostic* PAC model. It is based on more realistic assumptions that can basically be shown to hamper learnability.

2.3.1. PAC learnability of concept classes

There are different possible assumptions of how a target attribute $\mathcal{Y}$ may depend on an instance space $\mathcal{X}$. The original PAC learning framework assumes a functional dependency between each instance x and its label y. This dependency can formally be represented in terms of a *target function* $f : \mathcal{X} \to \mathcal{Y}$. The label $\mathcal{Y}$ is assumed to be boolean, so each target function simply distinguishes positive from negative examples. This motivates the simplification of target functions to *concepts* $c \subseteq \mathcal{X}$ that contain exactly the positive examples. The learner may rely on the fact that the target function comes from a *concept class* $\mathcal{C}$. Boolean expressions of a specific syntactical form, hyperplanes in Euclidian spaces, and decision trees are typical examples of concept classes.

The target class c, e.g. a decision tree that perfectly identifies the positive examples, is of course unknown to the learner. Hence, the goal is to select a model, referred to as a *hypothesis* in PAC terminology, from a *hypothesis space* $\mathcal{H}$, which approximates the unknown target concept well. Just as the concept class, the hypothesis space $\mathcal{H}$ is a subset of the powerset $\mathcal{P}(\mathcal{X})$ of the instance space $\mathcal{X}$. The quality of an approximation is stated with respect to an unknown probability density function (pdf) D underlying the data, rather than with respect to the available training data.

Definition 1 *For a given probability density function* $D : \mathcal{X} \to \mathbb{R}^+$*, two concepts*

$$c \subseteq \mathcal{X} \quad \textit{and} \quad h \subseteq \mathcal{X}$$

are called ϵ*-close for any* $\epsilon \in [0, 1]$*, if*

$$Pr_D\left[(c \setminus h) \cup (h \setminus c)\right] \leq \epsilon.$$

The learner's choice of a model depends on the training set, of course, which is assumed to be sampled i.i.d. Samples always bear a small risk of not being informative, or of even being misleading. For example, it may happen that the sample suggests a much simpler than the correct target concept c, because an important subset of the instance space is drastically underrepresented. The reader may want to think of a very extreme case: a single example might be sampled over and over again. It consequently cannot be expected that a learner always selects a good model when being provided with only a finite number of examples. The PAC model takes the risk of poor samples into account by allowing that the learner *fails* with a probability of at most $\delta \in (0, 1)$, an additional *confidence* parameter.

A crucial assumption of PAC learning is, that the model is deployed in a setting that shares the (unknown) pdf D underlying the training data. Assumptions about D are avoided by requiring that PAC algorithms succeed for *any* choice of D with a probability of at least $1 - \delta$. The following definition states these ideas more precisely[1].

Definition 2 *A concept class* $\mathcal{C} \subseteq \mathcal{P}(\mathcal{X})$ *is said to be* PAC learnable *from a hypothesis space* $\mathcal{H} \subseteq \mathcal{P}(\mathcal{X})$ *if there exists an algorithm* A *that, for any choice of* $\delta, \epsilon \in (0, 1)$*, any* $c \in \mathcal{C}$ *and every probability distribution* D *over* $\mathcal{X}$*, outputs with probability at least* $1 - \delta$ *a hypothesis* $h \in \mathcal{H}$ *that is* ϵ*-close to* c*, if* A *is provided with an i.i.d. sample* $\mathcal{E} \sim D^m$ *(of size* m*), where* m *is upper-bounded by a polynomial in* $1/\epsilon$ *and* $1/\delta$*.*

Please note that definition 2 is based solely on the information about a target class that can be derived from samples of a specific size. An algorithm is simply considered to be a recursive function, mapping sets of classified samples to $\mathcal{H}$. If the *information* extractable from samples is not sufficient to identify the target class, then it is not necessary to consider specific algorithms in order to prove non-learnability. If learning is possible, however, then one is interested in concrete algorithms and their efficiency. Definition 2 does not demand polynomial time complexity for the identification procedure that yields an ϵ-close hypothesis; hence, this notion of learnability induces a broader complexity class (unless NP=RP) than that which corresponds to *efficiently* learnable target classes as defined below.

Definition 3 *A concept class* $\mathcal{C} \subseteq \mathcal{P}(\mathcal{X})$ *is called* efficiently PAC learnable *from a hypothesis class* $\mathcal{H} \subseteq \mathcal{P}(\mathcal{X})$ *if it is PAC learnable from* $\mathcal{H}$ *and one of the algorithms satisfying the constraints given in Def. 2 has a runtime polynomially bounded in* $1/\epsilon$ *and* $1/\delta$*.*

An example of a concept class $\mathcal{C}$ which – choosing $\mathcal{H} = \mathcal{C}$ – is PAC learnable, but not efficiently, is k-term DNF[2]. For a set of boolean variables a DNF (disjunctive normal form) is a disjunction of conjunctions of literals. The class k-term DNF consists of all DNF formulae containing at most k conjunctions. It is interesting to note, that k-term DNF is efficiently PAC learnable using another hypothesis language, namely k-term CNF, consisting of all conjunctions of disjunctions that contain at most k literals. These results indicate that the information theoretic concept of

[1] $\mathcal{P}(\mathcal{X})$ denotes the power set of $\mathcal{X}$.

[2] To be precise: This statement holds for all $k \geq 2$.

learnability (Def. 2) does not imply the concept of efficient learnability (Def. 3), and that even for these well structured base problems of machine learning the choice of an appropriate hypothesis space (or *model class*) has a serious impact on learnability.

The most fundamental results for PAC learnability are based on the Vapnik-Chervonenkis dimension of hypothesis spaces.

Definition 4 *The* Vapnik-Chervonenkis dimension *of a concept class $\mathcal{H}$, denoted as* $\mathrm{VCdim}(\mathcal{H})$*, is defined as the cardinality $|\mathcal{E}|$ of a largest example set $\mathcal{E} \subseteq \mathcal{X}$ meeting the following constraint: For each potential assignment of labels to $\mathcal{E}$ there exists a consistent hypothesis in $\mathcal{H}$, formally:*

$$\mathrm{VCdim}(\mathcal{H}) \geq \nu \;:\Leftrightarrow\; (\exists \mathcal{E} \in \mathcal{X}, |\mathcal{E}| \geq \nu)(\forall c \in \mathcal{P}(\mathcal{E}))(\exists h \in \mathcal{H}) : \mathcal{E} \cap h = c$$

If the above property holds for arbitrarily large $\mathcal{E}$, then we define $\mathrm{VCdim}(\mathcal{H}) := \infty$.

It is easily seen that any finite concept class has a finite VCdim, but the same holds for many practically relevant infinite concept classes. An example of the latter are halfspaces in $\mathbb{R}^n$ (classes separable by hyperplanes); they have a VCdim of $n + 1$.

Blumer et al. (1989) proved the following theorem, which is one of the foundations of algorithmic learning theory.

Theorem 1 *Any concept class $\mathcal{C} \subseteq \mathcal{P}(\mathcal{X})$ with finite* VCdim *is PAC learnable from $\mathcal{H} = \mathcal{C}$. Any algorithm that outputs a concept $h \in \mathcal{C}$ that is consistent with any given sample S labeled according to a concept $c \in \mathcal{C}$ is a PAC learning algorithm in this case. For a given maximal error rate ϵ, confidence parameter δ, and sample size*

$$|S| \geq \max\left(\frac{4}{\epsilon}\log\frac{2}{\delta}, \frac{8 \cdot \mathrm{VCdim}(\mathcal{C})}{\epsilon}\log\frac{13}{\epsilon}\right)$$

it fails to select a hypothesis that is ϵ-close to c with a probability of at most δ.

There is a corresponding negative result shown by the same authors:

Theorem 2 *For any concept class*[3] *$\mathcal{C} \subseteq \mathcal{P}(\mathcal{X})$, $\epsilon < 1/2$, and a sample size*

$$|S| < \max\left(\frac{1-\epsilon}{\epsilon}\ln\frac{1}{\delta}, \mathrm{VCdim}(\mathcal{C}) \cdot (1 - 2(\epsilon(1-\delta) + \delta))\right)$$

every algorithm must fail with a probability of at least δ to yield an ϵ-close hypothesis. No concept class with infinite VCdim *is PAC learnable from any hypothesis space.*

For concept classes with infinite VCdim there is often a natural structure of $\mathcal{H}$ and $\mathcal{C}$, inducing a complexity measure for hypotheses. In this case, the VCdim is often finite, if only hypotheses up to a specific maximal complexity are considered. In other terms, if the sample complexity is allowed to grow polynomially in a complexity parameter, like the maximal considered depth of decision trees, then PAC learnability (defined slightly different) can often be shown, although the VCdim of the embedding concept class is infinite. For brevity, this aspect is not discussed here. For proofs and further reading please refer to (Kearns & Vazirani, 1994).

[3] To be precise, it is necessary to claim that $\mathcal{C}$ is non-trivial, i.e. that it contains at least 3 different concepts.

2.3.2. Weakening the notion of learnability

The definitions provided in the last subsection address tasks in which each target concept in $\mathcal{C}$ can be approximated arbitrarily well by a concept taken from $\mathcal{H}$. Practical experiences made with most learning algorithms indicate, that it is unrealistic to expect arbitrarily good performances. Still, for real-world datasets the induced models almost always perform significantly better than random guessing. The notion of *weak* learnability seems to reflect this observed capability of learning algorithms to a certain extent. The following definition is simpler than e.g., the one used by Kearns and Vazirani (1994). It does neither distinguish between hypotheses of different length, nor does it exploit a complexity structure over $\mathcal{H}$, like the depth of decision trees. One of the consequences is, that the required sample size m may be any constant.

Definition 5 *A concept class $\mathcal{C} \subseteq \mathcal{P}(\mathcal{X})$ is said to be* weakly PAC learnable *from a hypothesis class $\mathcal{H} \subseteq \mathcal{P}(\mathcal{X})$ if there exists an algorithm A, with fixed $\epsilon < 1/2$ and $\delta \in (0, 1]$, so that for any $c \in \mathcal{C}$, and for every pdf D over $\mathcal{X}$, algorithm A provided with an i.i.d. sample of any fixed size outputs with probability at least $1 - \delta$ a hypothesis $h \in \mathcal{H}$ that is ϵ-close to c.*

Although this notion of learnability seems far less restrictive at the first sight, weak and strong learnability have constructively been shown to be equivalent by Schapire (1990) when the choice of $\mathcal{H}$ is not fixed. Boosting algorithms increase the predictive strength of a weak learner by invoking it several times for altered distributions, that is, in combination with a specific kind of sub-sampling or in combination with example weights (cf. section 3.4.2). The result is a weighted majority vote over base models predictions (cf. section 2.6), which usually implies that the boosting algorithm selects its models from a hypothesis space that is more expressive than the one used by its base learner. Although this learning technique is very successful in practice, the early theoretical assumptions of weak learnability, which originally motivated it, are obviously violated in practice. One point is, that target functions usually cannot be assumed to lie in any a priori known concept class. Another one is, that the target label is usually rather a random variable than functionally dependent on each $x \in \mathcal{X}$. This implies, that there is often a certain amount of irreducible error in the data, regardless of the choice of any specific model class. Boosting will be discussed more elaborately in chapter 5.

2.3.3. Agnostic PAC learning

With their *agnostic PAC learning model*, Kearns et al. (1992) try to overcome some aspects of original PAC learning that are unrealistic in practice. The main difference, apart from various generalizations of the original model, is that the assumption of any a priori knowledge about a restricted target concept class $\mathcal{C}$ is weakened. The agnostic learning model makes use of a *touchstone class* $\mathcal{T}$ instead; it is assumed that any target concept $c \in \mathcal{C}$ can be approximated by a concept in $\mathcal{T}$, without any further demands on $\mathcal{C}$. The notion of a target concept *class* is basically dropped.

As a second difference to the original PAC-learning model, it is no longer required to approximate the target concept arbitrarily well. It is sufficient if the learner outputs a model h from any hypothesis class $\mathcal{H}$ which is ϵ-close to the best model in $\mathcal{T}$, while in the original PAC model h is required to be ϵ-close to the target concept c itself. The constraint of ϵ-closeness needs to hold with a probability of at least $1 - \delta$, where ϵ and δ are again parameters of the learner, and the number of training examples m is bounded by $\tilde{m}(1/\epsilon, 1/\delta)$ for a fixed polynomial function $\tilde{m}$.

As a third point, Kearns et al. (1992) extend the PAC model to be capable of capturing probabilistic dependencies between $\mathcal{X}$ and $\mathcal{Y}$, while the original PAC model assumes functional dependencies. In this more general setting the learner models the conditional distribution $Pr(y \mid x)$

for each label $y \in \mathcal{Y}$ and example $x \in \mathcal{X}$, or tries to yield a model close to Bayes' decision rule, which predicts the most probable class for each $x \in \mathcal{X}$. The extension of the PAC model is formally achieved by distinguishing between a *true* label $\mathcal{Y}'$ and an *observed* label $\mathcal{Y}$. The same extension allows to model different kinds of noise, which have also been studied as extensions to the original PAC model: *White noise* at a rate of η means, that with a probability of η the label of the (boolean) target attribute is flipped. White noise changes the data unsystematically, so it can be tolerated up to a certain rate by several learners. *Malicious noise* is a model for systematic errors of the worst kind, only bounded by the noise rate η. The reader may want to think of this kind of noise as an "opponent", who analyzes the learning algorithm and the training sample at hand. The opponent then selects a fraction of up to η of all the examples and flips the corresponding labels, following the objective to make the learning algorithm perform as badly as possible. For these noise models Fischer (1999) summarizes some important results on PAC learnability and the corresponding increase in sample complexity.

Kearns et al. (1992) show that learnability in their agnostic PAC-learning model is at least as hard as original PAC-learning with the class label altered by malicious noise. This means that, unless NP=RP, the rather simple problem of learning monomials over boolean attributes is already intractable. As illustrated by the algorithm T2 by Auer et al. (1995), the agnostic PAC model still allows for practically applicable learners. Exploiting one of the results presented in (Kearns et al., 1992), an efficient algorithm selecting any model in $\mathcal{H}$ that has a minimal disagreement (training error) is an agnostic PAC learning algorithm, if $\mathcal{H}$ has a finite VCdim, or a VCdim polynomially bounded in an input complexity parameter. The VCdim of depth-bounded decision trees is finite. T2 exhaustively searches the space of all decision trees with a depth of 2, so it guarantees to output a tree from this class that minimizes the disagreement.

It is no surprise that minimizing the training error is a reasonable strategy for minimizing the generalization error, unless the model class allows to fit arbitrarily complex models to the data. The fact that many unrealistic assumptions were removed from this last PAC learning model is attractive on the one hand, but consequently makes it much harder to derive strong results, on the other. This is one of the implications of the *no free lunch theorem* by Wolpert and Macready (1997): No reasonable guarantees on the performance of learning algorithms can be given without introducing any assumptions or exploiting any domain-specific knowledge. Please note that sampling i.i.d. from the same underlying distribution at training and at application time remains as one of the last assumptions in the agnostic PAC learning model. Thus, the weak results that can be derived in this framework apply to the very general class of problems that share this assumption. Most of the sampling-based techniques discussed in this thesis make no further assumptions either, so they can well be analyzed in frameworks similar to the PAC model.

2.4. Model selection criteria

The induction of models from classified examples has been studied extensively in the machine learning literature throughout the last decades. A variety of metrics like predictive accuracy, precision, or the binomial test function have been suggested to formalize the notions of interestingness and usefulness of models. There are several learning tasks that can be formulated as optimization problems with respect to a specific metric. Classifier induction and subgroup discovery are two important examples. The following paragraphs provide definitions of the most relevant selection metrics.

2.4.1. General classifier selection criteria

The goal when training classifiers in general is to select a predictive model that accurately separates positive from negative examples.

Definition 6 *The (predictive)* accuracy *of a model* $h : \mathcal{X} \to \mathcal{Y}$ *with respect to a pdf* $D : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^+$ *is defined as*

$$\mathrm{ACC}_D(h) \quad := \quad \Pr_{(x,y)\sim D}[h(x) = y] .$$

The error rate *of* h *is defined as* $\mathrm{Err}_D(h) := 1 - \mathrm{ACC}_D(h)$.

These definitions allow to formulate the classifier induction task – previously discussed in the setting of PAC learning – in terms of a formal optimization problem.

Definition 7 *For a hypothesis space* $\mathcal{H}$*, an instance space* $\mathcal{X}$*, a nominal target attribute* $\mathcal{Y}$*, and a (usually unknown) density function* $D : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^+$*, the* task of classification *is to find a model* $h \in \mathcal{H}$ *that maximizes* $\mathrm{ACC}_D(h)$*, or that minimizes* $\mathrm{Err}_D(h)$*, respectively.*

For the process of constructing such models in a greedy general-to-specific manner, but also to evaluate complete models, impurity criteria have successfully been applied. In the best case, a model can reliably separate the different classes of $\mathcal{Y}$. This corresponds to a constructive partitioning of $\mathcal{X}$ with subsets that are pure with respect to the classes. If none of the candidates separates the classes perfectly, then choosing a candidate with highest resulting purity is one way to select classifiers. Top-down induction of decision trees (e.g., Quinlan (1993)) is the most prominent, but not the only learning approach that applies impurity criteria. This approach starts to partition the data recursively, each time selecting a split that leads to the purest possible subsets.

The *entropy* is the best-known impurity criterion. It is an information-theoretic measure (Shannon & Weaver, 1969), evaluating the expected average number of bits that are required to encode class labels. If class i occurs with a probability of p_i, then it can be encoded by $\log \frac{1}{p_i} = -\log p_i$ bits in the best case. Weighting these encoding lengths with the probability of each class we reach at the well-known entropy measure.

Definition 8 *For a nominal target attribute* $\mathcal{Y}$ *the* entropy *of an example set* $\mathcal{E}$ *is defined as*

$$\mathrm{Ent}(\mathcal{E}) = -\sum_{y' \in \mathcal{Y}} \frac{|\{y = y' \mid (x,y) \in \mathcal{E}\}|}{|\mathcal{E}|} \cdot \log\left(\frac{|\{y = y' \mid (x,y) \in \mathcal{E}\}|}{|\mathcal{E}|}\right).$$

To evaluate the utility of splitting $\mathcal{E}$ into v disjoint subsets, $\mathcal{E}^{(1)}, \ldots, \mathcal{E}^{(v)}$, the entropy of each subset is weighted by the fraction of covered examples:

$$\mathrm{Ent}(\{\mathcal{E}^{(1)}, \ldots, \mathcal{E}^{(k)}\}) = \sum_{i=1}^{v} \frac{|\mathcal{E}^{(i)}|}{|\mathcal{E}|} \cdot \mathrm{Ent}(\mathcal{E}^{(i)}).$$

The same criterion can be stated with respect to a pdf underlying the data, e.g. to capture the generalization impurity of a model:

Definition 9 *Let* $D : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^+$ *denote a pdf,* $T : \mathcal{X} \to \{1, \ldots, v\}$ *be a function that partitions* $\mathcal{X}$ *into* v *disjoint subsets* $\{C^{(1)}, \ldots, C^{(v)}\}$*, and let*

$$p_{i,y} := \Pr_D(y = y' \mid (x, y') \in C^{(i)})$$

abbreviate the conditional probability of class $y \in \mathcal{Y}$ *in partition* $C^{(i)}$. *Then the* generalization entropy *is defined as*

$$\mathrm{Ent}_D(T) := -\sum_{i=1}^{v} \mathrm{Pr}_D\left[C^{(i)}\right] \cdot \left(\sum_{y \in \mathcal{Y}} p_{i,y} \log p_{i,y}\right).$$

The decision tree induction algorithms C4.5 (Quinlan, 1993), as well as the WEKA reimplementation J48 (Witten & Frank, 2000) used in later parts of this thesis, are based on the principle of heuristically minimizing entropy at the leaves of a small decision tree. Large trees are known to overfit to the training data, which means that the training error is considerably lower than the generalization error. For this reason, most of the intelligence of decision tree induction algorithms addresses questions like "When to stop growing a tree?", or "How to prune, so that predictive accuracy is not compromised by overfitting?".

Another important impurity metric is the *Gini Index*, which is known from various statistical contexts, but may be used to induce decision trees, as well.

Definition 10 *For an example set* $\mathcal{E}$ *and nominal target attribute* $\mathcal{Y}$ *the Gini index is defined as*

$$\begin{aligned}\mathrm{Gini}(\mathcal{E}) &:= \sum_{y_i, y_j \in \mathcal{Y}, y_i \neq y_j} \frac{|\{y = y_i \mid (x,y) \in \mathcal{E}\}|}{|\mathcal{E}|} \cdot \frac{|\{y = y_j \mid (x,y) \in \mathcal{E}\}|}{|\mathcal{E}|} \\ &= 1 - \sum_{y_i \in \mathcal{Y}} \left(\frac{|\{y = y_i \mid (x,y) \in \mathcal{E}\}|}{|\mathcal{E}|}\right)^2.\end{aligned}$$

Similar to entropy, the Gini index for splits $\mathcal{E}^{(1)}, \ldots, \mathcal{E}^{(v)}$ partitioning $\mathcal{E}$ into disjoint subsets is defined by weighting the individual subsets by the fraction of examples they contain:

$$\mathrm{Gini}(\{\mathcal{E}^{(1)}, \ldots, \mathcal{E}^{(v)}\}) := \sum_{i=1}^{v} \frac{|\mathcal{E}^{(i)}|}{|\mathcal{E}|} \cdot \mathrm{Gini}(\mathcal{E}^{(i)}).$$

Decision trees are also used for estimating conditional class probabilities, i.e. for predicting $\mathrm{Pr}_D(y_i \mid x)$ for all examples $x \in \mathcal{X}$ and classes $y_i \in \mathcal{Y}$. A simple method is to use the class distributions of the training set at each leaf, and to assume that they reflect the true conditional distributions at those leaves. For fully grown trees these estimates are highly biased, however, because the splits are chosen as to minimize impurity, which systematically favors splits that lead to overly optimistic estimates.

A popular technique to reduce this effect is known under the name *Laplace estimate* (Cestnik, 1990): For any example subset, the counter of examples observed from each class is initialized with a value of 1, which reflects high uncertainty when computing estimates from small samples. For increasing sample sizes the impact of the constant offsets vanishes. This technique reduces overfitting in a heuristic manner, which does not allow to give probabilistic guarantees like confidence bounds for the true value of $\mathrm{Pr}_D(y_i \mid x)$.

An alternative is to utilize hold-out sets, which allows to compute *unbiased* estimates and confidence bounds for class distributions; an unbiased estimator has the property that the expected estimated value equals the true target value. As a disadvantage, hold-out sets reduce the number of examples available for training. Evaluating model performances and computing confidence bounds will be discussed in detail in chapter 3.

Probabilistic estimates can hardly be measured using the metrics defined so far. For this purpose several metrics have been proposed in the literature. The similarity of probabilistic predictions to regression tasks suggests to apply loss functions that are used for continuous target

labels. The most common of these loss functions is the mean squared error, averaging the individual losses $L_{SQ}(h(x), y) = (h(x) - y)^2$.

Definition 11 *For a density function* $D : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}+$*, a boolean target attribute* $\mathcal{Y} = \{0, 1\}$*, and a probabilistic (or "soft") classifier* $h : \mathcal{X} \to [0, 1]$ *that approximates conditional probabilities* $Pr_D(\mathcal{Y} = 1|x)$*, the* root mean squared error *(RMSE) of* h *is defined as*

$$RMSE_D(h) = \sqrt{\int_D (h(x) - y)^2 \, dx \, dy}.$$

It is well known that *Bayes' decision rule* is the best way to turn soft classifiers into "crisp" ones that take the form $h : \mathcal{X} \to \mathcal{Y}$. For estimated class probabilities $\widehat{Pr}$ this decision rule predicts the mode

$$\hat{y} := \arg\max_{y \in \mathcal{Y}} \widehat{Pr}(y \mid x),$$

which is the most likely class $\hat{y} \in \mathcal{Y}$ for each $x \in \mathcal{X}$.

Another family of metrics that is applicable to the task of selecting probabilistic classifiers measures the goodness of example rankings. The best known of these metrics is the *area under the ROC curve* (AUC), which is only discussed for boolean classification tasks. The origin of the name of this metric will become clear in subsection 2.5.2. The following definition of the AUC is based on the underlying distribution, and hence is appropriate when the task is to generalize the training data.

Definition 12 *For a soft classifier* $h : \mathcal{X} \to [0, 1]$ *and a pdf* $D : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^+$ *the* area under the ROC curve *metric is defined as the probability*

$$AUC_D(h) := Pr_{(x,y),(x',y') \sim D^2} \left[h(x) \geq h(x') \mid y = 1, y' = 0 \right]$$

that a randomly sampled positive example is ranked higher than a randomly sampled negative one.

For a given example set $\mathcal{E}$, the empirical AUC for this set $\mathcal{E}$ can be computed by ordering all examples by their estimated probabilities (or confidences) to be positive. For sets that are ordered in this fashion, the AUC can be shown to be proportional to the number of switches between neighboring examples, in the sense of the bubble sort algorithm, that are required to "repair" the ranking; for repaired rankings all positive examples are ranked higher than all negative examples. More precisely, let $\Lambda(h, \mathcal{E})$ denote the number of required switches for an example set $\mathcal{E}$ ordered according to the predictions made by h. Let further denote $\mathcal{E}_+$ the subset of positive examples and $\mathcal{E}_-$ the subset of negative ones. Then the AUC of h for $\mathcal{E}$ is

$$AUC(h, \mathcal{E}) := \frac{\Lambda(h, \mathcal{E})}{|\mathcal{E}_+| \cdot |\mathcal{E}_-|}.$$

As this definition illustrates, the AUC metric is invariant to monotone transformations of h.

2.4.2. Classification rules

Logical rules are well interpretable models, commonly used to formulate complete programs in languages like PROLOG (Sterling & Shapiro, 1994), and to represent background knowledge for a domain, if the reasoning process needs to be communicated to domain experts (Scholz, 2002b). This kind of background knowledge can be exploited by some Inductive Logic Programming approaches (Muggleton, 1995). A restriction of Horn logics allows for tractable inference and induction.

Definition 13 *A* classification rule *consists of an* antecedent A*, which is a conjunction of atoms over* $A_1, \ldots, A_k$*, and a* consequence C*, predicting a value for the target attribute. It is notated as* $A \rightarrow C$*. If the antecedent evaluates to true for an example, the rule is said to be* applicable *and the example is said to be* covered*. If the consequence also evaluates to true, the rule is said to be* correct*.*

The syntactical form of rules is of minor importance in this work. In numerical domains, atoms usually take the form $A_i \oplus \theta$, with A_i denoting an attribute, θ being a threshold from the corresponding domain, and $\oplus \in \{<, \leq, \geq, >\}$ being an operator that compares attribute values to thresholds. In boolean and nominal domains it is common to check for equality only, i.e. to use atoms of the form $A_i = \theta$.

The function $\mathtt{Ext}$ will sometimes be used for the sake of clarity in the context of rules, e.g., to point out that set operations do not refer to syntactical elements. $\mathtt{Ext}$ maps antecedents A and consequences C to their *extensions* $\mathtt{Ext}(A) \subseteq \mathcal{X}$ and $\mathtt{Ext}(C) \subseteq \mathcal{X}$, those subsets of the instance space for which the expressions evaluate to true.

For many applications, rules cannot be expected to match the data exactly. It is sufficient if they point out interesting regularities in the data, which requires to refer to the underlying pdf D. In this setting, antecedents and consequences are considered to be probabilistic events, e.g.,

$$\Pr_D[A] := \Pr_D[\mathtt{Ext}(A)] .$$

The intended semantic of a probabilistic rule $A \rightarrow C$ is to point out that the conditional probability $\Pr_D[C|A]$ is higher than the class prior $\Pr_D[C]$; in other terms, the events represented by antecedent and conclusion are correlated. Probabilistic rules are sometimes annotated with their corresponding conditional probabilities:

$$A \rightarrow C \;\; [0.8] \quad :\Leftrightarrow \quad \Pr_D[C \mid A] = 0.8$$

The usefulness of such rules, and hence the reasons to prefer one probabilistic rule over another, may depend on several task-dependent properties. The next paragraphs provide a brief introduction to rule evaluation metrics.

2.4.3. Functions for selecting rules

Performance metrics are functions that heuristically assign a utility score to each rule under consideration. Different formalizations of the notion of *rule interestingness* have been proposed in the literature, see e.g. (Silberschatz & Tuzhilin, 1996). Interestingness is interpreted as unexpectedness throughout this work. The following paragraphs discuss a few of the most important metrics for rule selection.

First of all, the notion of accuracy can be translated to classification rules $A \rightarrow C$ in boolean domains by making the assumption that a rule predicts any class C when it applies, and the opposite class $\overline{C}$, whenever it does not.

Definition 14 *The* accuracy of a rule $A \rightarrow C$ *is defined as*

$$\mathrm{ACC}(A \rightarrow C) \;\; := \;\; \Pr[A, C] + \Pr\left[\overline{A}, \overline{C}\right]$$

However, in prediction scenarios rules are generally not considered to make any prediction if they do not apply, but only for the subset $\mathtt{Ext}(A)$. The *precision* is a metric similar to accuracy, that only considers the subset $\mathtt{Ext}(A)$ which is covered by a rule.

Definition 15 *The* precision *of a rule reflects the conditional probability that it is correct, given that it is applicable:*

$$\text{PREC}(A \rightarrow C) := \Pr[C \mid A]$$

In contrast to predictive accuracy, misclassifications due to examples from class C that are not covered are not accounted for. However, when assuming that a rule predicts the negative class if it does not apply, accuracy is equivalent to the naturally weighted precisions of a rule for the subsets $\text{Ext}(A)$ and $\text{Ext}(\overline{A})$:

$$\begin{aligned} \text{ACC}(A \rightarrow C) &= \Pr[A, C] + \Pr\left[\overline{A}, \overline{C}\right] \\ &= \Pr[C \mid A] \cdot \Pr[A] + \Pr\left[\overline{C} \mid \overline{A}\right] \cdot \Pr\left[\overline{A}\right] \\ &= \Pr[A] \cdot \text{PREC}(A \rightarrow C) + \Pr\left[\overline{A}\right] \cdot \text{PREC}(\overline{A} \rightarrow \overline{C}) \end{aligned}$$

The notion of *confidence*, equivalent to precision, is common in the literature on mining frequent itemsets (Agrawal & Srikant, 1994). For classification rules, the precision may also be referred to as the *rule accuracy* (e.g., in (Lavrac et al., 1999)), suggesting that a final classifier consists of a disjunction of such rules. This confusing notion is avoided in this work.

A short-coming of the precision metric is that it does not take into account *class priors* $\Pr[C]$. This is an important information, however, to quantify the advantage of a rule over random guessing. The following metric captures a kind of information that is similar to precision, but overcomes this drawback. Its origins are rooted in the literature on frequent itemset mining (Brin et al., 1997). In supervised contexts it measures the difference in the target attribute's frequency for the subset covered by a rule, compared to the prior.

Definition 16 *For any rule* $A \rightarrow C$ *the* LIFT *is defined as*

$$\text{LIFT}(A \rightarrow C) := \frac{\Pr[A, C]}{\Pr[A]\Pr[C]} = \frac{\Pr[C \mid A]}{\Pr[C]} = \frac{\text{PREC}(A \rightarrow C)}{\Pr[C]}$$

The LIFT of a rule captures the value of "knowing" the prediction for estimating the probability of the target attribute:

- $\text{LIFT}(A \rightarrow C) = 1$ indicates that A and C are independent events.
- With $\text{LIFT}(A \rightarrow C) > 1$ the conditional probability of C given A increases,
- with $\text{LIFT}(A \rightarrow C) < 1$ it decreases.

The LIFT may be considered to be a version of PREC that has been normalized with respect to the class skew. It will show that, for selecting and combining rules, considering the LIFT is often more convenient and informative, in particular because even random guessing may yield a high PREC for skewed datasets.

A comparable measure, well-known from subgroup discovery (Klösgen, 1996), is the *bias*, which reflects the linear deviation of a rule's precision to the precision of random guessing.

Definition 17 *The* bias *of a rule* $A \rightarrow C$, $C \in \mathcal{Y}$ *under pdf* D *is defined as the difference between the conditional probability of* C *given* A *and the prior of class* C*:*

$$\text{BIAS}_D(A \rightarrow C) := \Pr_D[C \mid A] - \Pr_D[C] = \Pr_D[C] \cdot (\text{LIFT}_D(A \rightarrow C) - 1)$$

Rules that cover only a few examples are generally not relevant for data mining. In descriptive settings such rules are are often valid just by chance, hence misleading, which can formally be shown by significance tests. Consequently, such rules also tend to generalize poorly. The coverage is a useful criterion to avoid the selection of such rules.

Definition 18 *The* coverage (COV) *of a rule* $A \rightarrow C$ *is defined as the probability that it is applicable for an example sampled from the underlying pdf* D*:*

$$\mathrm{Cov}_D(A \rightarrow C) := \Pr_D[A]$$

The coverage of a rule is identical to its *support* defined in section 2.2, which is the term traditionally used in the context of frequent itemset and association rule mining (Agrawal & Srikant, 1994).

Besides taking care that the discovered rules all have a large coverage, the *significance of rules* can directly be considered as a rule selection criterion. This prevents us from reporting rules that bear a high risk of performing well just by chance. Rules not worth to be reported perform about as well as random guessing. This means that the conditional probability $\Pr[C \mid A]$ of the predicted class C, given that the rule applies, is approximately identical to the class prior $\Pr[C]$. Please note that rules with a lower conditional probability can easily be turned into interesting rules by just inverting their predictions, e.g. from C to $\overline{C}$.

One popular example of a significance test is the *binomial test*. It allows to compute the probability of the null hypothesis $\Pr[C \mid A] = \Pr[C]$ that a rule $A \rightarrow C$ under consideration does not contribute any information. For i.i.d. samples the probability $\Pr[C \mid A]$ is fixed. We consider a subsample that contains m observations from $\mathrm{Ext}(A)$. Let $X_1, \ldots, X_m$ denote Bernoulli random variables, with $X_i = 1$ if the class of example i is C, and 0 otherwise. Let further $Y := \frac{1}{m}\sum_{i=1}^{m} X_i$ denote the mean of these variables. According to the central limit theorem this distribution (binomial) is approximately normal for large samples. If the null hypothesis is correct, then we have

$$\mathbb{E}(Y) = \frac{1}{m}\sum_{i=1}^{m}\mathbb{E}(X_i) = \frac{1}{m}\left(m \cdot \Pr(C \mid A)\right) = \Pr(C),$$

and a standard deviation of

$$\sigma(Y) = \sqrt{\sum_{i}^{m}\sigma^2(X_i/m)} = \sqrt{\frac{\Pr(C)\cdot(1-\Pr(C))}{m}}.$$

Normalizing Y to mean 0 and standard deviation of 1 for the assumed values $\mathbb{E}(Y)$ and $\sigma(Y)$ of the null hypothesis above yields the *binomial test function*:

$$f(Y, \Pr(C), m) := \frac{Y - \Pr(C)}{\sqrt{\Pr(C)\cdot(1-\Pr(C))/m}} \tag{2.2}$$

Referring to the normal approximation, this gives us the standard z-score (inverse standard normal distribution) of the null hypothesis. Hence, we may compute (e.g., table look-up) the quantile $z(\alpha)$ of the standard normal distribution for any significance level α and accept rules only if e.g. $f(Y, \Pr(C), m) < z(\alpha)$, which corresponds to a one-sided test. Alternatively, the values of f can be used to define a preference order over the set of rule candidates. This prefers significant rules over less significant ones, while still leaving the option of considering further criteria in parallel.

For any significance test it is important to take into account the number of evaluated rule candidates, i.e. the cardinality of the hypothesis space $\mathcal{H}$, because otherwise even reasonably low significance levels may not help to prevent any of the poor rules from passing. This issue is discussed in more detail in subsection 3.3.2.

A commonly used class of utility functions in the scope of subgroup discovery is captured by the following definition.

Definition 19 *For a given parameter* α *and underlying probability density function* D*, the utility (or quality)* $q_D^{(\alpha)}$ *of a rule* $r \in \mathcal{H}$ *is defined as*

$$q_D^{(\alpha)}(r) := \text{Cov}_D(r)^\alpha \cdot \text{Bias}_D(r).$$

The parameter α allows for a data and task dependent trade-off between coverage and bias. Definition 19 covers various relevant metrics. With $\alpha = 0.5$ we reach at a metric factor-equivalent to the binomial test function: In eqn. (2.2) the quantity m denotes the number of examples in a subset $\text{Ext}(A)$, while the random variable Y corresponds to the fraction of examples of class C in $\text{Ext}(A)$. For a considered rule $r : A \rightarrow C$ this allows to rewrite the binomial test function in terms of the previous definitions:

$$\begin{aligned} f(Y, \Pr(C), m) &= \frac{Y - \Pr(C)}{\sqrt{\Pr(C) \cdot (1 - \Pr(C))/m}} = \frac{\Pr(C|A) - \Pr(C)}{\sqrt{\Pr(C) \cdot (1 - \Pr(C))/(\text{Cov}(r) \cdot |\mathcal{E}|)}} \\ &= \sqrt{\text{Cov}(r)} \cdot (\Pr(C \mid A) - \Pr(C)) \cdot \underbrace{\sqrt{\frac{|\mathcal{E}|}{\Pr(C) \cdot (1 - \Pr(C))}}}_{=:c \text{ (constant)}} \\ &= \sqrt{\text{Cov}(r)} \cdot \text{Bias}(r) \cdot c = c \cdot q^{(0.5)}(r) \end{aligned}$$

The total number of examples $|\mathcal{E}|$ and the class priors $\Pr(C)$ and $\Pr(\overline{C})$ are constants, so they can be ignored for rule selection problems. For simplicity, we will prefer $q^{(0.5)}$ over the binomial test function in the remainder of this work.

Choosing $\alpha = 2$ in definition 19, we reach at a function commonly used to put higher emphasis on the bias. The choice of $\alpha = 1$ yields a popular utility function for subgroup discovery that will be used at several occasions throughout this work:

Definition 20 *For an underlying pdf* D *the* weighted relative accuracy *of a rule* $r \in \mathcal{H}$ *is*

$$\text{WRAcc}_D(r) := \text{Cov}_D(r) \cdot \text{Bias}_D(r).$$

WRAcc can be considered a default metric for subgroup discovery, e.g. applied in the early system Explora (Klösgen, 1996) and the algorithm Midos (Wrobel, 1997).

Finally, it should be mentioned that the usage of a variety of metrics for different purposes has a long tradition in the rule learning community. For brevity, several metrics were omitted; for a broader overview please refer to Fürnkranz (1999).

2.5. ROC analysis

As discussed in the last section, the performances of classifiers can be measured in terms of different metrics. The receiver operator characteristics (ROC) are a useful tool for analyzing the differences between these metrics, and for visualizing the performances of different classifiers. The framework of ROC analysis has its roots in signal detection theory, but has recently gained much attention in the machine learning community, leading e.g. to several focused workshops, like (Hernández-Orallo et al., 2004; Ferri et al., 2005). The following paragraphs provide only the most relevant aspects for the remainder of this work. Fawcett (2003) provides a more extensive introduction to ROC graphs for data mining.

	$\mathbf{C}$	$\overline{\mathbf{C}}$
$\mathbf{A}$:	TP	FP
$\overline{\mathbf{A}}$:	FN	TN
$\sum$:	**P**	**N**

	$\mathbf{C}$	$\overline{\mathbf{C}}$	$\sum$	PREC
$\mathbf{A}$:	63	11	74	$\approx 85\%$
$\overline{\mathbf{A}}$:	32	121	153	$\approx 79\%$
$\sum$:	**95**	**132**	**227**	

Figure 2.1.: Columns of contingency matrices are defined with respect to the true labels, rows with respect to the predictions. The left table illustrates the defined abbreviations, the right an example, where PREC refers to the precisions of $A \to C$ and $\overline{A} \to \overline{C}$.

2.5.1. Visualizing evaluation metrics and classifier performances

Traditional ROC analysis in machine learning addresses boolean classification problems. Models make predictions for the label of each element of an example set $\mathcal{E}$. In the case of *crisp* classifiers the range of predictions is equal to the set of labels $\mathcal{Y}$, here assumed to contain the two classes y_+ (pos.) and y_- (neg.). As in PAC learning theory, the extension of a classifier h is defined as $\mathtt{Ext}(h) := \{h(e) = y_+ \mid e \in \mathcal{E}\}$, the subset of example set $\mathcal{E}$ for which the classifier predicts the positive class. Regardless of whether the classifier is a rule or any other kind of model, the set $\mathtt{Ext}(h)$ is said to be *covered*.

The absolute number of covered examples that are in fact positive (*true positives*) is denoted as

$$\mathsf{TP} := |\{y = y_+ \mid (x, y) \in \mathtt{Ext}(h)\}|.$$

Accordingly, the negative examples that were covered "by mistake" are referred to as the *false positives*, so $\mathsf{FP} := |h| - \mathsf{TP}$. All examples in $\overline{h} = \mathcal{E} \setminus h$ are *not covered*, and, for most model classes, may be assumed to be predicted negative. In this case, the number of those examples that are in fact negative is denoted as TN (for *true negatives*), the number of uncovered positives defines the *false negatives*, FN. Finally, the absolute numbers of positives in $\mathcal{E}$ are denoted as P, the absolute numbers of negatives as N. Table 2.1 summarizes these quantities in a so called *contingency matrix*.

For ROC analysis the true and false positive *rates* are of special interest. These quantities allow to establish a view on the performances of classifiers that abstracts from the class skew P/N, simply by normalizing TP with respect to P and FP with respect to N:

Definition 21 *For a boolean crisp classifier that predicts (exactly) all examples from a subset* $\mathtt{Ext}(h) \subseteq \mathcal{E}$ *of a given example set* $\mathcal{E}$ *as positive, and that has* TP *true positives and* FP *false positives, the* true positive rate TPr *and the* false positive rate FPr *are defined as*

$$\mathsf{TPr} := \mathsf{TP}/\mathsf{P} = \frac{|\{y = y_+ \mid (x, y) \in \mathtt{Ext}(h)\}|}{|\mathtt{Ext}(h)|} \text{ and}$$

$$\mathsf{FPr} := \mathsf{FP}/\mathsf{N} = \frac{|\{y = y_- \mid (x, y) \in \mathtt{Ext}(h)\}|}{|\mathtt{Ext}(h)|},$$

respectively. The true positive rate of a model is also referred to as its recall.

ROC plots provide two-dimensional visualizations of classifier performances. Each point in a ROC plot refers to a false positive rate (x-axis) and a true positive rate (y-axis), and thereby reflects the fractions of correctly and incorrectly covered examples. Figure 2.2 shows a ROC plot with three points, each of which represents a different classifier performance. The diagonal with slope 1 in ROC diagrams represents the expected classifier performances that can be achieved by plain random guessing. The two $(\mathsf{FP}, \mathsf{TP})$ points $(0, 0)$ and $(1, 1)$ represent the two default

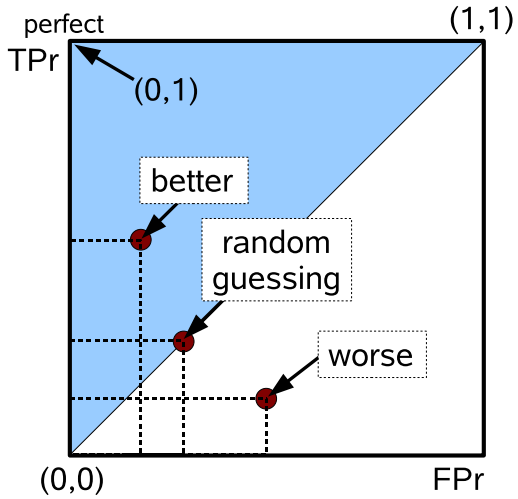


Figure 2.2.: Basic ROC plot properties: The diagonal corresponds to the performance of random guessing, points in the upper left triangle perform better. The point $(0, 1)$ represents perfect predictions.

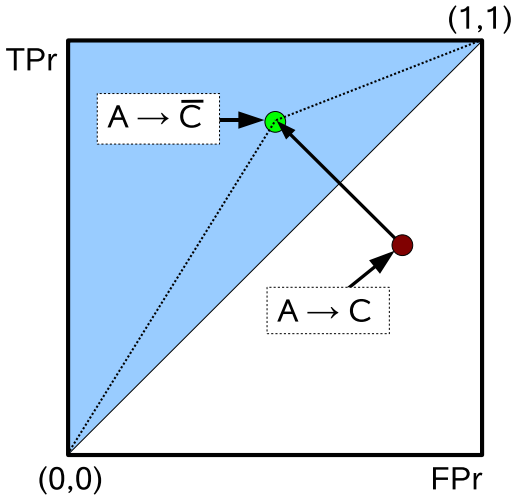


Figure 2.3.: Inverting predictions of models in the lower right triangle mirrors its performance along the diagonal. The green point refers to a useful model; it can be interpolated with $(0, 0)$ and $(1, 1)$.

hypotheses, the former predicting all examples as negative, the latter as positive. Any point in between represents a probabilistic combination of these default hypotheses. This does not require to inspect the data at all. More generally, any point lying within the convex hull of classifier performances anywhere in ROC space can be reached by (probabilistic) interpolation of classifiers.

Any classifier showing a performance in the upper left triangle of a ROC plot performs better than random guessing. Optimal classifiers predict always correctly, and hence have a true positive rate of 1 and a false positive rate of 0. This corresponds to the point $(0, 1)$ at the upper left edge. Since such classifiers cover exactly the positives, they do, in turn, cover none of the negatives. The same principle applies to less well performing classifiers. This illustrates why the complete contingency matrix is represented by ROC plots, although only the performance on the covered subset $\mathtt{Ext}(h)$ is visualized explicitly. Points lying in the triangle below the diagonal perform worse than random guessing. However, if the performance of such a classifier is known, the predicted label can simply be flipped. The geometric effect in ROC plots is that the point is mirrored at the diagonal, so that the performance of the corresponding rule is on the "good" (upper left) side afterwards. This is illustrated in figure 2.3. Keeping this in mind, the worst case are points lying on (or close to) the diagonal. The corresponding classifiers do not provide much information on how the class labels are distributed in the example set.

As discussed in section 2.4, how to decide which classifiers to select is an important question in machine learning. For each evaluation metric the corresponding preference ordering can well be illustrated in terms of ROC plots. We will start with the classification error. Decomposing the absolute number of errors into misclassified positives and misclassified negatives, the total error ϵ_{abs} (number of misclassified examples) is

$$\epsilon_{\text{abs}} = (1 - \text{TPr}) \cdot \text{P} + \text{FPr} \cdot \text{N}. \tag{2.3}$$

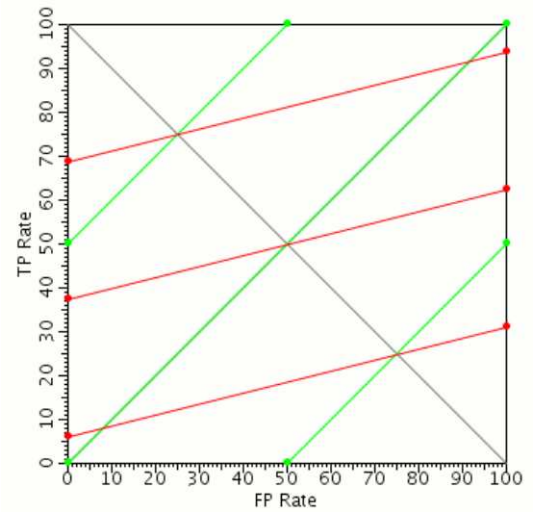


Figure 2.4.: The isometrics of ACC without class skew (green) and with 80% positives (red). In both cases the slope of isometrics is N/P. Isometrics are shown for ACC of 25%, 50%, and 75%.

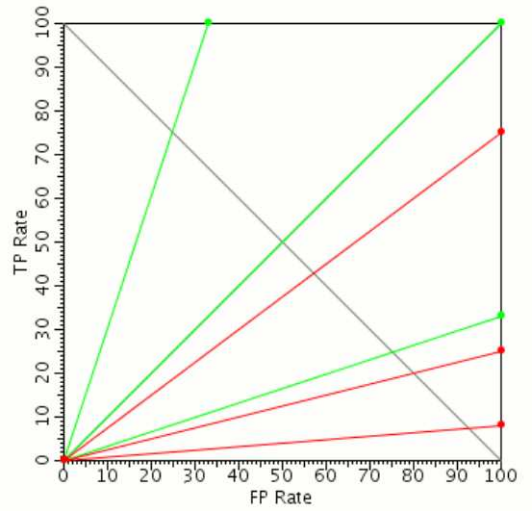


Figure 2.5.: Isometrics for PREC of 25%, 50%, and 75%, without skew (green) and with 80% positives (red). Because of the high skew, the red line performances are worse than random guessing.

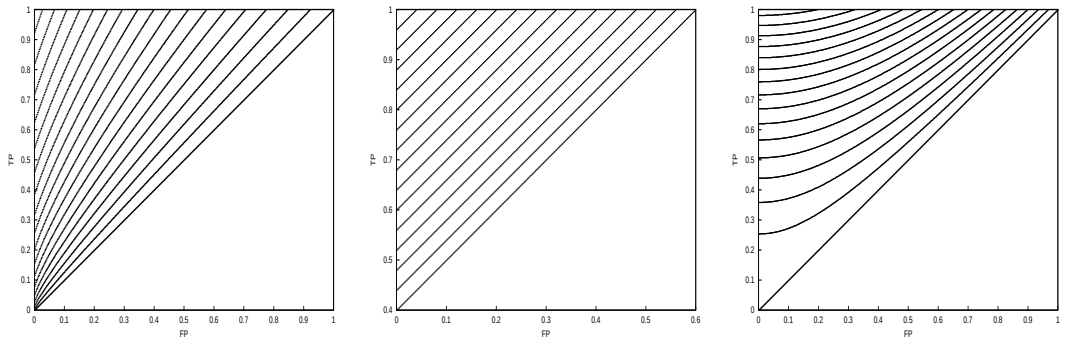
Rearranging terms, it directly follows that in this case two classifiers with true and false positive rates (FPr_1, TPr_1) and (FPr_2, TPr_2) perform equally well, if and only if

$$\frac{TPr_2 - TPr_1}{FPr_2 - FPr_1} = \frac{N}{P} =: c. \tag{2.4}$$

For any known class skew c, which corresponds to the ratio $Pr(y_-)/Pr(y_+)$, this defines linear *isometrics* with a slope of c in ROC space. An isometric is a line in a ROC diagram that consist of performances for which a considered metric yields the same score. Hence, all classifiers lying on the same isometric line are exchangeable when optimizing with respect to this metric. In this setting, the goal of classifier induction, the maximization of ACC, translates into the identification of a model that lies on an isometric line with slope N/P that intercepts the y-axis as closely as possible to the point $(0, 1)$.

Based on visualizing ROC isometrics, Fürnkranz and Flach (2003) analyzed and compared several important classifier evaluation metrics. Figure 2.4[4] shows the isometrics for accuracies of 25%, 50%, and 75%. For unskewed data these isometrics have a slope of $N/P = 1$, and are hence parallel to the diagonal that connects the point $(0, 0)$ to $(1, 1)$. For 80% positives and 20% negatives the isometrics are much flatter parallel lines. The precision metric does not consider the coverages of rules, but only the ratio of covered positives to covered negatives. This ratio does not change along any of the lines crossing the origin of a ROC plot, which explains the different geometry of the isometrics shown in Fig. 2.5 compared to Fig. 2.4. Again, the isometrics are shown for values of 25%, 50%, and 75% for unskewed data, and for 80% positives. It is interesting to note that even a precision of 75% leads to an isometric line in the lower right

[4]The plot was produced using the ROCOn software:
`http://www.cs.bris.ac.uk/Research/MachineLearning/rocon/`



(a) $\alpha = 1/2$, binomial test (b) $\alpha = 1$, WRAcc (c) $\alpha = 2$

Figure 2.6.: ROC isometrics of $Q^{(\alpha)}$ utility functions for different values of α.

triangle, because at the given skew level the default classifier that always predicts positive labels results in a precision of 80%. However, as discussed before, any model with a precision of e.g., 25% can be turned into a well performing model by just inverting its predictions. Figures 2.6(a)-2.6(c) show the isometrics in ROC space for three members of the family of utility functions $Q_D^{(\alpha)}(r)$ used for subgroup discovery (Def. 19). Please recall, that $\alpha = .5$ is factor equivalent to the binomial test function, which hence has the same geometry of ROC isometrics. A comparison between the plots for $\alpha = .5$ and 2 illustrates the role of the coverage, which increases along each of those lines with an increasing distance to the origin. The choice of $\alpha = 1$ leads to the weighted relative accuracy, which has the interesting property that its isometrics have the same geometry, parallel lines with a slope of 1, for all class skews. This indicates invariance against skewed class distributions. As shown in figure 2.4, this is different for the accuracy metric. Flach (2003) systematically studies the behavior of different metrics for varying class skews in a three-dimensional variant of ROC spaces, where the third dimension represents the degree of skew. For unskewed class distributions, i.e. $P = N$, WRAcc and Acc rank models identically. There is a similar connection between Lift and Prec; the former is a skew-invariant variant of the latter. Stronger connections between WRAcc and Acc than pointed out here will be derived in chapter 3.

Finally it is worth to note, that the isometrics of entropy are very similar to the isometrics of predictive accuracy, which implies that in most cases both metrics will suggest to select the same model. A more detailed discussion, including implications of the results for separate-and-conquer rule induction can be found in (Fürnkranz & Flach, 2005).

2.5.2. Skews in class proportions and varying misclassification costs

In the scope of classifier induction, the idea of sampling identically and independently from a single distribution during training and at application time is often unrealistic. Depending on the kind and degree of violations, viable solutions range from ignoring the changing nature of the data generating processes to changing the model class and learning strategy. As pointed out by Provost and Fawcett (1997), one of the advantages of ROC analysis is that it allows to decouple classifier performances from specific class distributions and cost assumptions. In particular, ROC plots have the appealing property of being invariant to skewed class proportions, because true positive and true negative rates are normalized in this respect. In practice, a sceptical position regarding this invariance is advisable, however; for some domains a changing class skew may indicate additional effects, which may change the performance of a classifier drastically. This

issue has recently been discussed controversially by Webb and Ting (2005) and Fawcett and Flach (2005).

The following paragraphs identify different ways in which the i.i.d. assumption might be violated:

Skewed class distributions A moderate violation of the i.i.d. assumptions underlying the presented learning framework is given, if the concepts underlying the data remain, but the class priors may vary over time. In particular, the case in which the class priors at application time are unknown during training is relevant for practical applications. Tackling this case robustifies models against minor changes in the underlying distribution. The *fraction* of spam emails when applying a spam filter is typically unknown during the off-line training phase, for example, and is likely to increase over time. A desirable feature in such a setting is that selected models work well at all reasonable skew levels, no matter whether e.g. 5% or 99.9% of incoming emails are spam.

Cost-sensitive learning Closely related to changing priors – from a technical point of view – is the fact that (i) misclassification costs are often different for positive and for negative examples, (ii) can often hardly be stated exactly, and (iii) may also change over time. Discarding a non-spam email may be much worse than not to detect a large fraction of the incoming spam, for example. This implicitly means that misclassification *costs* are asymmetric in this application domain, and they are obviously of a subjective nature. In many real-world domains misclassification costs change over time. For a spam filter this happens, for example, if a user waits for an exceptionally important email, or if she decides to forward all incoming emails to a pager for a while.

Arbitrarily changing distributions Even for given sample independence the underlying distribution may be known not to be stationary, but to change over time. Models used to detect fraud, network intrusions, or spam emails will generally make use of older example sets. Such sets cannot be expected to reflect the current distributions well, due to the quickly changing nature of these domains. Either the underlying target concepts change over time, or the distribution gives higher weight to poorly modeled subsets. An example of the former kind of change are users suddenly considering the posts on specific mailing lists as spam, an example of the latter are spammers adapting their emails maliciously to the detection capabilities of available filters. Learning from non-stationary target distributions is also referred to as *learning under concept drift* and will be addressed in chapter 6.

The issue of different class skews underlying a data set has already been raised in subsection 2.5.1. Eqn. (2.4) captures the connection between this skew N/P and the slope of the corresponding ROC isometrics for accuracy. This can be considered as a specific case of cost-sensitive learning, where the misclassification of each example causes the same costs (or losses), e.g. of 1. In more general settings these costs may depend on the true label of the misclassified example, leading to *asymmetric loss-functions*.

Let $L_c : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}^+$ denote a loss function depending on the target attribute $\mathcal{Y}$, with $L_c(y, y) = 0$ for all $y \in \mathcal{Y}$, and costs $L_c(y, y') := l(y) \geq 0$ for true label y and predicted label y', where $l(y)$ is a constant that depends on the true class, only. In this case eqn. (2.3) for computing the total error of a classifier on a given data set can be generalized to capture the total cost. Let TP denote the true positives and FP the false positives of a classifier $h : \mathcal{X} \to \mathcal{Y}$. Then

its total misclassification costs on an example set $\mathcal{E}$ are

$$\sum_{(x,y)\in\mathcal{E}} L_c(y, h(x)) = \underbrace{(1 - TPr) \cdot P \cdot l_{y_+}}_{\text{Misclassified positives}} + \underbrace{FPr \cdot N \cdot l_{y_-}}_{\text{Misclassified negatives}}$$

It is easily seen that the slope of ROC isometrics N/P derived in eqn. (2.4) can be adapted to the asymmetric loss function in a straightforward manner: For misclassification costs l_{y_+} and l_{y_-} two classifiers with true and false positive rates (FPr_1, TPr_1) and (FPr_2, TPr_2) perform equally well, if and only if

$$\frac{TPr_2 - TPr_1}{FPr_2 - FPr_1} = \frac{N \cdot l_{y_-}}{P \cdot l_{y_+}} =: c. \tag{2.5}$$

This implies that the class skew and asymmetric misclassification costs can be combined to a *single kind of skew* c underlying the data, defining the slope of the linear isometrics, which is handled as a parameter of the learning task of classifier induction. As discussed before, for a known slope the best classifier has a performance which lies on an isometric line as close as possible to the upper left point $(0, 1)$. The practically relevant case of *varying* slopes can be understood and addressed by ROC analysis as well.

First of all, it is interesting to investigate the behavior of boolean *soft classifiers* that associate confidence scores to their predictions. For boolean $\mathcal{Y}$, the (crisp) predictions can be replaced by a single confidence score per example, e.g. the confidence that the example is positive. When normalizing confidence scores to the interval of $[0, 1]$, soft classifiers are functions of the form $h : \mathcal{X} \to [0, 1]$. A predicted value of 1 (0) corresponds to a most confident prediction of class y_+ (y_-). Most classifiers in use are able to yield confidence-rated predictions, which allows to address a broader variety of prediction problems, like finding and ranking the n documents that are probably most interesting to a user. Any soft classifier can be turned into a crisp one by applying a threshold function; such functions map all confidence scores below a fixed threshold θ to a negative prediction, and all others to a positive one.

ROC analysis supports the visualization of soft classifiers and the selection of a threshold with corresponding minimal misclassification costs. Ranking the examples by their confidence scores and increasing the threshold as to cover one more example at a time results in $|\mathcal{E}| + 1$ different classier performances (FPr, TPr). A plotting procedure may start with a threshold that covers no examples and hence has a performance of $(0, 0)$. Each time a positive example is added to the set of covered examples the corresponding point has the same FPr and a higher TPr score, which shows as a (small) vertical move in ROC space. Analogously, adding a negative example leads to a step to the right. If sets of p positive and n examples share the same confidence score, then these examples are aggregated in ROC space. This results in a (usually non-axis parallel) line with slope n/p.

In practice the ideal slope for classifier and threshold selection is often unknown in advance, e.g., because misclassification costs and class skews may vary. In this case it is desirable to have low expected costs for various slopes. Figure 2.7 shows an example of a soft classifier visualized in ROC space. The plot allows to identify an optimal threshold for any slope c defined by a given class skew N/P and asymmetric loss function L_c for incorporating misclassification costs. Optimality is defined in terms of the expected overall misclassification costs. Assuming (for now) that the slope c was known at each point in time, but that it changed continuously, we could always pick an optimal threshold for each classification by choosing the one that corresponds to a point lying on the best possible isometric line with slope c. The depicted slopes of 0.5 and 2 could have been caused by different combinations of cost ratios and class proportions. The

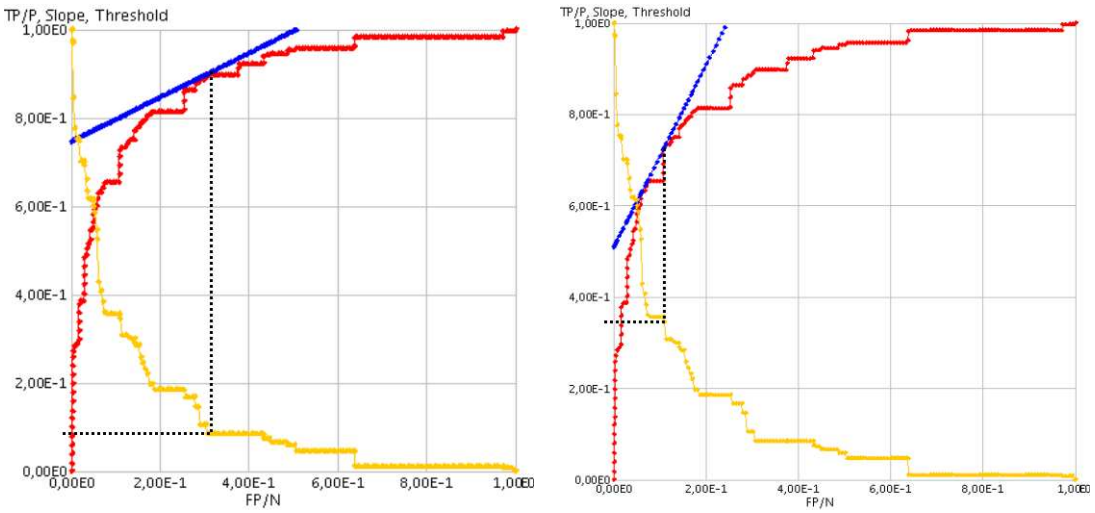


Figure 2.7.: The ROC plot shows the performances of a decision tree soft classifier (red) for different thresholds (yellow). Thresholds are plotted so that they correspond to the ROC curve for each fixed FPr. In the left plot, the assumed data skew leads to a slope of 0.5. The optimal threshold of about 0.1 can be found by intersecting the red with the blue line and projecting this point to the yellow curve. No line closer to $(0, 1)$ and parallel to the plotted blue line has any intersection with the red curve. In the right plot, the slope is 2. The best possible isometric line intersects with the red ROC curve for a threshold of about 0.35.

former slope could e.g. be the result of having twice as many negatives than positives at equal misclassification costs. The latter may be the result of having a cost ratio $l_{y_+} : l_{y_-}$ of $4 : 1$ for the same class skew.

One noticeable property is that only thresholds with points on the edge of the convex hull of the ROC plot need to be considered, because the performance for every other threshold will be dominated by one lying on this edge for *any* slope. As discussed before, the points on the edge of the convex hull can be interpolated by randomly switching between the corresponding crisp classifiers, which allows to reach any point lying on the edge of the hull. How to compute (and visualize) the variance of such curves in a meaningful and reliable way is still an active field of research, see e.g. (Mackassy et al., 2005).

If a classifier needs to perform well for different class skews and costs, then the area under the ROC curve (AUC) provides a good scalar measure for classifier selection. As the name suggests, the AUC is proportional to the area under the curve of soft classifier performances in ROC space; in figure 2.7, these are the areas under the red curves. The plots illustrate why selecting a soft classifier maximizing this area will perform well on average for changing slopes, and why such a classifier will often dominate most other classifiers for any slope.

The convex hull in ROC space can also be utilized for selecting a single best soft classifier and threshold taken from a set of candidates, in order to minimize the expected misclassification costs under varying slopes (Provost & Fawcett, 2001). The first step is to plot the performances of all classifiers under consideration for all thresholds in a single ROC diagram and to compute the convex hull of all those points. All classifiers that do not contribute a single point to this hull may be ignored. The others are stored, and whenever a point in the ROC space turns out to be optimal for a given slope at hand, the corresponding classifier and the threshold related to that point are used for making predictions. For this kind of application ROC spaces can easily

be transformed into so-called cost curves, depicting the expected misclassification costs for each slope (Drummond & Holte, 2006). Please note, that such techniques still select a *single* classifier and threshold for each slope, which is inherently different from ensemble methods that *combine* base model predictions.

2.6. Combining model predictions

In the last section it was discussed how to select a single classifier from a collection of models under varying class skews and/or misclassification costs. Given a variety of different models for a classification task at hand, it is intuitively not necessarily the best choice to rely on the predictions of only one of these classifiers, even if it promises to have the best performance in ROC space. There are many popular approaches that combine the predictions of several base classifiers based on voting. The most simple case of uniform voting is discussed in subsection 2.6.1. As an alternative to voting, subsection 2.6.2 illustrates how to combine arbitrarily many base classifier predictions based on the NAÏVEBAYES assumption of conditional independence. To overcome this unrealistic assumption, a generalized approach based on logistic regression is finally discussed in subsection 2.6.3.

2.6.1. Majority Voting

Many machine learning algorithms are known to produce classifiers that are sensitive to minor changes in the training set. One example are decision trees. The syntactical structure, but also the predictions of specific trees produced by algorithms like CART (Breiman et al., 1984) or C4.5 (Quinlan, 1993) will usually change remarkably when trained on different subsets of the training data. Repeatedly applying a learner to subsets of a training set is also referred to as *bootstrapping*.

In more technical terms, the variance of commonly applied decision tree learners is high, which is an unpleasant effect for practical applications, because the predictions for individual examples as well as the overall generalization performances of the resulting classifiers are not reliable (or "robust"). A simple ensemble strategy that helps to reduce the variance, and that often even leads to an increased accuracy of the resulting model is to combine the models trained from bootstraps of the original training sets by means of voting. Breiman (1996) proposed and analyzed the so-called *bagging* approach: Each model is trained on another bootstrap sample, i.e. on another uniform subsample of constant size of the training set.

If $h_t(x) \in \mathcal{Y}$ denotes the prediction of the classifier that was trained in iteration t, and if k classifiers are trained, then the final prediction $\hat{y}$ of the ensemble is

$$\hat{y} := \operatorname*{argmax}_{y \in \mathcal{Y}} \left(\sum_{t=1}^{k} I[h_t(x) = y] \right), \qquad \text{or} \qquad \hat{y} := \operatorname{sign} \left(\sum_{t=1}^{k} h_t(x) \right) \tag{2.6}$$

for $\mathcal{Y} = \{-1, +1\}$, respectively. The function $I[\cdot]$ above refers to the indicator function again; it is 1, if the argument evaluates to true, and 0, otherwise. If k is chosen as an odd number, then the result is unique for boolean $\mathcal{Y}$. The ensemble members can be considered to *vote* on the final prediction, with each model having the same impact. As mentioned above, the same approach applies to multi-class problems, but it can even be extended to cover regression. In this case, all predictions $h_t(x) \in \mathbb{R}$ are simply averaged.

The main disadvantage of bagging is that only a subset of the available data is used for training each classifier. Bootstraps are often generated by sampling with replacement, but even if the

samples contain the same number $|\mathcal{E}|$ of examples as the original datasets (due to containing duplicates) each example will be contained in the bootstrap sample with a probability of

$$1 - \left(1 - \frac{1}{|\mathcal{E}|}\right)^{|\mathcal{E}|} \approx 1 - \exp(-1) \approx 0.632.$$

Hence, the number of different instances used for training will still be significantly lower than $|\mathcal{E}|$. For unstable learners, i.e. learners yielding quite different classifiers in each iteration, this is compensated well by the procedure, however; the generalization performance improves while the variance decreases. As summarized by Breiman (1996) and confirmed by other researches (e.g., Grandvalet (2004)), bagging unstable classifiers usually improves them, while bagging stable classifiers might not be a good idea.

The notion of classifier stability is similar to the notion of *ensemble diversity*, which has recently gained much attention in the machine learning community. When combining an arbitrary number of classifiers, which are not necessarily the result of a single learner, then any combination of accurate classifiers with a high disagreement rate will produce an accurate ensemble classifier. Hence, reducing the overlap between different rules or the agreement of classifiers in general, has been recognized as one of the most important criteria in ensemble construction. Cunningham and Carney (2000) summarize corresponding results for regression and confidence-rated classification based on the root mean squared error, which allow to constrain the error rate of final ensemble classifiers. Based on these results they suggest different notions to measure diversity (or ambiguity) for plain classification problems. The entropy of predictions is one possible measure of ensemble diversity. The soft predictions of k classifiers can be stored in a vector of $k \cdot |\mathcal{Y}|$ estimates per example, if $|\mathcal{Y}|$ denotes the number of classes. If $p_{t,j}(e)$ denotes the estimate of classifier t for class j for an example e, then an entropy-based definition of the diversity for example e is

$$H(e) := \sum_{j=1}^{|\mathcal{Y}|} \left(\frac{1}{k} \sum_{t=1}^{k} p_{t,j}(e) \right) \cdot \left(\sum_{t=1}^{k} -\log p_{t,j}(e) \right),$$

where the factor in the left brackets reflects the prediction for class j averaged over all classifiers, and the logarithmic term represents the encoding length of each individual prediction. The entropy for a training set $\mathcal{E}$ is simply the average of all example entropies, so

$$H(\mathcal{E}) := |\mathcal{E}|^{-1} \sum_{e \in \mathcal{E}} H(e).$$

Please note, that this diversity measure does not depend on the true labels. In later work Tsymbal et al. (2003) studied the behavior of several different diversity measures empirically for ensemble construction via feature subset selection. Several other approaches to maximize diversity of ensemble classifiers have recently been studied, e.g. constructing artificial examples to enforce diversity of constructed ensembles (Melville & Mooney, 2003). Zhang et al. (2006) take a semi-definite programming approach to prune existing classifier ensembles based on ensemble diversity and individual classifier accuracies. They state that in their experiments the common diversity measures performed about equally well. An analysis by Tang et al. (2006) shows that – assuming a fixed weighted base classifier accuracy – maximizing diversity leads to the largest margin in theory, which has been shown to correspond to low generalization error rates.

Besides maximizing diversity, assigning an appropriate *weight* to each ensemble member is important for improving the accuracy. Weighted voting is a straightforward extension of uniform

voting. Each classifier has a different impact on the final prediction in this case. If classifier t has a weight of w_t, then eqn. (2.6) in the multi-class case translates into

$$\hat{y} := \operatorname*{argmax}_{y \in \mathcal{Y}} \left(\sum_{t=1}^{k} w_t I[h_t(x) = y] \right), \qquad \text{or into} \qquad \hat{y} := \operatorname{sign} \left(\sum_{t=1}^{k} w_t h_t(x) \right)$$

for $\mathcal{Y} = \{-1, +1\}$, respectively. Weighted voting is often used in combination with altering example weights during training. The most popular corresponding learning technique, referred to as *boosting* (Freund & Schapire, 1997), is discussed in detail in chapter 5. This technique iteratively induces a set of base classifiers, changing example weights after each learning step. Examples that are misclassified by the ensemble of previously selected classifiers receive a higher weight in the next iteration, so that subsequent learning steps prefer models that help to correct remaining errors of the ensemble.

2.6.2. A NAÏVEBAYES-like combination of predictions

In the last paragraphs the simple concept of voting has been discussed to combine the predictions of a set of base classifiers. In this subsection, each classifier can rather be thought of as a constructed feature. For example, each rule $A \rightarrow C$ in a two-class prediction problem either applies (antecedent A evaluates to true), or it does not. This can be represented as a boolean feature, taking the values of 1 and 0, respectively. To combine classifier predictions we can use an arbitrary classifier on top of these constructed features.

The following ideas are based on Bayes' theorem, which is used to compute the conditional distribution of the label. The underlying assumption of NAÏVEBAYES (John & Langley, 1995) is that all attributes are conditionally independent given the class. Hence, the approach presented next does not take interactions between features into account, but just exploits the correlations between each feature and the label separately. NAÏVEBAYES classifiers work surprisingly well in practice, often even if the underlying independence assumption is known to be violated.

In our setting, the goal is to predict for each $x \in \mathcal{X}$ conditional class probabilities $\Pr[y \mid x]$, for any class $y \in \mathcal{Y}$, based on the predictions of a set of k classifiers.

Let $\vec{h}(x) := (h_1(x), \ldots, h_k(x)) \in \mathcal{Y}^k$ denote the predictions of models $\{h_t \mid 1 \leq t \leq k\}$ in the form of a vector. Then, for a given example $x \in \mathcal{X}$ and corresponding vector $\vec{y} = (y^{(1)}, \ldots, y^{(k)})$ of predictions, the NAÏVEBAYES classifier estimates the conditional probability of any class $y \in \mathcal{Y}$ given x as

$$\begin{aligned}
\Pr\left[y \mid \vec{h}(x) = \vec{y}\right] &= \frac{\Pr[y]}{\Pr\left[\vec{h}(x) = \vec{y}\right]} \cdot \Pr\left[(h_1(x), \ldots h_k(x)) = \vec{y} \mid y\right] \\
&\approx \frac{\Pr[y]}{\Pr\left[\vec{h}(x) = \vec{y}\right]} \cdot \prod_{1 \leq t \leq k} \Pr\left[h_t(x) = y^{(t)} \mid y\right] \qquad (2.7) \\
&= \frac{\Pr[y]}{\Pr\left[\vec{h}(x) = \vec{y}\right]} \prod_{1 \leq t \leq k} \frac{\Pr\left[y \mid h_t(x) = y^{(t)}\right]}{\Pr[y]} \prod_{1 \leq t \leq k} \Pr\left[h_t(x) = y^{(t)}\right].
\end{aligned}$$

The conditional independence assumption is only required in eqn. (2.7); before and afterwards exact transformations based on Bayes' theorem are applied. The latter of these application leads to a more convenient form of this classifier, because usually our individual model performance

estimates are of the form $\Pr\left[y \mid h_t(x) = y^{(t)}\right]$. Defining[5]

$$\alpha(x) := \frac{\prod_{1\leq t\leq k} \Pr\left[h_t(x) = y^{(t)}\right]}{\Pr\left[\vec{h}(x) = \vec{y}\right]}$$

allows to rewrite the equation above as

$$\Pr\left[y \mid \vec{h}(x) = \vec{y}\right] = \alpha(x) \cdot \Pr[y] \cdot \prod_{1\leq t\leq k} \textsc{Lift}((h_t(x) = y^{(t)}) \to y),$$

substituting $\textsc{Lift}((h_t(x) = y^{(t)}) \to y)$ for $\Pr\left[y \mid h_t(x) = y^{(t)}\right] / \Pr[y]$ (Def. 16, p. 22).
For boolean $\mathcal{Y} = \{y_+, y_-\}$ it is easier to consider the odds ratios

$$\begin{aligned} \beta(x) &:= \frac{\Pr\left[y_+ \mid \vec{h}(x) = \vec{y}\right]}{\Pr\left[y_- \mid \vec{h}(x) = \vec{y}\right]} \\ &= \frac{\Pr[y_+]}{\Pr[y_-]} \prod_{1\leq t\leq k} \frac{\textsc{Lift}((h_t(x) = y^{(t)}) \to y_+)}{\textsc{Lift}((h_t(x) = y^{(t)}) \to y_-)}, \end{aligned} \tag{2.8}$$

as $\alpha(x)$ cancels out, but it is still possible to recompute

$$\Pr\left[y_+ \mid \vec{h}(x) = \vec{y}\right] = \frac{\beta(x)}{1 + \beta(x)}. \tag{2.9}$$

based on eqn. (2.8). So, following the conditional independence assumption it is possible to combine rules to predict class probabilities, just knowing their LIFT values (Def. 16) and the class priors. In the case of prediction rules, these may also be considered to make predictions if they do not apply. This is justified by the following lemma.

Lemma 1 *For $\mathcal{Y} = \{y_+, y_-\}$ the following connection between any rule $A \to y_+$ and its dual rule $\overline{A} \to y_-$ holds:*

$$\textsc{Lift}(A \to y_+) > 1 \;\Leftrightarrow\; \textsc{Lift}(\overline{A} \to y_-) > 1$$

The precisions and LIFT *values of rule and dual rule may differ.*

Proof
We have

$$\begin{aligned} \Pr(A, y_+) > \Pr(A) \cdot \Pr(C) &\Leftrightarrow \Pr(A) - \Pr(A, y_+) < \Pr(A) - \Pr(A) \cdot \Pr(y_+) \\ &\Leftrightarrow \Pr(A, y_-) < \Pr(A) - \Pr(A) \cdot (1 - \Pr(y_-)) \\ &\Leftrightarrow \Pr(A, y_-) < \Pr(A) \cdot \Pr(y_-), \end{aligned}$$

and analogously

$$\begin{aligned} \Pr(A, y_-) < \Pr(A) \cdot \Pr(y_-) &\Leftrightarrow \Pr(y_-) - \Pr(A, y_-) > \Pr(y_-) - \Pr(A) \cdot \Pr(y_-) \\ &\Leftrightarrow \Pr(\overline{A}, y_-) > \Pr(y_-) - \Pr(y_-) \cdot (1 - \Pr(\overline{A})) \\ &\Leftrightarrow \Pr(\overline{A}, y_-) > \Pr(\overline{A}) \cdot \Pr(y_-). \end{aligned}$$

[5] The term cancels out, but has an intuitive meaning. A generalized version of the LIFT applied to sets of events. α^{-1} measures the degree of correlation in the ensemble in terms of this generalized LIFT.

		true		
		y_+	y_-	$\sum$
predicted	A (y_+)	0.2	0.1	0.3
	$\overline{A}$ (y_-)	0.3	0.4	0.7
	$\sum$	0.5	0.5	1.0

Table 2.1.: The contingency table shows an example for which the LIFTs of the two rules $A \rightarrow y_+$ and $\overline{A} \rightarrow y_-$ differ.

Rewriting the inequalities in terms of the LIFT yields

$$\text{LIFT}(A \rightarrow y_+) > 1 \Leftrightarrow \text{LIFT}(A \rightarrow y_-) < 1 \Leftrightarrow \text{LIFT}(\overline{A} \rightarrow y_-) > 1,$$

which simply reflects a positive correlation between predictions and label.

It is easy to construct a case in which the precisions and LIFT values of a rule and its dual rule differ. For example, the contingency matrix depicted in table 2.1 has the values

$$\Pr(A, y_+) = 0.2, \;\; \Pr(\overline{A}, y_-) = 0.4, \;\; \Pr(A) = 0.3, \;\; \text{and } \Pr(y_+) = 0.5.$$

This implies

$$\begin{aligned}
\text{PREC}(A \rightarrow y_+) &= \Pr(y_+ \mid A) = \frac{0.2}{0.3} = 2/3 \approx 0.67 \\
\text{PREC}(\overline{A} \rightarrow y_-) &= \Pr(y_- \mid \overline{A}) = \frac{0.4}{0.7} = 4/7 \approx 0.57 \\
\text{LIFT}(A \rightarrow y_+) &= \frac{\text{PREC}(A \rightarrow y_+)}{\Pr(y_+)} = 4/3 \approx 1.33 \\
\text{LIFT}(\overline{A} \rightarrow y_-) &= \frac{\text{PREC}(\overline{A} \rightarrow y_-)}{\Pr(y_-)} = 8/7 \approx 1.14
\end{aligned}$$

□

So even if considering the model class of classification rules, each rule $(A \rightarrow y_{+/-})$ can well be considered to partition the instance space into $\text{Ext}(A)$ and $\text{Ext}(\overline{A})$, implicitly making a prediction for both subsets. Basically, any LIFT value different from 1 is useful to improve a soft classifier ensemble. This will be shown in chapter 5. The more critical aspect of the approach discussed above is the conditional independence assumption. The following section shows a way to combine predictions that does not require this assumption.

2.6.3. Combining classifiers based on logistic regression

Logistic regression is a well-known and approved classification technique with a long history of successful applications. It applies to multi-class problems, but can easier be illustrated for binary classification problems. For a more detailed discussion, including the mathematical foundations and improved optimization techniques, please refer to (Komarek, 2004). A more compact introduction that includes multi-class extensions and a detailed discussion of generalized additive models can be found in (Hastie et al., 2001).

For mathematical simplicity, the target attribute $\mathcal{Y}$ is assumed to take values in $\{0, 1\}$ in this section, while the range of corresponding *estimates* $\widehat{\Pr}(\mathcal{Y} = +1 \mid x)$ is $[0, 1]$. First, the underlying ideas of logistic regression are discussed in general, before the applicability to combining classifier predictions is illustrated.

Logistic regression for continuous attributes

In the general case, the explaining attributes $A_1, \ldots, A_d$ are assumed to be numerical. As before, each example is of the form (x, y) with $x \in \mathcal{X} = A_1 \times \ldots \times A_d$ and $y \in \mathcal{Y}$. For each example the value of $\mathcal{Y}$ can be considered to be a random variable. Analogously, $A_1, \ldots, A_d$ are random variables, where the value of A_t is denoted as x_t. The expected value (or mean) of the target variable $\mathcal{Y}$ is denoted as $\mu(x)$, a function of x.

As a straightforward goal, one could directly try to approximate $\mu(x)$ in terms of a function linear in x. It is clear, however, that in this case the range of linear functions will generally not be constrained to $[0, 1]$. As a solution, a *link function* g that transforms the values of $\mu(x)$ is introduced. The functions linear in $A_1, \ldots, A_d$ are then chosen as to approximate $g(\mu(x))$. Denoting by β_t the factor for attribute A_t, and by β_0 the offset at the origin, the resulting estimate $\hat{g}(\mu(x))$ for $g(\mu(x))$ is hence

$$\hat{g}(\mu) = \beta_0 + \sum_{t=1}^{d} \beta_t x_t.$$

After extending x by an additional constant $x_0 = 1$, the function can be rewritten as $\beta^T x$, where β^T denotes the vector $(\beta_0, \ldots, \beta_d)$. The characteristic of logistic regression is the usage of a specific link function, the so called *logit link function*, defined as

$$\text{logit}(x) := \log\left(\frac{\mu(x)}{1-\mu(x)}\right) = \log\left(\frac{\Pr(\mathcal{Y}=1)}{\Pr(\mathcal{Y}=0)}\right). \tag{2.10}$$

For any linear model $\text{logit}(x) \approx \beta^T x$ that yields real-valued estimates we can easily compute corresponding estimates

$$\widehat{\Pr}(\mathcal{Y} = +1 \mid x) = \hat{\mu}(x) := \frac{\exp[\beta^T x]}{1 + \exp[\beta^T x]}$$

for the conditional distribution of $\mathcal{Y}$. The optimization step selects parameters β that yield a maximum-likelihood model. By definition, such models maximize the likelihood

$$\mathbb{L}(\mathcal{E}, \beta) := \prod_{(x,y)\in\mathcal{E}} \widehat{\Pr}(\mathcal{Y} = y \mid x) = \prod_{(x,y)\in\mathcal{E}} \left(\hat{\mu}(x)^y \cdot (1-\hat{\mu}(x))^{1-y}\right),$$

that is, the probability that the assumed model produced the labels of training set $\mathcal{E}$. This optimization problem is convex, but unfortunately cannot be solved in closed form. In practice it is tackled by computationally complex gradient-descent search strategies.

An adaptation for combining crisp classifier predictions

Besides uniform voting, weighted voting, and NAÏVEBAYES-like combinations, the framework of logistic regression provides a further alternative to combine the predictions of different base classifiers. The following paragraphs discuss why the generalized additive modeling framework does not fit this task sufficiently well. Subsequently it is shown that logistic regression on top of crisp base classifier predictions simplifies to a more precise NAÏVEBAYES-like combination, with basically both kinds of models sharing a single model class. Finally, the main advantages and disadvantages of both alternatives are discussed.

Generalized additive models A common approach found in the literature extends the logistic regression framework sketched above to *generalized additive modeling*. In this case, regression is not based on the original attributes, but each original attribute A_t is first transformed by a well-chosen function f_t. The remaining framework is not changed, so we still aim to find a maximum-likelihood model by selecting a vector β for estimating

$$\widehat{\text{logit}}(\mu(x)) = \beta_0 + \sum_{t=1}^{d} \beta_t f_t(x_t)$$

close to the true values $\text{logit}(\mu(x))$. The functions f_t are usually used to incorporate non-linear dependencies to a certain degree.

Generalized additive models seem to allow for flexible combinations of base classifier predictions, which are also functions defined over x, but combining classifiers differs in three ways from this kind of modeling. First, we do not have one base classifier per attribute, but most classifiers depend on several attributes. In turn, each attribute may be used by several classifiers, or by none. Second, the number of considered base classifiers is not necessarily equal to the number of attributes. Third, classifiers are assumed to make crisp predictions. Hence, for boolean classification tasks the regression can be assumed to be based on attributes taking only the values 0 and 1. This simplifies the setting compared to the case of generalized additive models considerably. The latter are usually based on functions taking continuous values.

NaïveBayes versus logistic regression When using only the predictions $h_t(x) \in \{0, 1\}$ of base models as explaining variables (attributes), then the kind of logistic regression that is applicable to the problem of combining base classifiers is very similar to NAÏVEBAYES. This can easily be seen when considering a logarithmic form of NAÏVEBAYES and comparing eqn. (2.10) to eqn. (2.8). The connection is that

$$\text{logit}(x) = \log\left(\frac{\Pr(\mathcal{Y} = +1)}{\Pr(\mathcal{Y} = -1)}\right) = \log \beta'(x),$$

where $\beta'(x)$ refers to the odds ratios used by NAÏVEBAYES. This leads to the following formula based on eqn. (2.8):

$$\widehat{\text{logit}}(x) = \underbrace{\log\left(\frac{\Pr[y_+]}{\Pr[y_-]}\right)}_{\beta_0} + \sum_{1 \le t \le k} \underbrace{\log\left(\frac{\text{LIFT}((h_i(x) = y^{(t)}) \to y_+)}{\text{LIFT}((h_t(x) = y^{(t)}) \to y_-)}\right)}_{\beta_t \cdot h_t(x)} \tag{2.11}$$

This is a linear function for estimating $\text{logit}(x)$, in the same syntactical form that is used by logistic regression. However, two differences between NAÏVEBAYES and logistic regression remain. First, logistic regression selects a maximum-likelihood model, while NAÏVEBAYES uses a coarse heuristic or the unrealistic assumption of conditional independence, respectively. Second, the reformulation does not work as simple as the under-braces of eqn. (2.11) suggest. Logistic regression models provide just one factor β_t per base classifier; NAÏVEBAYES uses different estimates for $\Pr(\mathcal{Y} = +1 \mid h(x) = 1)$ and $\Pr(\mathcal{Y} = +1 \mid h(x) = 0)$, leading to (almost) arbitrarily different LIFT ratios depending on whether a base model predicts the positive or the negative class. For boolean prediction tasks, NAÏVEBAYES models can be transformed accordingly, however, by "shifting" a constant offset to β_0 for each base model, so that a single weight per classifier suffices. This is further discussed in subsection 5.3.2. Please note, that the result

of this transformation is a weighted voting scheme. Hence, for boolean classification tasks all the discussed combination techniques, namely NAÏVEBAYES, logistic regression, and voting schemes, basically use the same class of models, but may assign different weights to models.

Hybrid solutions The illustrated connection between NAÏVEBAYES and logistic regression suggests, that the logistic regression framework is more precise than – or even subsumes – the NAÏVEBAYES-like strategy for combining base model predictions. It can be used to compute maximum-likelihood models based on a number of base classifiers without the problematic conditional independence assumption. This assumption is not justified in most cases. Moreover, unlike linear discriminant analysis and other simpler variants, logistic regression has the advantage to not introduce further assumptions, e.g., regarding the underlying distributions. Even the linear dependency assumption between the explaining attributes and $\text{logit}(x)$ is meaningless for boolean attributes.

The main disadvantage of logistic regression is that computing maximum-likelihood models is computationally expensive. The parameters are usually optimized by gradient-based search strategies, exploiting the convexity and the fact that the objective function is continuously differentiable. In contrast, NAÏVEBAYES classifiers require only a single scan over the training set to compute all the required probability estimates. This is a very desirable property when learning from very large databases. Another advantage of NAÏVEBAYES-like combinations is, that they are easily understood by a human analyst, because complex interactions between rules are not considered. This means that each rule weight can be interpreted without the context of other rules, which might sometimes be preferable in descriptive data analysis.

In later parts of this thesis base classifiers are mainly combined using a hybrid strategy. This property is shared by several weighted voting schemes. For reasons that will become obvious later, the models can be considered to be descendingly ordered by their abilities to separate classes *given the preceding base classifiers*. As a substantial simplification, the factors or weights (β_t) of the preceding classifiers 1 to $k-1$ are not changed, and β_t is chosen as 0 for each $t > k$ during the computation of the factor for base classifier k. Exploiting the connection between NAÏVEBAYES and logistic regression reflected by eqn. (2.11), β_k can then be optimized during a single scan over the example set. This may include an update of β_0. Iterating just once over the set of classifiers yields a set of parameters that is usually a sufficiently close approximation to the optimal weight vector β with respect to generalization performances.

In subsequent chapters this hybrid technique is shown to work well for sequential subgroup discovery (chapter 4) and boosting (chapter 5). Chapter 6 will illustrate that it even applies to classification tasks with changing target concepts, so-called *concept drifts*. The techniques will be presented taking a sampling-centered view, because sampling (i) provides a sound theoretical foundation of data mining, (ii) allows for large-scale applications, and (iii) is a powerful preprocessing operator at the same time. Before going into details, the next chapter hence provides a general overview of sampling techniques in the context of KDD.

3. Sampling Strategies for KDD

3.1. Motivation for sampling

The most challenging aspect of modern KDD applications is scaling up traditional data mining algorithms to rapidly growing volumes of data, sometimes in the order of terabytes. Volumes that do not allow to perform experiments in main memory become more and more important in practice, due to the increasing amounts of data collected, but with these applications a number of novel technical and conceptual problems emerge. The raw data that is subject to the analysis is usually stored in a relational database, in order to ensure data integrity and to establish a generic and efficient interface to various kinds of applications. Modern relational database management systems (RDBMSs) offer an optimized access, but it turns out that SQL interfaces do not necessarily meet the demands of data mining algorithms. The main problem is that most algorithms require to process the same example very often, in more or less random order.

From a practical point of view, it is reasonable to preprocess the data in an RDBMs (cf. chapter 8). However, one usually gains performance in the next step by caching the preprocessed training data in flat files, unless it fits into main memory. There are a good reasons in general to restrict data mining to subsamples in order to circumvent DBMS overhead. The first reason is of a technical nature; the communication overhead and several services provided by RDBMSs considerably slow down the data access in the context of data mining. Sarawagi et al. (1998) compare different ways of accessing data for mining association rules from databases. In essence, in their experiments caching the data in a file system clearly outperformed SQL-based solutions, stored-procedures, and an optimized implementation using a vendor specific programmer's interface to the relational database under consideration. Rüping (2002) describes a SUPPORT VECTOR MACHINE running as a JAVA stored procedure in an ORACLE database. The author points out that such an implementation may be useful for specific applications, but that it is not competitive if runtime is a major issue. Musick and Critchlow (1999) analyze scenarios that are typical for scientific applications: a computationally complex analysis of large high-dimensional datasets with high I/O rates. They report that solutions based on native code clearly outperformed DBMS-based queries in their experiments.

Besides the more technically motivated reasons to perform the data mining step outside the DBMS, there is a second, even stronger constraint in practice: Since most state of the art machine learning algorithms scale super-linearly, even training data that easily fits into main memory may cause unreasonably high computational costs, so exceeding main memory size drastically seems inappropriate. For example, a single experiment that evaluated boosted decision trees (10 iterations) by 10fold cross-validation took several (> 3) days for the Quantum Physics data set of the KDD Cup 2004[1]. This set contained only 40 MB of training data. YALE (Mierswa et al., 2006) was used as the learning environment, run on an AMD ATHLON MP 2100+ double processor machine.

Sampling is a general technique to tackle these problems. As this and following chapters will illustrate, there is a large spectrum of benefits that can be achieved by sophisticated sampling

[1] http://kodiak.cs.cornell.edu/kddcup/

strategies. The most commonly applied strategy is *uniform sub-sampling*, aiming at a speed-up of the data mining step, and at a reduction of main memory consumption. Based on a few reasonable assumptions, theoretically sound sub-sampling strategies allow to give probabilistic guarantees that the induced models are close to models trained from all the data. Section 3.2 discusses the theoretical foundations; practically relevant counterparts are described in section 3.3.

For many applications, further benefits can be achieved by sampling from another distribution than the one originally underlying the data. Appropriately resampled subsets are the input to the data mining step in this case. Some useful techniques that allow to sample from altered distributions are introduced in section 3.4. The most important strategies in this context are *stratification* (subsection 3.4.1) and *rejection sampling* (subsection 3.4.2), which are both shown to have a variety of interesting applications in the scope of KDD. The former strategy allows to simplify subgroup discovery when choosing the WRAcc metric; this will come in handy in the next chapter. The latter strategy constitutes the basis for knowledge-based sampling, which is also introduced in chapter 4, and will further be discussed in subsequent parts of this work; it is a generic way of making supervised data mining algorithms sensitive to probabilistic prior knowledge.

3.2. Foundations of uniform sub-sampling

Unless noted otherwise, in KDD the term *sampling* usually refers to uniform sub-sampling from a given set of examples. The goal of this kind of sampling is to construct a smaller training set without altering the distribution underlying the data. This section provides a few basic definitions, and it discusses important techniques that allow to compute confidence-intervals for estimated performances when training models from sub-samples.

3.2.1. Sub-sampling strategies with and without replacement

When discussing sub-sampling strategies, one should keep in mind that throughout this thesis (except for chapter 6) any *complete* example set is merely assumed to be an i.i.d. sample from an unknown probability density function (pdf) D defined over an instance space $\mathcal{X}$, or over an instance space with a designated target attribute, $\mathcal{X} \times \mathcal{Y}$, respectively. The following definition provides a starting point for discussing sub-sampling in general.

Definition 22 (Sub-sampling) *Any deterministic or probabilistic algorithm* A *that takes a sample* $\mathcal{E} \sim \mathrm{D}^{\mathfrak{n}}$ *of arbitrary size* $\mathfrak{n}$ *from* D *as its input and outputs a subset* $\mathcal{E}' \subseteq \mathcal{E}$ *of user-specified size* $\mathfrak{m} \leq \mathfrak{n}$ *is called a* sub-sampling algorithm*. Each such algorithm* A *implicitly defines a distribution with joint pdf* $\mathrm{D}_{\mathrm{A}} : (\mathcal{X} \times \mathcal{Y})^{\mathfrak{m}} \to \mathbb{R}^{+}$ *depending on the input pdf* D *and (just) operationalizes the transformation algorithmically.*

Please note that in general, following definition 22, the resulting samples are no longer i.i.d. For instance, pdf D_{A} may not correspond to the product density function $(\mathrm{D}')^{\mathfrak{m}}$ of any pdf D'. The algorithm might e.g., have a bias towards selecting sets of similar examples. For sub-sampling *uniformly*, however, there are additional constraints. The main concern in this case is to avoid the introduction of a bias. This means that the distribution implicitly defined by the algorithm over samples of target size $\mathfrak{m}$ has to match the resulting distribution if sampling $\mathfrak{m}$ examples i.i.d. from D; in more technical terms this translates into $\mathrm{D}'_{\mathrm{A}} = \mathrm{D}^{\mathfrak{m}}$, which obviously requires sampling $\mathfrak{m}$ instances of $\mathcal{X}$ *independently* as stated by the following constraint.

Constraint 1 *For any algorithm that sub-samples a set* $\mathcal{E}_m$ uniformly *from an input example set* $\mathcal{E}$*, the probability* $\Pr(e' \in \mathcal{E}_m)$ *to select an example* $e' \in \mathcal{E}$ *has to be independent of the selection of other examples:*

$$(\forall e' \in \mathcal{E})(\forall S \subset \mathcal{E} \setminus e') : \Pr(e' \in \mathcal{E}_m \mid S \subset \mathcal{E}_m) = \Pr(e' \in \mathcal{E}_m)$$

If the same example e' *occurs multiple times in* $\mathcal{E}$*, e.g. as* e_1 *and* e_2*, then the independence constraint requires that both occurrences are not considered identical, hence* $e_2 \in \mathcal{E} \setminus e_1$.

The following class of algorithms satisfies the necessary condition of preserving sample independence as demanded by constraint 1. It covers all sub-sampling strategies discussed in this thesis, not only uniform sub-sampling.

Definition 23 (I.i.d. preserving) *A sub-sampling algorithm* A *is called* i.i.d. preserving *if it outputs i.i.d. sub-samples of specified size from a distribution* D_A *whenever the input example set is an i.i.d. sample from a distribution* D*. The target distribution* D_A *depends on* D *and is implicitly defined by the algorithm* A.

Focusing on i.i.d. preserving sub-sampling simplifies matters a lot. The initial distribution and hence the target distribution are in fact generally i.i.d., which allows to describe the target distribution much more intuitively in one of the following two ways.

density function In contrast to D'_A (definition 22), the target D_A describes the probability to sample a single example. D_A is defined over $\mathcal{X}$, or over $\mathcal{X} \times \mathcal{Y}$, respectively, and does not depend on the target sample size or on the context of other examples being selected.

transformation Since D_A depends on D it is straight-forward to describe it in relative terms, for instance as a function of D.

For sub-sampling uniformly, providing a description of the latter kind is trivial, since by definition the transformation of D is the identity. Before discussing the properties and benefits of uniform sub-sampling, it is worth to discuss some basic algorithmic aspects.

For illustration we consider the case of (sub-)sampling from a database. If the i.i.d. assumption is justified and all of the available data are stored in random order or in the timely order they were sampled, then e.g., returning just the first m examples meets the requirements of a uniform sub-sampling algorithm. However, this most simple case is seldomly found in practice, since all the data in a database usually follow a semantically oriented or time-depended order.

Sub-sampling with replacement

When each example of an example set $\mathcal{E}$ is sub-sampled with the same probability, then repeatedly selecting a single example at random results in uniform sub-sampling *with replacement*. This is, because each element $e \in \mathcal{E}$ may occur several times in the resulting target sample. The probability of duplicates, and hence the bias introduced by this practically common sampling technique, depends on the ratio between the size of the original database and the target sample size. The phenomenon that a small number of m suffices to have a duplicate with high probability is sometimes referred to as the *birthday paradox*. For a database containing n tuples the probability p_m to have at least one duplicate in a sample of size m is

$$p_m = 1 - \frac{n}{n} \cdot \frac{n-1}{n} \cdot \frac{n-2}{n} \cdots \frac{n-m+1}{n} = 1 - \frac{n!}{(n-m)! \cdot n^m}.$$

Hence, the probability to have *no* duplicates $(1 - p_m)$ decreases exponentially fast with growing target sample size m.

Sub-sampling without replacement

Uniform sampling *without replacement* requires to sample each example at most once, which means to eliminate duplicates. It is easy to see that this strategy yields uniform sub-samples $\mathcal{E}_m \sim D^m$ without a bias whenever the input set $\mathcal{E}_n$, $n \geq m$, has been sampled i.i.d. from the initial distribution D: Uniform sub-sampling selects a subset of size $m < n$, so each example of $\mathcal{E}_n$ has the same chance $\frac{m}{n}$ of being selected, and all examples are selected independently. Please note that without the i.i.d. assumption of the initial distribution this is not necessarily the case, for instance if the probability to select a specific example depends on the previously selected one in the input set $\mathcal{E}_n$.

Sampling without replacement is more precise, as it does not introduce a bias. The main disadvantage lies in an increase of the computational overhead. No matter whether the primary key values of all selected examples are stored in memory[2] or duplicates are eliminated after sorting, in any case the asymptotic runtime for sampling increases by a logarithmic cost factor in the target sample size. As stated before, the main motivation when sampling uniformly, for example in the scope of predictive modeling, is to reduce the runtime of the applied learning algorithm. Most learning algorithms scale super-linearly. Hence, sampling techniques that are complex with respect to their runtime complexities are usually tolerable.

3.2.2. Estimates for binomial distributions

From a theoretical point of view, the common assumption of sampling independently from an unknown but stationary distribution (sampling i.i.d.) and sub-sampling from a given example set $\mathcal{E}$ have similar properties. In both cases, the sample is considered to provide an *incomplete* view, while the goal of data mining techniques is to identify models or patterns that are at least approximately optimal in terms of the complete data set or an unknown underlying distribution, respectively. Many of the relevant optimization and estimation problems of data mining can be formulated in terms of binomial distributions in this setting. This subsection discusses important techniques for estimating crucial quantities that emerge in the context of uniform sub-sampling. The common goal of these techniques is to provide probabilistic guarantees regarding the deviation of estimated from true values.

For boolean target attributes some useful theorems known as Chernoff bounds (Chernoff, 1952) and Hoeffding bounds (Hoeffding, 1963) have successfully been applied to a variety of analytical problems. These bounds apply to binomial distributions.

As a motivating example, the reader may think of estimating the accuracy of a hypothesis given a sub-sample. For a given sample size, both Chernoff and Hoeffding bounds allow to compute confidence bounds that reflect how likely the empirically observed values are "close" to the true values. For practical applications the utility of such estimates can be improved significantly by continuously computing more precise estimates after seeing parts of the data. Corresponding techniques are known under the names of *progressive* and *adaptive* sampling and are discussed in section 3.3.

We will reconsider the setting introduced in the context of the binomial test function (p. 23). Let p denote the probability of success in the underlying Bernoulli experiments, i.e. that an example is sampled from the subset for which a hypothesis makes a correct prediction. For i.i.d. samples, p is obviously a constant for each hypothesis. For a target sample of size m, we define boolean random variables X_i, $1 \leq i \leq m$, to be 1 in the case of a success, and 0 otherwise. Let $Y := \frac{1}{m} \sum X_i$ denote the fraction of successes for these m independent Bernoulli random

[2] To check whether a primary key is in the dictionary requires logarithmic time in the dictionary size.

variables. Y follows a binomial distribution with

$$\mathbb{E}(Y) = p \text{ and } \sigma(Y) = \sqrt{\frac{p(1-p)}{m}},$$

and m repetitions; this is denoted as $Y \sim B(m, p)$.

Theorem 3 (Chernoff bounds) *For any parameter $\lambda \in [0, 1]$ and random variable $Y \sim B(m, p)$ the following inequalities hold:*

$$\Pr(Y \geq (1+\lambda)p) \leq \exp(-\lambda^2 mp/3) \quad (3.1)$$
$$\Pr(Y \leq (1-\lambda)p) \leq \exp(-\lambda^2 mp/2) \quad (3.2)$$

The term $(1 \pm \lambda)p$ reflects a multiplicative deviation of Y from its expected value p.

As an example, consider the task to estimate the risk that a hypothesis with a true accuracy ($p = \text{ACC}$) of 75% performs no better than random guessing on an i.i.d. sample of size m. The estimated accuracy $\widehat{\text{ACC}}$ of a hypothesis is the fraction of correctly classified examples in this sample. We are interested in computing a sample size m, for which this risk is bounded by a fixed confidence parameter δ.

In the example, the deviation is $\lambda = 1/3$, because $(1 - 1/3) \cdot \text{ACC} = 50\%$. This allows to apply eqn. (3.1) to compute

$$\Pr(\widehat{\text{ACC}} \leq (1-1/3) \cdot \text{ACC}) \leq \exp(-(1/3)^2 m \cdot \text{ACC}/2)$$
$$\Leftrightarrow \quad \Pr(\widehat{\text{ACC}} \leq 1/2) \leq \exp(-\frac{1}{9}m\frac{3}{8}) = \exp(-\frac{m}{24}).$$

A sufficient criterion to fail ($\widehat{\text{Acc}} \leq 50\%$) with a probability of e.g., at most $\delta := 5\%$ can now easily be derived:

$$\exp(-\frac{m}{24}) \leq \delta \Leftrightarrow -\frac{m}{24} \leq \ln\delta = -\ln\frac{1}{\delta} \Leftrightarrow m \geq 24 \ln\frac{1}{\delta} = 24 \ln 20,$$

which is true for all sample sizes $m \geq 72$. In other words, the risk of sampling 72 examples i.i.d. for which a 75% accurate (or better) model misclassifies at least half of the examples is lower than 5%.

Hoeffding bounds are of a similar nature, but they address *additive* deviations of random variables from their true mean. Hence, they are also known under the name of *additive Chernoff bounds*. In the presented form they apply to binomial distributions, but also to other independent observations of m random variables with range $[0, 1]$. Any bounded random variable can be normalized to this range.

Theorem 4 (Hoeffding bounds) *For any parameter $\epsilon \in \mathbb{R}^+$ the following inequalities hold for random variables $Y \sim B(m, p)$:*

$$\Pr(Y - p \geq \epsilon) \leq \exp(-2\epsilon^2 m) \quad (3.3)$$
$$\Pr(Y - p \leq -\epsilon) \leq \exp(-2\epsilon^2 m) \quad (3.4)$$
$$\Pr(|Y - p| \geq \epsilon) \leq 2\exp(-2\epsilon^2 m) \quad (3.5)$$

A typical data mining application is to compute confidence bounds for the true accuracy ACC, given only an empirical estimate $\widehat{\text{ACC}}$. A major advantage of Hoeffding bounds for this application, when comparing the inequalities to those derived from theorem 3, is that the bounds are independent of the true value to be estimated, e.g. ACC.

For illustration, let the goal be not to over- or underestimate the true accuracy ACC of a given model by an additive constant of more than 10% when relying on a sample estimate $\widehat{\text{ACC}}$. An application of eqn. (3.5) yields the following bound:

$$\Pr(|\widehat{\text{ACC}} - \text{ACC}| \geq 0.1) \leq 2\exp(-2 \cdot (0.1^2)m) \leq 2\exp(-0.02m).$$

Now, a sufficient sample size m for a confidence parameter $\delta = 5\%$ can be derived:

$$2\exp(-0.02m) \leq 0.05 \Leftrightarrow -0.02m \leq \ln\frac{1}{40} \Leftrightarrow 0.02m \geq \ln 40 \Leftrightarrow m \geq 50\ln 40 \approx 184.44$$

For both Chernoff and Hoeffding bounds, δ decreases exponentially fast for a growing sample size m. The bounds are especially relevant for analytical settings, where the aim usually is to give asymptotical guarantees. Hoeffding bounds are also exploited by some data mining algorithms, e.g., by VFDT proposed by Domingos and Hulten (2000). However, the bounds are known to be rather loose, which increases the sample complexity of such algorithms unnecessarily. The following way of computing estimates yields better bounds when candidate sample sizes are in the range of a hundred or more observations.

Referring to the central limit theorem, the normal distribution approximates the binomial distribution sufficiently well, unless considering a very small number of samples. Let σ denote the standard variation of a binomial random variable $V \sim B(\mu(V), |\mathcal{E}|)$. Then, for sufficiently large sample sizes $|\mathcal{E}|$, the deviation of estimated from true mean, normalized with respect to standard deviation, approximately follows a standard normal distribution:

$$V' := \frac{V - \mu}{\sigma} \sim N(0, 1).$$

This directly allows to derive a two-sided confidence interval for the event that the mean of the sample deviates from the true mean of V by more than a fixed additive constant $z \in \mathbb{R}$:

$$\begin{aligned} \Pr(|V - \mu| > z) < \delta &\Leftrightarrow \Pr(V - \mu > z) < \delta/2 \\ &\Leftrightarrow \Pr(V' > z/\sigma) < \delta/2 \\ &\Leftrightarrow \Pr(V' < z/\sigma) > 1 - \delta/2 \end{aligned}$$

The *inverted standard normal distribution* allows to solve for z at a given confidence level δ. It can e.g., be utilized by table look-ups. For a standard normal random variable X, the term z_p refers to the value z for which $\Pr(X < z) = p$. In contrast to Chernoff and Hoeffding bounds, this method yields tight confidence-bounds up to the (quickly vanishing) difference between binomial and normal distribution.

It should be noted that using this technique for computing confidence bounds based on empirically observed values is unreliable for small samples, however. Poor samples may yield overly optimistic confidence bounds if the empirical standard deviation is used, which does not allow to give strong guarantees. The reason is that the standard deviation observed in a sample is a random variable itself. This shall be illustrated by the following example: When flipping a fair coin a few times, one may observe a series of only heads showing up. The sample frequency of heads is 1, and the sample variance is 0. Referring to the normal distribution and substituting the estimated values is very misleading now, because it indicates that our estimate of 1 is correct with arbitrary confidence. This is clearly wrong, because the true value is $1/2$.

Student's t-distribution is more precise when estimating the mean of a normally distributed random variable. This distribution correctly handles the standard deviation observable in a sample as a random variable, and thus has one degree of freedom less than sampled examples. It

better reflects the extractable information for normally distributed variables, i.e. the confidence bounds for the estimated mean. The problem in our setting is, that for small sample sizes the normal distribution is a poor approximation of the binomial distribution. Revisiting the coin flip example above, using Student's t-distribution yields the same value of 1 as an estimate for the mean, with an arbitrarily small confidence interval. With increasing degrees of freedom (larger samples) the approximation improves, because the difference between binomial and normal distribution vanishes very quickly. However, the same holds true for Student's t-distribution and the normal distribution, so using the latter for large sample sizes is sufficiently precise and more convenient.

There are several approaches of different complexity to circumvent any problems with small sample sizes, e.g. substituting the worst-case variance for binomially distributed random variables in the equations, using Hoeffding bounds, or the formula presented by Kohavi (1995). Scheffer and Wrobel (2002) suggest to take the size of samples into account, and to use Hoeffding bounds for evaluating on small samples and the estimates based on the normal distribution for evaluating on large samples. For sample sizes of 100 and more examples the latter can be considered reliable, except for means lying very close to 0 or 1.

As a more complex application of such bounds, let us consider the problem of association rule mining. At a reasonably high support level this task is well suited to be combined with sampling. The first reason is, that only rules for which a high fraction of examples provides evidence are considered relevant, the second is, that precision (alias confidence) is a metric based on counting the fraction of examples in subsets.

An early approach that exploits sampling for association rule mining has been presented by Toivonen (1996). As common in the literature, he tackles the problem of association rule mining using a two-stage approach. The first stage addresses the more crucial part of detecting all frequent itemsets, while the second derives the corresponding association rules, based on the results of the first.

He presents a sampling-based approach for mining frequent itemsets that exploits the lattice structure of all itemsets induced by set inclusion. Based on a uniformly drawn sub-sample the border between frequent and non-frequent itemsets is identified. Those itemsets in the lattice that are not frequent themselves, but all of their strict subsets are, are referred to as the *negative border*. After analyzing a sample, one subsequent full database scan is sufficient to validate the correctness of the corresponding negative border and to compute precise counts for all frequent itemsets.

In the case of sufficiently representative samples, the border can be identified correctly and this procedure yields all frequent itemsets at low computational costs. The procedure fails whenever an itemset is frequent in the database, but not in the considered sample. By definition this is the case if at least one frequent itemset lies on the wrong side of the negative border. For each frequent set the probability of the support follows a binomial distribution, because there is a fixed probability to sample an example that supports the itemset under consideration. Chernoff bounds allow to constrain the risk of errors, as illustrated by the following result provided by Toivonen (1996).

Proposition 1 *Let $\mathcal{I}$ denote a set of itemsets, and let $\delta, \epsilon \in (0,1)$ denote two parameters. Then, for a uniform random sample of size*

$$m \geq \frac{1}{2\epsilon^2} \ln \frac{2|\mathcal{I}|}{\delta},$$

the probability that for any of the itemsets in $\mathcal{I}$ the relative frequency in the sample and in the complete database differ by at least ϵ is at most δ.

Itemsets that are frequent in the sample but not in the database do not pose a problem, because the counts are explicitly computed in the next step. This fact can be exploited to reduce the probability of the other kind of error, simply by setting the minimum frequency threshold to a lower value than that which was originally specified by the user. Toivonen (1996) illustrates how to precisely reduce the threshold in order to meet a user given confidence not to err on the negative border:

Proposition 2 *Let* min_fr *denote an original threshold,* low_fr *be a lowered threshold, and* $\delta \in (0,1)$ *be a user-given confidence parameter. If these variables satisfy the constraint*

$$\text{low\_fr} < \text{min\_fr} - \sqrt{\frac{1}{2m} \ln \frac{1}{\delta}},$$

then the probability that a frequent set in the database is not frequent in a sample of size m *is at most* δ.

Clearly, decreasing the threshold increases the number of itemsets. Hence, in settings where the number of database scans is not the only critical issue, it should also be taken care that the number of candidate itemsets does not increase unnecessarily. An alternative sampling strategy for frequent itemset mining is discussed at the end of subsection 3.3.1.

3.3. Iterative refinement of model estimates

The last section introduced the most important techniques for estimating relevant quantities in the context of uniform sub-sampling. This section will illustrate how these techniques can be applied to a variety of different data mining problems. Subsection 3.3.1 will discuss the issue of finding an appropriate trade-off between computationally complex model induction based on large samples, on the one hand, and a reduction in accuracy when training on smaller sub-samples, on the other. Exploiting confidence-rated estimates for a larger class of data mining applications, e.g. subgroup discovery with different utility functions, is the subject of subsection 3.3.2. The benefits of explicitly integrating the sampling procedure into learning algorithms will be illustrated by presenting algorithms from the literature, some of which will re-occur in subsequent chapters.

3.3.1. Progressive sampling

When training a single classifier on a sub-sample, the decrease in runtime is generally bought at the price of a decrease in accuracy. For practical applications, it can be assumed that the gain in runtime is much higher than the decrease in model performance. This is because the runtimes of most learning algorithms scale super-linearly with the size of the samples, while at the same time the increase in accuracy rapidly slows down with each new example added to the training set, and practically may be assumed to level off at a specific but unknown sample size. A *learning curve* depicts the expected accuracy of a model selected by a specific learning algorithm depending on the sample size. John and Langley (1996) state that the following *power law* provides a good fit to learning curves according to experiments in machine learning and psychology:

$$\mathbb{E}\left(\text{Acc}(m)\right) \approx a - bm^{-\alpha}.$$

In this formula $\mathbb{E}\left(\text{Acc}(m)\right)$ denotes the (expected) model accuracy for a uniform random sample of size m, and a, b, and α are parameters, which for specific learning tasks may be estimated

from empirically observed accuracies. Such curves may also be used to extrapolate learning curves, in order to find a value of m that yields models satisfying specific constraints, like being ϵ-close (Def. 1, p. 13) to a model that is trained from all the data.

Progressive sampling techniques train models iteratively, using an *increasing* sample size. The procedure continues as long as the performances of the corresponding models increase. This requires the assumption that learning curves are monotone, which usually holds true in practical applications. For incremental learning algorithms like NAÏVEBAYES it is comparably easy to augment the training sample by one example at a time, since each update is cheap. In the more general case of learners that learn "from scratch" each time, this strategy is too costly, which motivates the notion of a *sampling schedule*.

Definition 24 *For a training set of size n, a sampling schedule is a set $\{m_0, m_1, \ldots m_k\}$ of $k+1$ increasingly indexed integers satisfying $0 < m_0$, $m_k \leq n$. In iteration i a* progressive sampling strategy *trains and evaluates models with sample size m_i until convergence, which is assumed to be detectable.*

This approach directly addresses the fact that most learning algorithms scale super-linearly. The costs of inducing models from small samples are hence neglible, while it will usually just take a few costly iterations to identify an approximately optimal model. For complex learners and large samples this strategy can be expected to often outperform a single training step based on an unnecessarily large sample.

A sample size that is sufficiently large can only be found experimentally in practice, because this property depends on the combination of a specific data set and learning algorithm at hand. Thus, an interesting question is how to construct efficient sampling schedules.

Two instances that have been discussed in the literature are the arithmetic and the geometric sampling schedules.

Definition 25 *An* arithmetic schedule *for an example set of size n starts with a sample of size m_0, which is iteratively increased by a fixed number c of examples. This results in schedules of the form*

$$M_a := \{m_0, m_1, \ldots, m_k\},\ \ m_i := \min(m_0 + c \cdot i, n),\ \ k := \lceil (n - m_0)/c \rceil$$

for given parameters $m_0, c \in \mathbb{N}^{\leq n}$.

Assuming an unknown minimal sample size m_{opt} that leads to (almost) optimal expected performance, the arithmetic schedule may result in many training iterations if the selected constants turn out to be inappropriate, e.g. $m_0 \ll m_{opt}$ and c chosen too small. Guessing bad constants is less problematic for geometric schedules.

Definition 26 *For an example set of size n the* geometric schedule M_g *for $m_0 \in \mathbb{N}$, $0 < m_0 \leq n$ and $c \in \mathbb{R}^+$ is defined as*

$$M_g := \{m_0, m_1, \ldots, m_k\},\ \ m_i := \min(\lceil c^i \cdot m_0 \rceil, n),\ \ k := \lceil \log_c \frac{n}{m_0} \rceil$$

Progressive sampling has been applied in various data mining contexts. For example, Domingo et al. (2001) report experiments with an algorithm that performs progressive sampling using a geometric schedule, especially for boosting decision stumps (cf. chapter 5). They conclude that – without any decrease in accuracy – progressive sampling helps to decrease runtime performance significantly, even if the considered dataset is of main memory size.

Provost et al. (1999) provide a proof that for super-linear (non-incremental) learning algorithms geometric sampling schedules are asymptotically optimal regarding runtime, assuming that convergence ($m_i \geq m_{opt}$) is detectable after learning from a sufficiently-sized sample. However, the authors point out that in practice detecting convergence remains a crucial issue, since running a learner on a number of samples that are larger than m_{opt} will generally degrade the overall runtime performance compared to progressive sampling until m_{opt}.

More recent work by Leite and Brazdil (2004) addresses the problem of estimating m_{opt} by meta-learning. The authors report experiments with a database of learning curves from 60 benchmark datasets. For a new dataset, classifiers were trained for the first sample sizes of a geometric schedule. Based on the resulting accuracies and other criteria of the datasets the most similar learning curves from the database were retrieved in a k-nearest neighbor fashion. The value of m_{opt} was estimated by averaging the corresponding values of the retrieved curves. Leite and Brazdil (2004) report a decrease in runtime of about one order of magnitude, because often several iterations of running the base learner could be skipped. In some cases, the accuracy decreased, however, because the learning process was stopped too early. Adaptive sampling seems to benefit from meta-learning, but several issues are not yet well understood. In particular, there are several constants to be set in the reported approach, the roles and impacts of which need to be evaluated in more detail.

Progressive sampling also applies to unsupervised learning tasks. Chen and Yang (2005) exploit a corresponding technique to estimate the sampling error in the context of frequent itemset mining. For a given example set $\mathcal{E}$ they define the sampling error (SE) of a single item A_i in a sample S_m of size m as

$$\mathrm{SE}(A_i, S_m) := \left| \frac{|\{A_i(x) \mid x \in S_m\}|}{m} - \frac{|\{A_i(x) \mid x \in \mathcal{E}\}|}{|\mathcal{E}|} \right| = |\mathrm{BIAS}_{D_\mathcal{E}}(S_m \rightarrow A_i)|\,,$$

where $A_i(x)$ denotes the event that example (or tuple) x contains item A_i, and $D_\mathcal{E}$ denotes the uniform distribution of $\mathcal{E}$. In the scope of this thesis the idea of a "complete" database – and thus exact frequencies – can be generalized to probabilities of seeing specific items when sampling from an underlying distribution D. For a set of d different items $A^* := \{A_1, \ldots, A_d\}$ the individual sampling errors on samples S_m are aggregated, which is captured by the following definition:

$$\mathrm{A\_SE}(A^*, S_m) := \sqrt{\frac{\sum_{j=1}^{d} \mathrm{SE}(A_i, S_m)^2}{d}}.$$

Chen and Yang (2005) argue that the sampling error dominates the model accuracy of association rules, so any schedule that minimizes the sampling error up to a small fraction can basically also be expected to allow for reliable association rule mining. The authors present an algorithm that scans a database once, collects the counts for all single items, and simultaneously simulates sampling with different target sample sizes. To simulate the relevant aspects of sampling it is sufficient to maintain counts for the selected items. This information is used in the next step to estimate the sampling errors of all samples. An appropriate sample size is identified and used for progressive sampling. For the specific task of association rule mining, the algorithm will generally provide a cheap solution, requiring a single database scan in advance. The authors assessed the performance using an arithmetic schedule, but it can easily be replaced by a more efficient geometric schedule as described above.

It is worth noting that the sampling error computation is a heuristic approach, based on specific properties of the frequent itemset mining task. The tasks of detecting convergence and of finding optimal schedules remain crucial issues when sampling progressively in more general settings.

3.3.2. Adaptive sampling

Progressive sampling techniques treat learning algorithms as black boxes, evaluate their learning curves for specific sample sizes and stop as soon as they have enough evidence that the curves are saturated. Although this kind of sampling is practically relevant, it has two disadvantages. First, the observed saturation can be misleading due to statistical fluctuations, so there is an uncontrolled risk of stopping too early. Second, when using the black box approach, valuable information is wasted. In many settings the evaluation procedure can effectively be integrated into the learners, which helps to rule out poor model candidates early on with high confidence.

The idea of adaptive sampling is mainly illustrated for the task of subgroup discovery (Klösgen, 1996), because the most interesting aspect is how to give probabilistic guarantees for different potential evaluation metrics, while the class of models that is used by an algorithm is of minor importance. As stated before, optimizing utility functions (Def. 19, p. 24) that formalize the degree to which a rule (or more generally: "model") is deemed interesting covers a broad class of machine learning tasks.

Confidence bounds for rule evaluation metrics

For large databases it is attractive to compute quality estimates of candidate rules with respect to uniformly drawn sub-samples. Formally, this defines a different family of evaluation functions that are not based on the true density function D underlying $\mathcal{X} \times \mathcal{Y}$.

Definition 27 (Estimator function) *Let $q_D : \mathcal{H} \to \mathbb{R}$ be a utility function that assigns a real-valued score to each rule h from any hypothesis space $\mathcal{H}$ with respect to any underlying density function D. Let further for each sample $\mathcal{E}$ the term $D_\mathcal{E}$ denote the uniform distribution of $\mathcal{E}$. Then the* estimator function $\hat{Q} : \mathcal{H} \times (\mathcal{X} \times \mathcal{Y})^{\mathbb{N}} \to \mathbb{R}$ *of q_D is defined as*

$$\hat{Q}(h, \mathcal{E}) := q_{D_\mathcal{E}}(h).$$

In the general case D is unknown, so we have to rely on estimates computed from samples. The definition above is very general, and subsumes two different important scenarios. In the first, the complete data set is in fact considered to be an i.i.d. sample, and the goal is to learn about the underlying joint pdf D. This is a *classification* setting, aiming at generalizing the data and making predictions in the future. In the second scenario, we sub-sample from a large database, so we define D as the uniform distribution of all examples contained in that database. In this case, evaluating on all the data would yield precise results, but might simply be too expensive. Hence, it is desirable to get utility estimates based on sub-samples that reflect the values of the large superset. This is rather a *descriptive* learning task.

In its original formulation, the task of subgroup discovery is to exactly find the k best rules with respect to a user-specified utility function. Now, the goal is to provide guarantees that rule utilities estimated on samples are close to the true utilities. In a sampling context there is always a risk of facing a misleading sample, so the formal learning problem has been adapted accordingly. Similar to the PAC learning framework (cf. section 2.3), the sample complexity is estimated for the problem of finding the approximately k best rules with high confidence (Scheffer & Wrobel, 2001):

Definition 28 (Approximately k-best rules problem) *Let $\delta \in (0, 1)$ denote a given confidence parameter and $\epsilon \in \mathbb{R}^+$ denote a highest acceptable error interval. Then the* approximately k-best rules problem *for utility function q_D is to identify a set G of k rules, $G \subseteq H$, such that, with*

confidence (probability) of at least $1-\delta$*:*

$$(\forall h' \in \mathcal{H} \setminus G) : Q_D(h') \leq \min_{g \in G} (Q_D(g) + \epsilon) .$$

Scheffer and Wrobel (2002) provide a detailed analysis of the sample complexity for different utility criteria. The different nature of the corresponding functions is reflected by the fact that confidence bounds for individual rules are function-dependent. The notion of a *utility confidence interval* plays an important role in this context.

Definition 29 (Utility confidence interval) *A function* $E : \mathbb{N} \times \mathbb{R} \to \mathbb{R}^+$ *is called a* utility confidence interval*, if for all* $m \in \mathbb{N}$ *and* $0 < \delta \leq 1$ *with a probability of at least* $1-\delta$ *an i.i.d. sample* $\mathcal{E} \sim D^m$ *of size* m *misleads the empirical estimate* $\hat{Q}(h, \mathcal{E})$ *of* Q_D *by at most* $E(m, \delta)$*:*

$$(\forall h \in \mathcal{H}) : Pr_{\mathcal{E} \sim D^m} (|\hat{Q}(h, \mathcal{E}) - Q_D(h)| \leq E(m, \delta)) \geq 1 - \delta \tag{3.6}$$

As in the PAC model, the bounds must apply at each confidence level δ. One of the results is that – for successful applications of adaptive sampling – it has to be possible for each $\delta > 0$ to shrink the utility confidence intervals $E(m, \delta)$ to arbitrarily small values $\epsilon > 0$ by increasing the sample size m. A surprising negative implication is, that for some practically relevant utility measures it is not possible to provide sufficiently strong probabilistic guarantees at all, regardless of the sample size. One of these measures is the Gini index, known e.g. from decision tree induction (Breiman et al., 1984). This measure is order-equivalent[3] to

$$f(A \to C) := \frac{Cov(A \to C)}{1 - Cov(A \to C)} \cdot BIAS(A \to C)^2. \tag{3.7}$$

A publicly available proof seems to be lacking, so please refer to a proof by the author in appendix C, showing that these two utility functions are identical up to a constant additive term[4]. This directly implies that the Gini index and the utility function in eqn. (3.7) have identical confidence bounds, because, as definition 29 shows, additive constants have no effect on $E(m, \delta)$. For a detailed analysis of the utility function above, please refer to (Scheffer & Wrobel, 2002).

Before discussing how to exploit the confidence bounds algorithmically, an overview of such bounds for different utility functions is given. All the considered functions can be rewritten as combinations of binomially distributed random variables, basically the coverage and bias of rules. For these random variables the techniques outlined in subsection 3.2.2 allow to give estimates with associated confidence-bounds. Chernoff and Hoeffding bounds are not very tight, but still reliable when the empirical estimates are based on small samples. The corresponding bounds are not further discussed at this point, because the utility of adaptive sampling seems to be low for small samples. For larger samples the normal distribution is a sufficiently precise approximation of the binomial distribution. This allows to compute precise estimates of the random variables' means, but also of their standard deviations.

There are three relevant standard deviations when computing confidence intervals: ACC is estimated by averaging the 0/1-losses (cf. p. 10) of examples. There is just one random variable for this metric, and hence just one standard deviation, referred to as σ_h. For selecting rules, ACC is not a common metric, but functions based on the coverage and on the bias of the covered subset are preferred. The standard deviation of the coverage is referred to as σ_g. The standard deviation

[3] Two utility functions are called order-equivalent, if they always induce the same preference ordering of rule sets or model candidate sets.

[4] In the context of selecting an attribute for splitting, the class priors are constants.

Utility function	Confidence bound $E(m,\delta)$ and rule-dependent version $E_r(m,\delta)$
Acc	$E(m,\delta) = \frac{z_{1-\delta/2}}{2\sqrt{m}}$ $E_r(m,\delta) = z_{1-\delta/2}\sigma_h$
WRAcc	$E(m,\delta) = \frac{z_{1-\delta/4}}{\sqrt{m}} + \frac{(z_{1-\delta/4})^2}{4m}$ $E_r(m,\delta) = z_{1-\delta/4}(\sigma_g + \sigma_p + z_{1-\delta/4}\sigma_g\sigma_p)$
$Q^{(0.5)}$	$E(m,\delta) = \sqrt{\frac{z_{1-\delta/4}}{2\sqrt{m}} + \frac{z_{1-\delta/4}}{2\sqrt{m}}} + \sqrt{\frac{z_{1-\delta/4}}{2\sqrt{m}}\frac{z_{1-\delta/4}}{2\sqrt{m}}}$ $E_r(m,\delta) = \sqrt{\sigma_g z_{1-\delta/4}} + \sigma_p z_{1-\delta/4} + \sqrt{\sigma_g z_{1-\delta/4}}\sigma_p z_{1-\delta/4}$
$Q^{(2)}$	$E(m,\delta) = \frac{3}{2\sqrt{m}}z_{1-\delta/2} + \frac{m+\sqrt{m}}{4m\sqrt{m}}(z_{1-\delta/2})^2 + \frac{1}{8m\sqrt{m}}(z_{1-\delta/2})^3$ $E_r(m,\delta) = (2\sigma_g + \sigma_p)z_{1-\delta/2} + (\sigma_g^2 + \sigma_g\sigma_p)(z_{1-\delta/2})^2 + \sigma_p\sigma_g^2(z_{1-\delta/2})^3$

Table 3.1.: Utility functions and their corresponding confidence bounds $E(m,\delta)$ for large values of m (Scheffer & Wrobel, 2002). All values $E_r(\ldots)$ exploit properties of specific rules, basically their standard deviations σ_g, σ_p, and σ_h. Lemma 2 shows that, in fact, the tighter bounds for Acc also apply for WRAcc.

of another random variable, counting the number of examples for which a rule is applicable and correct, is denoted as σ_p. This corresponds to the standard deviation of the bias. Whenever we are computing intervals for a specific rule, estimates of these parameters help to compute tighter bounds. Without any specific rule under consideration it is still possible to substitute the worst-case deviations. This yields two different confidence intervals for each utility function.

Table 3.1 lists corresponding bounds for large sample sizes. The bounds shown for Acc even hold for a much broader class, the class of so-called *instance-averaging* functions.

Definition 30 (Instance-averaging) *A utility function* $Q_D : \mathcal{H} \to \mathbb{R}$ *is called* instance-averaging, *if there is a function* $f : \mathcal{H} \times \mathcal{X} \times \mathcal{Y} \to [0, R]$, *with* $R \in \mathbb{R}+$, *so that* Q_D *can be rewritten as*

$$Q_D(h) = \int_D f(h,x,y)\ dx\ dy.$$

The corresponding estimator function is

$$\hat{Q}(h,\mathcal{E}) = \frac{1}{|\mathcal{E}|}\sum_{(x,y)\in\mathcal{E}} f(h,x,y).$$

With f chosen as the $0/1$ loss, Acc is just one example of an instance-averaging function. It has a range of $[0,1]$, which is required for the upper (rule-independent) bound in table 3.1, and for confidence intervals derived from Hoeffding bounds (not shown). All loss functions that share this range, e.g. the root mean squared error (Def. 11, p. 20), can be substituted for f without losing any of these guarantees.

As the following proof by the author shows, this class is even broader than one might expect.

Lemma 2 *For known class priors the* WRAcc *metric is instance averaging.*

Proof
Let the constant π_c denote the known class prior $Pr[C]$, and for each rule $A \to C$ let $Ext(A)$, $Ext(C) \subseteq \mathcal{X} \times \mathcal{Y}$ denote the extensions of the antecedent A and consequence C, respectively.

Defining

$$w(A \to C, x, y) := \begin{cases} 1 - \pi_c, \text{ for } x \in \mathrm{Ext}(A) \wedge y \in \mathrm{Ext}(C) \\ 0 - \pi_c, \text{ for } x \in \mathrm{Ext}(A) \wedge y \notin \mathrm{Ext}(C) \\ 0 \qquad , \text{ for } x \notin \mathrm{Ext}(A) \end{cases}$$

for each $(x, y) \in \mathcal{X} \times \mathcal{Y}$ allows to rewrite WRAcc as

$$\begin{aligned} \mathrm{WRAcc}_D(A \to C) &= \mathrm{Cov}_D(A \to C) \cdot \mathrm{Bias}_D(A \to C) \\ &= \Pr_D[A] \cdot (\Pr_D[C \mid A] - \Pr_D[C]) \\ &= \Pr_D[A, C] - \Pr_D[A] \cdot \Pr_D[C] \\ &= \int_D w(A \to C, x, y)\, dx\, dy, \end{aligned} \tag{3.8}$$

and its estimator function as

$$\widehat{\mathrm{WRAcc}}(A \to C, \mathcal{E}) = \frac{1}{|\mathcal{E}|} \sum_{(x,y) \in \mathcal{E}} w(A \to C, x, y).$$

Substituting w in eqn. (3.8) is valid, because the first term of w (0 or 1) computes $\Pr[A, C]$, and the second one ($-\Pr[C]$ or 0) computes $-\Pr[A] \cdot \Pr[C]$ for known $\Pr[C]$. In order to rescale the range, we can finally add the constant prior π_C, leading to a new instance-averaging utility function

$$\mathrm{WRAcc}'(A \to C) := \mathrm{WRAcc}(A \to C) + \pi_C$$

with the desired range of $[0, 1]$. Transforming a utility function by additive constants has no effect on its confidence intervals. The function WRAcc' inherits the bounds of Acc, which hence also hold for WRAcc. □

This result simplifies computations, yields tighter bounds than those derived by Scheffer and Wrobel (2002) (cf. table 3.1), and hence reduces the sample complexity of adaptive sampling algorithms. Please note, that class priors are global properties of the data; if they are unknown it is cheap to get precise estimates with high confidence. Subsection 3.4.1 discusses further connections between Acc and WRAcc, providing a simple explanation for lemma 2.

Algorithms for adaptive sampling

Most of the popular utility functions can be evaluated in an adaptive sampling framework, using the confidence bounds discussed above. For this class of functions, the *generic sequential sampling* (GSS) algorithm by Scheffer and Wrobel (2002) allows to give probabilistic guarantees to find approximately optimal rules in the sense of definition 28. It usually requires far less examples than any static approach that computes the worst-case sample complexity in advance.

The hypothesis space $\mathcal{H}$ is required to be finite for GSS, because for each element of $\mathcal{H}$ a separate estimate is maintained: After each sampled example the algorithm updates the counts for covered positives and negatives for *each* rule in $\mathcal{H}$ (still) under consideration[5], and computes the corresponding utility scores. As the sample size increases, some rules will usually receive

[5] For more general tasks the full contingency matrix is stored.

a very low score compared to others, which allows to discard them as soon as the confidence bounds are tight enough. This requires to keep track of the k empirically best rules in $\mathcal{H}$. Definition 29 allows to compute an upper-bound for the probability that a "good" rule is discarded by mistake. Based on the union bound, the algorithm distributes half of the tolerable probability of failure ($\delta/2$) to the iterations and rules: For an upper-bound of m examples that need to be sampled in the worst case, each of the $|\mathcal{H}|$ rule evaluations is allowed to fail with a probability of at most $\delta/(2m|\mathcal{H}|)$. Applying this confidence parameter at each local evaluation, the risk of discarding any of the best rules is upper-bounded by $\delta/2$. The algorithm terminates as soon as there are at most k rules left, which are then known to be approximately optimal (up to an ϵ) with a probability of at least $1-\delta$.

A upper-bound m based on the static sample complexity of the task is computed as a fallback. This allows to terminate the search after a pre-computed number of examples. If GSS does not terminate due to ruling out all but k candidates after m iterations, then it is valid to output the empirically k best rules at that time. This bound is also of a probabilistic nature, of course, so the algorithm "reserves" the "remaining" probability of failure, namely $\delta/2$, for this second termination criterion.

The asymptotic sample bound m for the approximately k-best rules problem computable in advance (static) is

$$m = O\left(\frac{1}{\epsilon^2}\log\frac{|\mathcal{H}|}{\delta}\right) \tag{3.9}$$

for the measures ACC and $q^{(\alpha)}(A \to C) = \text{COV}^{\alpha}(A \to C) \cdot \text{BIAS}(A \to C)$, $\alpha > 0$.

The rule-specific confidence bounds exploited by GSS will usually reduce the sample complexity drastically. Referring to an empirical study, the authors report substantial improvements over the sample complexities computable from eqn. (3.9). This is a consequence of the fact that the utilities of rules in $\mathcal{H}$ will usually vary over the full range. The worst-case is a large number of equally well performing rules. In this case, adaptive sampling may fail to reduce the required sample size, so the bound above is at the same time a worst-case bound for the sample complexity of adaptive sampling.

Further algorithms based on adaptive sampling have been proposed by (Hulten & Domingos, 2002). They also make use of Hoeffding bounds, but implicitly confine themselves to the evaluation of instance-averaging functions in a framework similar to the approximately k-best rules problem. The interesting aspect is, that they point out the broad applicability of adaptive sampling. One of their algorithms, the VFDT decision tree learner (Domingos & Hulten, 2000), decides which attribute to split on by comparing the best alternatives in an adaptive sampling framework. As for GSS: If two decision tree splitting candidates perform very differently, then it is easy to identify the better one; if both candidates perform about equally well, then an adaptive sampling algorithm may select any of them with just a small expected loss in accuracy. Strong probabilistic guarantees are provided for the constant time and memory algorithm VFDT. They state that its results will often be very close or identical to the tree that would have been selected after processing all the data. This algorithm is also capable of mining from data streams (cf. chapter 6), a result of its sampling-based nature.

As (Hulten & Domingos, 2002) point out, the idea of applying Hoeffding bounds also applies to incremental or active learning tasks. Basically any kind of discrete search based on instance-averaging evaluation functions can be addressed by estimating the scores of all candidates in a progressive sampling framework by applying Hoeffding bounds and by selecting the single or k approximately best solutions at each step. Exemplarily, this strategy has been applied to the problem of inducing a Bayesian network from very large training sets. The key observation

when taking the sampling approach is that the main decision of which network structure performs best can be decomposed. Estimates can be computed by averaging over a sample for the selected learner (Heckerman, 1995). For huge training sets, the learning task can hardly be addressed without sampling. As for VFDT, the proposed strategy allows to give strong asymptotic guarantees that the network is close to the one trained from all the data.

As mentioned above, the methods used by Hulten and Domingos (2002) allow only to estimate instance-averaging functions in settings where the underlying optimization problem is close to the approximately k-best rules problem. Hoeffding bounds do not apply to discrete search problems that require optimal solutions or have inherently complex utility functions. An impressive example of the latter issue is decision tree learning based on the popular Gini index. Precisely evaluating utility functions that are equivalent to eqn. (3.7) is intractable when using incomplete samples, regardless of their size. As discussed, the Gini index is order-equivalent to this utility function and inherits the intractability result, because the confidence bounds do not vanish for large sample sizes.

3.4. Monte Carlo methods

The previous sections illustrated the utility of uniform sub-sampling techniques for different data mining tasks. The main benefit in KDD is a decrease in runtime paired with probabilistic guarantees not to miss any relevant pattern, or to induce a model close to one a data mining algorithm would output after processing all the data, respectively.

This section discusses more general sampling strategies. With Monte Carlo sampling techniques it is possible to sample exactly from a *new* distribution, which is defined in terms of the distribution originally underlying the data. Only two very simple Monte Carlo techniques are applied in this work. The first one is stratification, which is discussed in subsection 3.4.1. The second technique is called rejection sampling and will be discussed in subsection 3.4.2. It is very general, but sometimes lacks efficiency. Mackay (1998) and Neal (1993) provide compact introductions to Monte Carlo sampling methods, also covering more sophisticated techniques, which are of minor relevance to this work, however. Rejection sampling provides the theoretical foundation for knowledge-based sampling, a novel technique that will be proposed in the next chapter. Stratification will be utilized in this context to simplify knowledge-based sampling for sequential subgroup discovery, and it is also a major foundation of a derived boosting technique discussed in chapter 5.

3.4.1. Stratification

Probably the simplest randomized method that effectively samples from another than from the original distribution is stratification. Unless mentioned otherwise, the term *stratification* describes the process of altering a distribution, so that all classes are equally likely; other target class distributions can be constructed using the same techniques. The following definition shows how to accordingly define a new probability density function based on the original pdf underlying the data.

Definition 31 *For* $D : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^+$ *the* stratified random sample density function D' *of* D *is defined as*

$$D'(x, y) := \frac{D(x, y)}{|\mathcal{Y}| \cdot \Pr_{(x',y')\sim D}[y = y']}$$

for each $(x, y) \in \mathcal{X} \times \mathcal{Y}$.

If $\mathcal{Y}$ functionally depends on $\mathcal{X}$, with $f : \mathcal{X} \to \mathcal{Y}$, then this definition simplifies to

$$D'(x) := \frac{D(x)}{|\mathcal{Y}| \cdot \Pr_D[f(x)]}.$$

In less technical terms, D' is defined by rescaling D by a constant factor for each class, so that the classes are equally like with respect to D' (or follow any other specified ratio), but the density of each instance, when conditioning on the class, does not change from D to D'. The following paragraphs discuss stratification techniques and typical data mining applications.

Classical objectives

A typical application of stratification is to reduce certain kinds of biases in the training data. If for example, in a poll only a fraction of 10% of all responses were made by men, although it should be about 50%, then stratification may be a well-suited technique to correct this simple kind of *sample selection bias*. The result is a sample with a gender ratio that is approximately identical to the ratio observed in the overall population. Stratification is a sound modification if both, the underrepresented 10% group and the overrepresented 90% group are proportional ("representative") sub-samples of their classes.

Another problem with many real-world datasets is that class distributions are often highly skewed. Among the typical examples are the detection of fraud, network intrusion, and medical applications. If the fraction of fraudulent credit card transaction is e.g. 1%, then even the default classifier always predicting "no fraud" has an accuracy of 99%. In section 2.5, a general analysis framework based on ROC analysis has been discussed, which allows to consider varying class skews and misclassification costs as part of the classifier selection procedure. For varying skews and costs, maximizing the AUC to rank examples by their risk of being positive is a straightforward optimization goal for highly imbalanced datasets. Stratification has been reported to work well in such settings (Chawla et al., 2002).

In turn, for any fixed slope in ROC space defined by costs and skews, a separate specially targeted classifier can be trained after a specific step of stratification. As illustrated before, asymmetric costs can be handled just as implicitly changed class skews. This implies that a technique that is able to transform a skew-sensitive classification problem into a plain accuracy maximization problem allows to incorporate skews and costs in a very generic fashion, without any need to adapt base learning algorithms. At the end of this subsection (p. 63) it is shown that stratification is well-suited to address this problem.

Methods

The next paragraphs show how to operationalize the stratification specified by Def. 31. For a procedure that samples from D it is easy to design a Monte Carlo algorithm for sampling m examples from D'. If the target attribute is binary, then one can simply flip a coin m times and decide to include as many positive examples as heads showed up, say m_+, and choose the number of negatives as $m_- := m - m_+$, the number of tails. To construct a sample from D' now, it is sufficient to discard all but the first m_+ positives and first m_- negatives. As required, the class probabilities are both $1/2$, and the conditional probabilities (or densities) to see a specific example given the class under D' are by definition and construction identical to the probabilities of D. Hence, this simple technique samples exactly from the new distribution D' introduced by definition 31. It is easily seen that this algorithm also preserves the assumed i.i.d. nature of D in the sense of definition 23 (p. 43). This kind of sampling also applies if $|\mathcal{Y}| > 2$; it

is sufficient to replace the "coin" by another appropriate uniformly distributed (multinomial) random experiment.

To guarantee highly informative samples it is common practice to avoid the internal random experiment, but to sub-sample the same number of examples from each class uniformly without replacement. The rationale behind this simpler strategy is that usually a learner aims to approximate the pdf D' by choosing an appropriate model. This seems easier without additional variance introduced by an internal random experiment. Nevertheless, the technique sketched above samples from D' exactly.

Both variants have one disadvantage in common; they exclude examples that are available in the original sample from the stratified sample. An alternative that avoids to discard information in the case of small datasets is to use example weights. An example having a weight of w represents a *set* of w examples. One can simply use the denominators of definition 31 as weights in order to stratify a data set. Intuitively, this is even better suited to reflect the densities defined by D'. The main disadvantages of this approach are that (i) all subsequently applied operators are required to be capable of interpreting such a representation, and (ii) that it increases the risk of overfitting if examples are allowed to have weights greater than 1, because they simply cannot provide the information of more than one example; such weights can e.g. compromise results if a learner computes confidence intervals based on the weighted supports of subsets.

This short discourse already illustrates that even for the most simple non-uniform sub-sampling scenario of stratification there are several alternative operationalizations, each having its own application-dependent weaknesses. Hence, different authors have studied the behavior of several stratification techniques empirically. The sub-sampling strategies sketched above are referred to as *under-sampling the majority class* in the literature. Alternatively, one can sample more examples of the minority class with replacement, which is referred to as *over-sampling*. Chawla et al. (2002) give a broader overview and provide several pointers to the literature. They conclude that over-sampling is generally inferior to under-sampling, but propose a new kind of over-sampling without replacement that constructs examples artificially instead. Their algorithm, called SMOTE (for *synthetic oversampling technique*) works by repeatedly sampling an example e_{min} from the minority class, selecting a nearest neighbor e_{maj} from the majority class, and constructing a new example with the same class as e_{min} at a randomly chosen point in between. They report that this strategy worked better than the traditional stratification techniques for several benchmark datasets, using NAÏVEBAYES, decision trees, and a rule induction algorithm as their test bed. This can be explained by the fact that after constructing the artificial examples the learners favor decision boundaries that keep a certain distance between the supporting positive and negative examples. This is similar to margin maximization, the theoretical foundation of support vector machines (Schölkopf & Smola, 2002).

Stratification is an important part of the novel algorithms that will be presented in chapter 4 and 5. The algorithms can basically be combined with any of the discussed stratification techniques. Regarding the empirical evaluation it is important to note that the publicly available benchmark data sets easily fit into main memory. This does not hold for the data of many real-world applications. For small data sets the information loss caused by sampling is usually severe, while sub-sampling is an inevitable preprocessing step when mining from very large databases. Example weighting is the preferred method in the experiments reported in subsequent chapters, but not recommended for large-scale applications. SMOTE will not be used, because it is based on a heuristic, which makes the evaluation of the studied effects more difficult. Moreover, constructing artifical examples implicitly changes the underlying distribution.

Transformation of learning tasks

An interesting aspect of stratification is, that it allows to transform subgroup discovery confined to the utility function WRAcc into the better supported task of classifier induction. This connection has first been shown by the author in (Scholz, 2005b). It simplifies subgroup discovery with its most popular utility function from a practical point of view, which will be utilized in section 4.5.

The goal when inducing a classifier from data generally is to select a predictive model that separates positive and negative examples with high predictive accuracy. Many algorithms and implementations exist for this purpose (Mitchell, 1997; Witten & Frank, 2000), basically differing in the set of models (hypothesis space $\mathcal{H}$) and search strategies. Subgroup discovery requires to define a property of interest, which we assumed to be present in the form of a target attribute. In this sense, this task is also supervised. The process of model selection is guided by a utility function. The following definition simplifies subgroup discovery a bit; the goal here is to find just a single most interesting rule.

Definition 32 *Let $\mathcal{H}$ denote the set of candidate models (rules) considered by a learning algorithm, and let D denote a probability density function of $\mathcal{X}$. When referring to empirical performances, the uniform distribution $D_{\mathcal{E}}$ of a training set $\mathcal{E}$ is substituted.*
The task of classifier induction *is to find the most accurate model*

$$h^* := \operatorname*{argmax}_{h \in \mathcal{H}} \left(\mathrm{Acc}_D(h) \right).$$

For a given utility function Q evaluating model candidates with respect to D, the task of subgroup discovery *is to identify a model (rule) h^* of highest utility:*

$$h^* := \operatorname*{argmax}_{h \in \mathcal{H}} \left(Q_D(h) \right).$$

If the target attribute is boolean, then common classifier induction algorithms do not benefit from finding rules with a precision below 50%. In contrast, for subgroup discovery it is sufficient if a class is observed with a frequency that is significantly higher than in the overall population. In more technical terms, the precision of the rule needs to be higher than the prior of the predicted class. In cases of skewed class distributions the frequency in the covered subset might still be far below 50% for the most interesting rules. Choosing the utility function WRAcc we can transform subgroup discovery as defined above into classifier induction by means of stratification. Definition 31 allows to state the following theorem.

Theorem 5 *For every rule $A \rightarrow C$ the following equalities hold if D' is a stratified random sample density function of D:*

$$\begin{aligned} \mathrm{Acc}_{D'}(A \rightarrow C) &= 2\mathrm{WRAcc}_{D'}(A \rightarrow C) + 1/2 \\ &= \mathrm{WRAcc}_D(A \rightarrow C) \cdot \underbrace{\frac{1}{2\Pr_D[C] \cdot \Pr_D[\overline{C}]}}_{\text{irrelevant for ranking rules}} + 1/2 . \end{aligned}$$

Proof
Let $\mathcal{H}$ denote a set of classification rules. The antecedent of a rule candidate under consideration is denoted as A, its head as C. Substituting the definitions of the metrics, the two optimization problems are:

Classification Find an $(A \to C) \in \mathcal{H}$ that maximizes *predictive accuracy*:

$$\text{ACC}(A \to C) = \Pr[A, C] + \Pr\left[\overline{A}, \overline{C}\right]$$

Subgroup Discovery with WRAcc Find an $(A \to C) \in \mathcal{H}$ that maximizes

$$\text{WRACC}(A \to C) = \Pr[A] \cdot (\Pr[C \mid A] - \Pr[C])$$

The correctness of the theorem is shown based on two lemmas.

Lemma 3 *The two tasks are equivalent, if and only if the priors of both class labels are equal:*

$$\Pr[C] = \Pr\left[\overline{C}\right] = 1/2$$

Proof
First we rewrite predictive accuracy:

$$\begin{aligned}
\text{ACC}(A \to C) &= \Pr[A, C] + \Pr\left[\overline{A}, \overline{C}\right] \\
&= \Pr[A, C] + \left(\Pr\left[\overline{A}\right] - \Pr\left[\overline{A}, C\right]\right) \\
&= \Pr[A, C] + \Pr\left[\overline{A}\right] - (\Pr[C] - \Pr[A, C]) \\
&= 2\Pr[A, C] + \Pr\left[\overline{A}\right] - \Pr[C] \\
&= 2\Pr[C|A]\Pr[A] + 1 - \Pr[A] - \Pr[C] \\
&= 2\Pr[A](\Pr[C|A] - 1/2) + \Pr\left[\overline{C}\right] \qquad (3.10)
\end{aligned}$$

The order of rules according to this metric does not change if we drop the constant additive terms $P(\overline{C})$ and the constant factor of 2 in eqn. (3.10), so

$$\underset{(A \to C) \in \mathcal{H}}{\text{argmax}} \ \text{ACC}(A \to C) = \underset{(A \to C) \in \mathcal{H}}{\text{argmax}} \ (\Pr[A] \cdot (\Pr[C \mid A] - 1/2)) .$$

Obviously, the second term is equivalent to WRACC, if and only if $\Pr[C] = 1/2$. In this case ACC and WRACC induce the same ranking of rules. □

Lemma 3 basically just reflects a previously (p. 27) mentioned property of WRACC formally: The geometries of ROC isometrics are identical for ACC and WRACC if the data is stratified.

If the condition of lemma 3 is violated for the density function D originally underlying the data, then we can perform stratified sampling as captured by definition 31:

$$D'(x, y) := \frac{D(x, y)}{|\mathcal{Y}| \cdot \Pr_{(x', y') \sim D}(y = y')} \qquad (3.11)$$

Considering a sample from D' as defined by (3.11) we expect $\Pr_{D'}[A]$ and $\Pr_{D'}[C|A]$ to differ from $\Pr_D[A]$ and $\Pr_D[C|A]$, respectively. As the following lemma states, such samples are nevertheless appropriate for rule selection.

Lemma 4 *The preference ordering of a rule set $\mathcal{H}$ induced by the* WRACC *metric is equivalent for two probability density functions* D *and* D' *if eqn. (3.11) holds.*

Proof
Let us first exploit eqn. (3.11) to rewrite $P_{D'}(A)$ in terms of D:

$$\begin{aligned}
Pr_{D'}[A] &= \frac{Pr_D[A,C]}{2Pr_D[C]} + \frac{Pr_D\left[A,\overline{C}\right]}{2Pr_D\left[\overline{C}\right]} \\
&= \frac{Pr_D[A]}{2}\left(\frac{Pr_D[A,C]}{Pr_D[A]\,Pr_D[C]} + \frac{Pr_D\left[A,\overline{C}\right]}{Pr_D[A]\,Pr_D\left[\overline{C}\right]}\right) \\
&= Pr_D[A]\cdot\underbrace{\frac{1}{2}\left(\text{LIFT}_D(A\to C)+\text{LIFT}_D(A\to\overline{C})\right)}_{=:\alpha} \qquad (3.12)
\end{aligned}$$

Having $Pr_{D'}[A] = Pr_D[A]\cdot\alpha$ allows to reformulate $\text{WRACC}_{D'}$:

$$\begin{aligned}
&\text{WRACC}_{D'}(A\to C) \\
=\ & Pr_{D'}[A]\cdot(Pr_{D'}[C|A] - Pr_{D'}[C]) \\
=\ & Pr_{D'}[A]\cdot\left(\frac{Pr_{D'}[A,C]}{Pr_{D'}[A]} - 1/2\right) \\
=\ & Pr_D[A]\cdot\alpha\cdot\left(\frac{\frac{Pr_D[A,C]}{2Pr_D[C]}}{Pr_D[A]\cdot\alpha} - 1/2\right) \\
=\ & Pr_D[A]\cdot\alpha\cdot\left(\frac{1}{2}\frac{Pr_D[A,C]}{Pr_D[A]\cdot Pr_D[C]\cdot\alpha} - 1/2\right) \\
=\ & \frac{1}{2}Pr_D[A]\,(\text{LIFT}_D(A\to C)-\alpha) \qquad (3.13)
\end{aligned}$$

Formula (3.13) can be simplified by rewriting α, exploiting that

$$\begin{aligned}
\text{LIFT}_D(A\to\overline{C}) &= \frac{Pr_D\left[\overline{C}|A\right]}{Pr_D\left[\overline{C}\right]} = \frac{1-Pr_D[C|A]}{Pr_D\left[\overline{C}\right]} \\
&= \frac{1}{Pr_D\left[\overline{C}\right]} - \frac{Pr_D[C]}{Pr_D\left[\overline{C}\right]}\cdot\text{LIFT}_D(A\to C) \qquad (3.14)
\end{aligned}$$

After plugging (3.14) into α we have

$$\begin{aligned}
\alpha &= 1/2\cdot\left(\text{LIFT}_D(A\to C) + \frac{1}{Pr_D\left[\overline{C}\right]} - \frac{Pr_D[C]}{Pr_D\left[\overline{C}\right]}\cdot\text{LIFT}_D(A\to C)\right) \\
&= 1/2\cdot\left(\left(1-\frac{Pr_D[C]}{Pr_D\left[\overline{C}\right]}\right)\text{LIFT}_D(A\to C) + \frac{1}{Pr_D\left[\overline{C}\right]}\right) \\
&= \frac{1}{2Pr_D\left[\overline{C}\right]}\cdot\left(\left(Pr_D\left[\overline{C}\right]-Pr_D[C]\right)\text{LIFT}_D(A\to C)+1\right) \\
&= \frac{1}{2Pr_D\left[\overline{C}\right]}\cdot((1-2Pr_D[C])\,\text{LIFT}_D(A\to C)+1)
\end{aligned}$$

which can now be substituted into (3.13):

$$\begin{aligned}
&\frac{1}{2}Pr_D[A]\cdot(\text{LIFT}_D(A\to C)-\alpha) \\
=\ & \frac{1}{2}Pr_D[A]\cdot\left(\text{LIFT}_D(A\to C) - \frac{(1-2Pr_D[C])\,\text{LIFT}_D(A\to C)+1}{2Pr_D\left[\overline{C}\right]}\right)
\end{aligned}$$

$$
\begin{aligned}
&= \frac{1}{2}\mathrm{Pr}_D[A] \cdot \left(\mathrm{LIFT}_D(A \to C) \left(1 - \frac{1 - 2\mathrm{Pr}_D[C]}{2 - 2\mathrm{Pr}_D[C]}\right) - \frac{1}{2\mathrm{Pr}_D[\overline{C}]} \right) \\
&= \frac{1}{2}\mathrm{Pr}_D[A] \cdot \left(\mathrm{LIFT}_D(A \to C) \frac{1}{2 - 2\mathrm{Pr}_D[C]} - \frac{1}{2\mathrm{Pr}_D[\overline{C}]} \right) \\
&= \frac{1}{4\mathrm{Pr}_D[\overline{C}]} \cdot \mathrm{Pr}_D[A] \cdot (\mathrm{LIFT}_D(A \to C) - 1) \\
&= \frac{1}{4\mathrm{Pr}_D[\overline{C}] \cdot \mathrm{Pr}_D[C]} \cdot \mathrm{Pr}_D[A] \cdot (\mathrm{Pr}_D[C|A] - \mathrm{Pr}_D[C]) \\
&= \underbrace{\frac{1}{4\mathrm{Pr}_D[\overline{C}] \cdot \mathrm{Pr}_D[C]}}_{\text{irrelevant}} \cdot \mathrm{WRACC}_D(A \to C) \qquad (3.15)
\end{aligned}
$$

The constant factor on the left does not change the ranking of rule sets. We may drop it and end up with the definition of the WRAcc metric for D, which completes the proof of lemma 4. □

Combining eqn. (3.15) and eqn. (3.10) yields

$$
\begin{aligned}
\mathrm{ACC}_{D'}(A \to C) &= 2\mathrm{Pr}_{D'}[A](\mathrm{Pr}_{D'}[C|A] - 1/2) + \mathrm{Pr}_{D'}[\overline{C}] \\
&= 2\mathrm{Pr}_{D'}[A](\mathrm{Pr}_{D'}[C|A] - \mathrm{Pr}_{D'}[C]) + 1/2 \\
&= 2\mathrm{WRACC}_{D'}(A \to C) + 1/2 \\
&= \frac{1}{2\mathrm{Pr}_D[\overline{C}] \cdot \mathrm{Pr}_D[C]} \cdot \mathrm{WRACC}_D(A \to C) + 1/2,
\end{aligned}
$$

which proves theorem 5. □

As a consequence of theorem 5, subgroup discovery tasks with utility function WRAcc can as well be solved by rule induction algorithms that optimize predictive accuracy after stratified resampling, or after reweighting, respectively (cf. section 3.4.2). In any case, the induced ranking of rules will be identical to that induced by WRAcc. The close connection between Acc and WRAcc provides an intuitive explanation for lemma 2 (p. 53), i.e. that WRAcc is an instance-averaging function and shares the confidence bounds of Acc in adaptive sampling scenarios. The simplification of WRAcc will turn out useful in the next chapter for deriving a simple sequential subgroup discovery algorithm based on common Acc-maximizing rule discovery algorithms.

Lemma 2 also provides an explanation for a property mentioned briefly on page 57, that stratification helps to optimize the area under the ROC curve, whenever the chosen learners optimize predictive accuracy. The reason is, that maximizing accuracy after stratification is identical to directly maximizing the WRAcc. The latter, in turn, can easily be shown to be identical to AUC optimization in the specific case of a binary classifier. This is illustrated in Fig. 3.1. The isometrics of the WRAcc metric are parallel lines to the main diagonal. A classifier that maximizes this metric, exemplarily depicted by a green dot in the figure, will hence be as far as possible from the diagonal. The AUC can be decomposed into the area below the diagonal and the enclosed triangle above, connecting the green point with $(0, 0)$ and (P, N). Clearly, the area of the triangle is half the area of the shaded rectangle for all points that share the same isometric line. Since the area of the rectangle grows monotonically as the isometric line approaches the optimal point of $(0, P)$, the AUC grows strictly monotonically with an increasing WRAcc.

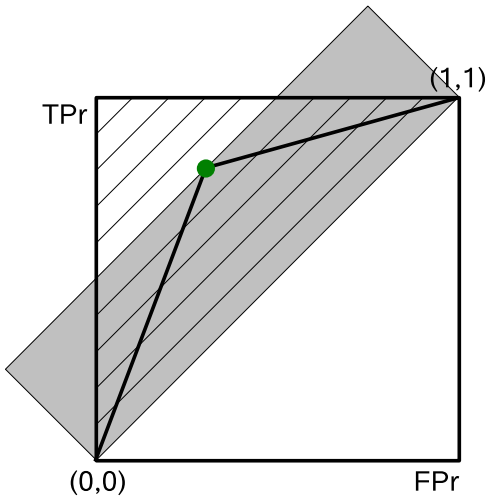


Figure 3.1.: The green point shows a classifier performance in ROC space. WRAcc isometrics are lines parallel to the main diagonal. The area of the enclosed rectangle above the main diagonal, and hence the AUC, only depend on the WRAcc of the classifier.

This connection can easily be exploited for greedily maximizing the AUC with decision tree soft classifiers that use only binary splits. Each leaf covers a set of examples for which the same soft prediction is made. After a step of stratification the utility of accuracy maximizing splits grow monotonically with the WRAcc, which means that the learner tries to "shift" examples as far as possible to the point $(0, P)$ in ROC space with each split. If each subset is stratified before selecting the splitting criterion, then splits based on accuracy maximization will implicitly favor AUC maximizing trees. In other words: Recursively stratifying (approximately) transforms AUC maximization into WRAcc maximization in the context of binary decision trees.

The task gets more complex in the general case, which also covers non-binary splits. The Smiles algorithm by Ferri et al. (2002) fits AUC maximizing trees by computing and comparing the resulting AUC scores of split candidates. The authors report that, averaged over several benchmark datasets, AUC maximization also led to better classification performances than classical impurity criteria.

There is another interesting result that can be derived from lemma 3. Accuracies under different misclassification costs or class skews can be maximized based on stratification. The only difference between WRAcc and Acc maximization is that the slope of the isometrics is 1 for the former in ROC space, while the slope of the latter is N/P. After a step of stratification that yields new values P' and N', the optimization of accuracy is based on the new slope of isometrics, $N'/P' = 1$. Hence, the *only* effect of stratification for accuracy optimization problems is that it in fact changes the slope of the isometrics. This property holds regardless of the target proportions. We can as well *introduce* class skews by changing the ratio of N'/P', using the same stratification techniques with different pre-defined target class proportions. This means that any target slopes, reflecting asymmetric misclassification costs or skews, may be used to define a new target class ratio. That way we can train a classifier with any common accuracy-maximizing learner for any user-given slope in ROC space. This idea is further illustrated in subsection 5.3.5. From a technical point of view, the corresponding sampling strategy can be subsumed under the notion of rejection sampling, a technique with a broader scope, which is discussed in the next subsection.

3.4.2. Rejection Sampling

The overall purpose of Monte Carlo sampling techniques is to sample from complex distributions. This is operationalized by internal random experiments. For stratification it depends on details of the analytical setting whether internal random experiments are beneficial. Otherwise, as discussed in the last subsection, it can be addressed by simpler techniques, which one would not necessarily subsume under the notion of Monte Carlo sampling.

The Monte Carlo method of rejection sampling is considerably more general. It allows to sample exactly from almost arbitrarily complex target distributions if some comparably weak assumptions are met. It is a useful technique for very different problems, but the way it is used in this thesis, and in the contexts of primary interest, is different from the typical use in the statistical literature (Mackay, 1998; Neal, 1993). The first part of this subsection discusses the strategy in general, its assumptions, typical application scenarios, possible shortcomings and more sophisticated strategies. The second part reviews some applications in the context of supervised learning. Rejection sampling is a theoretical foundation of the knowledge-based sampling technique, which will be presented in chapter 4.

Sampling from complex distributions

Assume that we want to draw samples $x \in \mathcal{X}$ with respect to a complex density function $P : \mathcal{X} \to \mathbb{R}^+$. If we only have random bits to operationalize this procedure, then this is a hard challenge for many functions. Please note that P may have very unpleasant properties, e.g., it might be high-dimensional and non-continuous, have irregular dependencies between different components of $\mathcal{X}$ etc. We only assume that we can compute $P(x)$ for each $x \in \mathcal{X}$. Now assume that we further know a much simpler density function $Q : \mathcal{X} \to \mathbb{R}^+$, for which we have an efficient sampling procedure and another procedure to compute $Q(x)$ for each $x \in \mathcal{X}$. If there is a constant $c \in \mathbb{R}^+$ with the property that

$$(\forall x \in \mathcal{X}) : c \cdot Q(x) \geq P(x),$$

then rejection sampling provides the means to sample from $P(x)$. The idea is to combine two random experiments. The first one is to sample an example $x \sim cQ$, which is not sufficient, of course, because the two functions will deviate for most subsets of $\mathcal{X}$. Hence, the second random experiment is designed to correct the errors of the first one by rejecting examples "proportionally to the error". Therefore, another internal Bernoulli random experiment with $p = P(x)/(c \cdot Q(x))$ is performed. If this random experiment "succeeds" (probability is p) the example is accepted, otherwise it is rejected and the procedure is repeated. Both random experiments are independent, so effectively the resulting density $D(x)$, $x \in \mathcal{X}$, defined by the procedure is

$$D(x) = c \cdot Q(x) \cdot \frac{P(x)}{cQ(x)} = P(x),$$

which proves the correctness of the approach. The procedure is illustrated in Fig. 3.2[6]. The red curve is the target density function, the green curve shows a normal distribution, where the scaling constant c is chosen large enough. A Bernoulli experiment after sampling $x = -2$ from cQ is depicted as a vertical line, where the intervals $[0, P(x)]$ and $[P(x), Q(x)]$ are proportional to the probability of acceptance and rejection, respectively. The efficient sampling procedure samples proportionally to the area under the red curve, while the internal experiment can be

[6]The picture and additional material on different Monte Carlo sampling techniques can be found at `http://www.inference.phy.cam.ac.uk/itprnn/code/mcmc/`

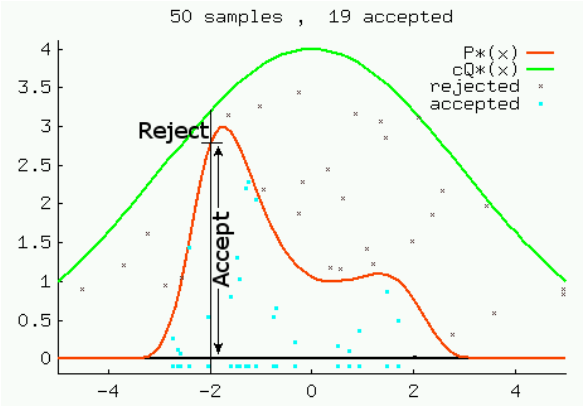


Figure 3.2.: A schematic example of rejection sampling: It is easy to sample from $Q(x)$, but hard to sample directly from $P(x)$. Since $c \cdot Q(x) > P(x)$ the procedure can simply sample from $Q(x)$ and accept x with probability $P(x)/(c \cdot Q(x))$. The points lying below the red line are accepted, the points above are rejected.

considered to uniformly add a second vertical dimension to each point, reaching up to the red line. Only the points below the red line are accepted. As the figure illustrates, this may lead to much smaller samples if P and Q are not similar, that is, if c is large.

This gives rise to more complex procedures, subsumed under the name of Markov Chain Monte Carlo (MCMC) algorithms. Such algorithms construct chains of examples $(x^{(0)}, x^{(1)}, \ldots)$ based on internal random experiments. The chains converge to the true distribution P. Such methods are applicable to a broad variety of problems, ranging from (i) sampling from a complex distribution, over (ii) estimating the expected value of a response for a given functional dependency between response and random variable $\mathcal{X}$, up to (iii) estimating posteriors in Bayesian settings, e.g., to compute the posterior distributions for unobserved variables of a given Bayesian network model (Pearl, 1991). None of the more complex MCMC techniques are applied in this thesis. The interested reader may want to refer to (Mackay, 1998) for a compact introduction and to (Neal, 1993) for a broader discussion with a focus on probabilistic inference.

Applications in the context of supervised learning

The example applications of rejection sampling discussed so far have in common that it is possible to *construct* sample points arbitrarily, e.g., in a space $\mathcal{X}$ like $\mathbb{R}^n, n \in \mathbb{N}$, because the responses (if any) can easily be computed for each $x \in \mathcal{X}$. For supervised data mining applications the situation is different. The functional or probabilistic dependency between $\mathcal{X}$ and $\mathcal{Y}$ is not known a priori, but it is the goal of a learner to find a good approximation of this dependency by selecting an appropriate model from a given class. In this setting, we can usually just *select* a subset of the instances from the training set, while the construction of new instances is not possible. Approaches like SMOTE (p. 58) are an exception to this rule, but technically they sample from yet another distribution, need to apply heuristics, and hence may fail to improve the performance for several combinations of learner and dataset.

Please recall, that we consider the full example set $\mathcal{E}$ itself to be an i.i.d. sample from a fixed pdf D. Using the previously discussed technique we can sub-sample from a different pdf D' now, that is defined with respect to D. Translated to the previously sketched rejection sampling notation we have $D = Q$ and want to sample from $D' = P$. Hence, the first kind of random experiments that are part of the rejection sampling procedure, namely sampling from Q, has

already been performed in advance. Now we need to find a constant c, as small as possible, while meeting the constraint $cQ(x) > P(x)$ for all $x \in \mathcal{E}$, and we need the ratios $P(x)/(cQ(x))$ alias $D'(x)/(cD(x))$. We can then select a subset of $\mathcal{E}$ using rejection sampling by performing a Bernoulli experiment based on these ratios. Clearly, unless $c = 1$ the total sample size will decrease in this case. Unlike for other typical applications of rejection sampling, like sampling from posterior distributions, this does not result in a waste of time in the first place, but rather in a waste of information. For this reason rejection sampling should only be considered in cases where the amount of available training data is large, and where the rejection rate reflected by the constant c is reasonably low. In other cases one can use example weights instead. The problems and advantages are similar to those discussed for stratification. In any case, a sound understanding of the principles underlying rejection sampling fosters a better understanding of the semantics of example weights; the weight of each example $x \in \mathcal{E}$ can simply be interpreted as the ratio $D'(x)/D(x)$, the acceptance ratio scaled by a normalization constant c. In several cases referring to changes of the underlying distributions, as operationalized by rejection sampling, eases the analysis of data mining algorithms. There are several approaches in the literature that explicitly refer to this kind of transformation. Three of them are exemplarily discussed in the next paragraphs; they are all related to this thesis.

Cost sensitive learning In section 2.5 and subsection 3.4.1 the incorporation of misclassification costs into learning has been shown to be possible by sampling or weighting, even if the used machine learning algorithm is not capable of incorporating costs. However, a required underlying assumption was that misclassification costs only depend on the true label of an example. Zadrozny et al. (2003) extended this framework, assuming that for each example a specific attribute contains the (positive) costs that are caused by misclassifying the example. They apply rejection sampling to provide an appropriately altered sample to the learner. Each example from the original training set is sampled proportionately to the associated costs. The acceptance rate for an example with misclassification costs (loss) l is hence l/l_{max}, where l_{max} denotes the highest costs observed for any example in the training set. That way, the learning algorithm selects a model minimizing costs without being "aware" of the cost-sensitive component, just as discussed for the simpler cost-sensitive classification framework in subsection 3.4.1. As an interesting theoretical result the authors provide a proof that in the agnostic PAC-learning framework (see subsection 2.3.3) the asymptotic sample complexity does *not decrease* in combination with cost proportional rejection sampling, although this sampling technique reduces the number of examples that are available for training. They take advantage of the gain in runtime (caused by the sample reduction) by training several base classifiers and combining them in a bagging-like fashion (cf. subsection 2.6.1). This approach led to convincing performances in an empirical study. Cost proportional sampling can be combined with any of the sampling strategies discussed in subsequent parts of this thesis.

Sample selection bias A crucial problem, especially in descriptive learning scenarios, is that the process of collecting the data is usually biased. In other words, the example set is not a sample from an underlying pdf D we would like to learn about, but a sample from a biased pdf that changes the class ratio, systematically puts higher weight on some subsets of the data, or even changes the probabilistic dependencies between $\mathcal{X}$ and $\mathcal{Y}$. These effects can be modeled by introducing an additional random variable $\mathcal{S} \in \{0, 1\}$ that determines whether an example sampled from the "true" distribution is included in the example set or not (Zadrozny, 2004; Fan et al., 2005). This means to effectively sample from $D : \mathcal{X} \times \mathcal{Y} \times \mathcal{S} \to \mathbb{R}^+$, where the goal is to analyze the marginal distribution over

$\mathcal{X} \times \mathcal{Y}$, but the presented example set contains only the subset with $\mathcal{S} = 1$. The case in which $\mathcal{S}$ is independent of $\mathcal{X}$ and $\mathcal{Y}$ is the unbiased case. The case where $\mathcal{S}$ only depends on $\mathcal{Y}$ has been discussed before, and can be addressed by stratifying the resulting example set. Without independence of any kind, the learning task becomes intractable. Rejection sampling can be used to address the last case, in which $\mathcal{S}$ depends on $\mathcal{X}$ and is independent of $\mathcal{Y}$ given $\mathcal{X}$. This holds e.g., for labels that functionally depend on the vector $x \in \mathcal{X}$, but approximately also in many other practically relevant settings. The following pdf D' corrects the bias of D:

$$D'(\mathcal{X} = x, \mathcal{Y} = y, \mathcal{S} = s) \propto \Pr(\mathcal{S} = 1) \cdot \frac{D(\mathcal{X} = x, \mathcal{Y} = y, \mathcal{S} = s)}{\Pr(\mathcal{S} = 1 \mid \mathcal{X} = x)}.$$

By construction, when conditioning on $\mathcal{S} = 1$ the random variables $\mathcal{X}$ (and $\mathcal{Y}$) are distributed under D' as the marginal distributions under D, that is, without conditioning; this is achieved by dividing the density of each $x \in \mathcal{X}$ by the conditional probability $\Pr(\mathcal{S} = 1 \mid \mathcal{X} = x)$ of showing x if it is sampled. As a result, the conditional probability to show a sampled example is identical for each $x \in \mathcal{X}$, which yields an unbiased sample. $\Pr(\mathcal{S} = 1)$ is an irrelevant normalization constant.

In settings where it is possible to get estimates of $\Pr(\mathcal{S} = 1 \mid \mathcal{X} = x)$, rejection sampling can be applied in a straightforward manner. Zadrozny (2004) shows that the expected loss for any hypothesis under consideration is identical when (i) sampling from the marginal distribution over $\mathcal{X} \times \mathcal{Y}$ and when (ii) sampling examples with a domain of $\mathcal{X} \times \mathcal{Y} \times \mathcal{S}$, discarding all example with $\mathcal{S} = 0$ and subsequently performing a step of rejection sampling to correct the sample selection bias.

Getting estimates for $\Pr(\mathcal{S} = 1 \mid \mathcal{X} = x)$ is not unrealistic. For example, in a poll it is possible to compare several quantities observed in the overall population to the subset of people that actually responded. Analogously, if in medical datasets specific kinds of values from a laboratory are only available for subsets of patients then the characteristics of such subsets can be compared to the characteristics of the set of all patients, in order to estimate the bias. As for cost-proportionate sampling, changing the pdf underlying the data to overcome known biases can be combined with any of the subsequently discussed sampling strategies.

Boosting Changing the distribution underlying a training set was the key idea of the first boosting algorithm. Although this kind of boosting proposed by Schapire (1990) has only limited practical relevance, the proof of its correctness settled the open question of whether weak and strong learnability define different classes of learning problems in the PAC framework or not (cf. section 2.3).

The core of the algorithm induces three weak learners that only need to perform slightly better than random guessing (most of the time). The reliability of the weak base learner, reflected by the parameter δ in the PAC framework, can easily be increased by repeating the learner several times and performing uniform voting (cf. subsection 2.6.1). With a linear number of repetitions the probability of failure δ decreases exponentially fast. The critical property is hence the error rate ϵ, while we can assume a moderate failure rate δ, if necessary as the result of simply running the base learner repeatedly.

The first of the three learners mentioned above is applied to a training set sampled from the original distribution. The distribution is then changed for the second learner, so that one out of two examples becomes misclassified by the first learner. This change of the

distribution is operationalized by rejection sampling. When applying the first model to the new distribution the expected error rate is $1/2$. Weak PAC learnability implies that the base learner yields (with probability $1-\delta$) a model which is at least slightly better than random guessing for *any* distribution. For this reason the second model will capture dependencies between the target variable $\mathcal{Y}$ and the instance space $\mathcal{X}$ that were not captured by the first model. For the third classifier all examples are rejected for which the first two models agree. Intuitively, it is used to break ties. The final model is established on top of the three base model predictions in terms of a uniform voting scheme.

It was proved by Schapire (1990) that the algorithm sketched above reduces the expected error rate sufficiently well. It is possible to apply this algorithm recursively, substituting the base model by combinations of three models or three aggregated models in each step. Omitting the involved technical details, the main implication is that by combining weak learners to an ensemble one can reach at an efficient (polynomial runtime) arbitrarily strong learner satisfying the ϵ/δ-criterion of PAC learning (summarized by Def. 3, p. 14). Hence, the concepts of weak and strong learnability are identical. In later work the practically more efficient boosting algorithm ADABOOST has been proposed (Freund & Schapire, 1997), which is discussed in chapter 5. It changes the underlying distribution in a similar fashion as the discussed algorithm does before calling the second learner. ADABOOST is usually applied in combination with example weighting, but it can as well be used in combination with rejection sampling.

3.5. Summary

This chapter illustrated the variety of possible applications for sampling techniques in the context of KDD applications. The most obvious benefit of sampling is, that it allows to speed up the learning process. Even if considering only a small fraction of the data, for many learning tasks statistical estimates allow to give guarantees that models induced by learning techniques are close to the corresponding models trained from all the available data. The gain depends on the specific data and learning task at hand; for subgroup discovery the specific choice of a utility function has a serious impact on sample complexity and confidence bounds, but for most practically relevant utility functions adaptive sampling allows to reduce the complexity of discovery tasks significantly.

Sampling has been shown to be a powerful preprocessing operator; it is possible to transform learning problems just by applying specific sampling techniques, as illustrated for subgroup discovery with the WRACC utility function, which can be solved by common rule induction algorithms after a step of stratification. Similarly, AUC maximization in the context of decision tree induction can to a large extent be reduced to WRACC maximization. Other data mining problems that can be handled by sampling rather than by adjusting learning algorithms contain the incorporation of misclassification costs, corrections of sample selection biases, the reduction of model variance, and the increase of ensemble accuracy. Even if not mentioned explicitly, any of the above-mentioned and subsequently presented sampling techniques can be combined arbitrarily.

The next chapter will discusses how to incorporate prior knowledge into supervised learning by means of sampling.

4. Knowledge-based Sampling for Sequential Subgroup Discovery

4.1. Introduction

The goal of knowledge discovery in databases (KDD) can be stated as to identify useful and novel patterns hidden in huge amounts of real-world data. Commonly recognized problems when applying existing data mining techniques are that these techniques either yield an unmanageable number of patterns, e.g. frequent itemsets, or that they report only "obvious" patterns, that are already known to domain experts.

In this thesis it is assumed, that each pattern is interesting to the degree to which it deviates from expectation (Guyon et al., 1996). This chapter addresses the discovery of patterns that can be represented by probabilistic rules. This leads to a setting similar to subgroup discovery (cf. subsection 2.2.3), for which the interestingness of rules is measured in terms of utility functions. The goal of subgroup discovery is to identify subsets of the population that show an unusually high (or low) frequency of a specified property of interest. This kind of task naturally emerges in very different domains, ranging from medical applications over marketing to spatial data mining (Lavrac et al., 2004a; Atzmüller et al., 2005; Klösgen & May, 2002).

In this chapter a technique is presented that helps to guide the search for interesting patterns in the presence of prior knowledge without any need to adapt underlying data mining algorithms. The main ideas published in the literature on subgroup discovery are adopted, but extended to respecting prior knowledge. This novel technique naturally allows to search for patterns iteratively. For a given target variable the absence of any pattern is assumed to be equivalent to its independence of all the other variables. In the presence of prior knowledge, however, absence of further patterns is given if the prior knowledge precisely models the conditional distribution of the target variable. In turn, a pattern may be defined as a regular deviation from the assumed independence, or from a given prior model, respectively. It can be observed as a correlation between the given target attribute and other variables. The most obvious patterns may either be elicited from domain experts beforehand, or they may be the result of a first application of data mining tools. The main objective in the proposed iterative data mining framework is to preprocess the data in such a way that subsequent steps do not report previously found patterns again, but automatically focus on interesting, uncorrelated, and novel patterns.

As a running toy example, the task of characterizing groups of car drivers with an unusually high risk of accidents is considered in this chapter. Assuming that the prior of this risk is about 1% per year, sufficiently large subgroups having a risk of 5% might be interesting to insurance companies. In descriptive settings, the prior of a class is simply the fraction of corresponding examples in the complete data set.

The example illustrates a major difference to classifier induction; the subset above represents an interesting pattern in terms of subgroup discovery, but not in terms of classification, because in the latter only rules predicting the mode are useful. The mode of both the overall population and the subset is "no accident", so knowing about the subgroup does not help to make a crisp prediction. In turn, common rule induction algorithms do not discover subgroups for which the

corresponding rule has a low precision. Nevertheless, the revealed close connection between subgroup discovery and common rule induction for predictive tasks (cf. theorem 5, p. 59) will allow to remove this obstacle.

This chapter is organized as follows: In order to motivate the novel approach, some drawbacks of existing work on subgroup discovery are discussed in section 4.2. The main contribution of this chapter is a generic sampling technique that makes subgroup discovery techniques sensitive to prior knowledge. It is presented in section 4.3 and operationalized in terms of a rejection sampling technique in section 4.4. In section 4.5, an algorithm for *sequential* subgroup discovery is derived, which is empirically evaluated in section 4.6. The utility of knowledge-based sampling for local pattern mining is briefly outlined in section 4.7. Section 4.8 summarizes the results presented in this chapter.

4.2. Motivation to extend subgroup discovery

Subgroup discovery (cf. subsection 2.2.3, p. 10) is a supervised learning task that aims at finding *interesting* subsets of an example set. Interestingness is usually defined in terms of coverage and deviations of class distributions from those in the overall population (bias). When considering the resulting example set constructed by the subsequently presented knowledge-based sampling, the building blocks of utility functions, coverage and bias, are adapted in an intuitive way: The coverage of rule candidates remain. At the same time, the biases of rule candidates are turned into *relative biases*, that is, into deviations from predictions derived from prior knowledge, rather than deviations from the class priors. Please note that the class priors provide a first kind of prior knowledge for subgroup discovery as well. The effect of knowledge-based sampling is, that subsequently applied subgroup discovery algorithms prefer rules that explain deviations of empirical from expected class distributions.

It is natural to consider previously discovered rules as prior knowledge, as well. This interpretation will help to overcome short-comings of classical approaches, which are discussed in the next paragraphs.

Drawbacks of classical approaches

The first drawback of subgroup discovery, a lack of expressiveness, shows e.g., when rules have interesting exceptions. These exceptions are hardly found by standard techniques, for mainly two reasons:

First, due to the syntactical structure imposed by classification rules (Horn logics) it is often hard to exclude exceptions from rules, even if this would improve the score assigned by a utility function. The syntactical form is still reasonable, however, because results are required to be interpretable, and because it is the main reason for diversity within the k best subgroups. Without any strong syntactical restrictions the second best subgroup would usually be the best one after adding or removing a single example. This syntactical restriction alone is not sufficient to avoid results containing many similar rules, however. Redundancy filters are a common technique to overcome this problem (Klösgen, 1996). Overlapping patterns like exceptions to rules are not found reliably that way. However, exceptions could still be represented by separate rules. This fails for the second reason, namely that utility functions evaluate rules globally. Interactions between rules do not affect their scores.

For the task of finding exception rules, some efficient algorithms have been proposed by Suzuki (2004). They focus on mining *pairs* of rules, one strong rule and a corresponding exception. This setting is very specific, however; subgroups do not necessarily have exceptions,

and they may overlap in arbitrary ways.

Lavrac et al. (2004b) suggested to use a ROC-based strategy as described by Fawcett (2001) (cf. subsection 2.5.2) for pruning rule sets and combining them to an ensemble classifier. Referring to their false positive and false negative rates, all rules are plotted in ROC space. Only rules lying on the convex hull are deemed relevant. Turning these rules into a single classifier by weighted majority vote has a major drawback, however; one of two rules covering disjoint subsets and having almost the same performance are systematically discarded. As soon as one of these rules is superior in both true positive and false negative rates the other rule is considered to be redundant. This is not desirable in descriptive scenarios, as the only rule covering a specific subset of the instance space should not easily be discarded, nor for predictive settings, as diversity is crucial for achieving high predictive accuracy (cf. subsection 2.6.1).

Incorporation of prior knowledge

A way to improve the interestingness and diversity of rule sets is to make use of previously found patterns and formalized prior knowledge during the selection procedure. Incorporating prior knowledge into existing data mining techniques, e.g., in the form of a Bayesian network (Pearl, 1991), is an active field of research. Popular classifier induction and regression algorithms like C4.5 (Quinlan, 1993), the SUPPORT VECTOR MACHINE (SVM) (Burges, 1998), NEURAL NETS, and NAÏVEBAYES (Mitchell, 1997) are only capable of exploiting domain and prior knowledge if the user encodes it into the representation of the examples or chooses the hypothesis space appropriately. An example of the latter is the choice of a polynomial SVM kernel for known feature interaction. Some Inductive Logic Programming algorithms like PROGOL (Muggleton, 1995) are able to incorporate background knowledge given in the form of restricted first order logic formulae directly. However, these algorithms are often designed to find rules matching the given data exactly, which is usually too strict for real world problems. Some approaches like (Wu & Srihari, 2004; Schapire et al., 2002) utilize prior knowledge to compensate for a lack of data. In these scenarios, the models are fitted to both the prior knowledge and the data at hand. In contrast, the goal of subgroup discovery is to find rules that contradict expectation, as this is assumed to indicate interestingness.

As a first consequence, subgroups should be identified by their extensions rather than by their syntactical descriptions. This makes a substantial difference, because many databases allow to identify the same or similar sub-populations using correlated attributes, but these correlations are often not subject to the data mining step. For example, the subgroup of young drivers is almost identical to the subgroup of people that recently acquired their driver license. If the former subgroup is known to have a high risk of accidents, then the same can be expected for the latter. Hence, there is no need to report both rules. Classical subgroup discovery algorithms systematically report rules with similar extensions, because the k best rules just need to be *syntactically* different. In contrast, we usually expect *sets* of rules to be more interesting if they are *extensionally* different; it is desirable that each rule captures a different pattern observable in the data.

Subgroup patterns may even be interesting *only* relative to prior knowledge, as illustrated by the following example:

$$\Pr[C \mid A] = \Pr[C] = 0.5 \quad \text{for a rule} \quad A \rightarrow C.$$

Class C is distributed in A just as in the overall population, so this rule would not be deemed interesting by any reasonable utility function. Now, assume that in the prior knowledge there is

a statement about a superset of A:

$$A' \rightarrow C \;\; [0.9] \;\; \text{with} \;\; \mathtt{Ext}(A) \subset \mathtt{Ext}(A').$$

This rule predicts a higher conditional probability of C given A'. Considering this context, the rule $(A \rightarrow C)$ becomes interesting as an exception to the prior knowledge, because one would rather expect $\Pr[C \mid A] \approx \Pr[C \mid A']$.

In the framework proposed in this chapter, any available information that allows to compute estimates of the user's expectation may help to refine the metric for selecting interesting rules. A similar idea has recently (but independently) been proposed in the scope of frequent itemset mining (Jaroszewicz & Simovici, 2004).

To the best of the author's knowledge, the only approach towards incorporating available knowledge into subgroup discovery reported in the literature prior to the work presented in this thesis (going back to 2004), is the ILP system RSD by Lavrac et al. (2002b; 2006). It uses background knowledge exclusively to propositionalize multi-relational data, a step which is out of the scope of this work. For the learning step itself CN2-SD is used.

An approach recently published by Atzmüller et al. (2005) uses background knowledge in the form of constraints to narrow down the search space, information on the estimated importance of attributes for focusing a beam search, and a discrete abnormality score for attribute values. Large parts of what the authors refer to as incorporating background knowledge is considered to be part of the preprocessing phase in this thesis, following the CRISP-DM model (cf. section 8.1). This contains refinements of the data representation, for example by constructing features or by discretizing them to reflect semantically meaningful categories. Other parts of the background knowledge are used by Atzmüller et al. (2005) to restrict and focus the search, in order to improve the results of the heuristic learner by focusing on interesting literals, and by excluding well-known or meaningless patterns. Besides, applying data mining is simplified by classical inferences for deriving constraints on the fly that guide the search. Structuring *conceptual domain knowledge* for mining only rules that meet specific type constraints was e.g., already supported by the early machine learning workbench MOBAL (Morik et al., 1993).

The work of Atzmüller et al. (2005) is complementary to the technique presented in this chapter, the knowledge-based sampling approach. The latter allows to implicitly "empower" arbitrary utility functions to prefer rules that are unexpected with respect to prior probabilistic knowledge. This frees users from reformulating constraints until reaching at unexpected patterns. Existing models can be refined by incorporating the data mining results iteratively. This allows to identify small diverse sets of probabilistic rules, each characterizing novel aspects of a property of interest. If prior knowledge is available, then it is utilized appropriately.

4.3. Knowledge-based sampling

After a first rough analysis or a step of formalizing domain knowledge, the input to the data mining step consists of a training set and a set of patterns. The most crucial question in such an iterative data mining framework is how to preprocess the training data, so that subsequent learning steps do not just report the same patterns or parts of the prior knowledge again. In other words, the goal is to find uncorrelated new patterns, so that the resulting rule set is compact, but still allows for a precise characterization of the target attribute. The two example subgroups mentioned in the introduction, one containing all young drivers, and the other containing all persons who recently acquired their driver license, illustrates how a stand-alone evaluation of each rule may result in highly overlapping rule sets. This bears the risk of finding many redundant rules.

The algorithm proposed in this chapter allows to focus on previously undiscovered patterns by means of sampling. Target samples are constructed in a way that does not allow to rediscover the available prior knowledge, because after sampling the target attribute is independent of the available predictions, but any undiscovered pattern remains observable.

The next subsections discuss the theoretical foundation of a generic sampling-based technique that allows to incorporate prior knowledge into subgroup discovery, and basically into supervised data mining algorithms in general. Section 4.3.1 narrows down the choice of a new distribution by introducing a set of constraints. These constraints are shown to define a unique distribution in 4.3.2, which is operationalized in section 4.4. In section 4.5, an algorithm is proposed that sequentially discovers subgroups relative to prior knowledge, and that directly augments the prior knowledge after the discovery of each new subgroup.

4.3.1. Constraints for re-sampling

The notation and formal framework used in this chapter are based on the definitions provided in section 2.1 and 2.2. This contains the notions of example sets $\mathcal{E}$, an instance space $\mathcal{X}$, and probability density functions D underlying $\mathcal{X}$. To simplify formal aspects, $\mathcal{X}$ is assumed to be finite in this section, but all results can easily be generalized to continuous domains.

The idea of removing prior knowledge by means of sampling can well be stated in terms of constraints. Obviously, any procedure for knowledge-based sampling will have to implicitly sample from a different distribution than that originally underlying the data. As discussed in subsection 3.2.1 (p. 42), i.i.d. preserving sampling procedures can be specified by the transformation of the probability density function they operationalize. This specification is the subject of this section, while section 4.4 addresses algorithmic aspects of the transformation.

The original probability density function (pdf) is denoted as D, while the new pdf to sample from is denoted as D' in this chapter. The new pdf should be as close as possible to D, in order not to disturb any of the yet undiscovered patterns. At the same time it should be "orthogonal" to the estimates produced by the available prior knowledge. The goal is that the previously discussed rule selection metrics – when applied to samples from D' – are "blinded" regarding the parts of rules that could already be concluded from prior knowledge. All that should be accounted for is the unexpected component of each candidate rule.

To illustrate the setting, the simplified case of prior knowledge consisting only of a single rule

$$r : A \rightarrow C$$

is considered. The distribution to be constructed should no longer support rule r, so as a first constraint, A and C should be independent events under D':

$$Pr_{D'}[C \mid A] = Pr_{D'}[C] \tag{4.1}$$

If r predicts a higher accident probability for young drivers, for example, then in the constructed sample this subgroup should share the class priors of accidents.

As further constraints, the probabilities of events part of the rule should not change, because it is sufficient to remove their correlation. This means that the class priors and the probability of r being applicable to a randomly drawn instance need to be equal for both distributions:

$$Pr_{D'}[A] = Pr_{D}[A] \tag{4.2}$$

$$Pr_{D'}[C] = Pr_{D}[C] \tag{4.3}$$

For the example rule, the probability of accidents and the probability of seeing a young driver will not change from the original training set to the constructed sample if these constraints are met.

Finally, within each partition sharing the same class and prediction of r the new distribution is defined proportionally to the initial one. The simple reason is that having just r as prior knowledge all instances within one partition are indistinguishable. Changing conditional probabilities within one partition would mean to prefer some instances over others, despite their equivalence with respect to the available prior knowledge. For the boolean rule r these constraints translate into the following equalities:

$$\Pr_{D'}(x \mid A, C) = \Pr_{D}(x \mid A, C) \tag{4.4}$$
$$\Pr_{D'}(x \mid A, \overline{C}) = \Pr_{D}(x \mid A, \overline{C}) \tag{4.5}$$
$$\Pr_{D'}(x \mid \overline{A}, C) = \Pr_{D}(x \mid \overline{A}, C) \tag{4.6}$$
$$\Pr_{D'}(x \mid \overline{A}, \overline{C}) = \Pr_{D}(x \mid \overline{A}, \overline{C}) \tag{4.7}$$

Hence, if the probability of sampling a specific driver halves from the original to the new distribution, then the same will happen to all other drivers sharing both, the property of being young or not, and the property of having had an accident or not. All that changes are the marginal probabilities of the four partitions. Further interesting patterns – even if they overlap with r – are still observable when sampling from D'. For instance, LIFTs of subgroups that are subsets of $\mathrm{Ext}(A)$ and have an even significantly higher or much lower LIFT than rule r are just rescaled proportionally. If e.g., unexperienced persons driving a specific kind of car tend to be involved in accidents even more frequently than young drivers in general, then this more specific rule can be found in a subsequent step. As motivated in section 4.2, various kinds of exceptions to previously found rules as well as patterns overlapping in some other way can be found analogously.

4.3.2. Constructing a new distribution

In subsection 4.3.1 the idea of sampling with respect to an altered distribution function has been proposed. Intuitively, prior knowledge and known patterns are "filtered out". This subsection proves that the proposed constraints (4.1)-(4.7) induce a unique target distribution. The following definition just eases notation.

Definition 33 *The* LIFT of an example $x \in \mathcal{X}$ *for a rule* $(A \to C)$ *is defined as*

$$\mathrm{LIFT}(A \to C, x) := \begin{cases} \mathrm{LIFT}(A \to C), \text{ for } x \in \mathrm{Ext}(A) \cap \mathrm{Ext}(C) \\ \mathrm{LIFT}(A \to \overline{C}), \text{ for } x \in \mathrm{Ext}(A) \cap \mathrm{Ext}(\overline{C}) \\ \mathrm{LIFT}(\overline{A} \to C), \text{ for } x \in \mathrm{Ext}(\overline{A}) \cap \mathrm{Ext}(C) \\ \mathrm{LIFT}(\overline{A} \to \overline{C}), \text{ for } x \in \mathrm{Ext}(\overline{A}) \cap \mathrm{Ext}(\overline{C}) \end{cases}$$

There are at most four different LIFT values associated with each rule.

Theorem 6 *For any initial distribution* D *and given rule* r *the constraints* (4.1)-(4.7) *are equivalent to*

$$\Pr_{x \sim D'}(x) = \Pr_{x \sim D}(x) \cdot (\mathrm{LIFT}_D(r, x))^{-1},$$

so they induce the target distribution $D' : \mathcal{X} \to \mathbb{R}^+$ *uniquely.*

Proof
Let $r : A \rightarrow C$ denote the rule under consideration. The proof is exemplarily shown for the partition $\mathtt{Ext}(A) \cap \mathtt{Ext}(C)$, in which the rule under consideration is both applicable and correct. Assuming that the constraints hold, D' can be rewritten in terms of D and $\textsc{Lift}_D(A \rightarrow C)$:

$$\begin{aligned}
(\forall x \in \mathtt{Ext}(A) \cap \mathtt{Ext}(C)) \quad : \quad & Pr_{D'}(x) = Pr_{D'}(x, A, C) \\
= \quad & Pr_{D'}(x \mid A, C) \cdot Pr_{D'}[A, C] \\
= \quad & Pr_D(x \mid A, C) \cdot Pr_{D'}[A] \cdot Pr_{D'}[C] \\
= \quad & \frac{Pr_D(x, A, C)}{Pr_D[A, C]} \cdot Pr_D[A] \cdot Pr_D[C] \\
= \quad & Pr_D(x) \cdot \frac{Pr_D[A] \cdot Pr_D[C]}{Pr_D[A, C]} \\
= \quad & Pr_D(x) \cdot (\textsc{Lift}_D(A \rightarrow C))^{-1}
\end{aligned}$$

The other three partitions can be rewritten analogously. In turn, it can easily be validated that D' as defined by theorem 6 is in fact a distribution satisfying the constraints:

$$\begin{aligned}
Pr_{D'}[A, C] \quad = \quad & Pr_D[A, C] \cdot (\textsc{Lift}_D(A \rightarrow C))^{-1} \\
= \quad & Pr_D[A, C] \cdot \left(\frac{Pr_D[A, C]}{Pr_D[A] \cdot Pr_D[C]} \right)^{-1} \\
= \quad & Pr_D[A] \cdot Pr_D[C] \,,
\end{aligned}$$

and analogously for the other partitions. This directly implies constraints (4.1)-(4.3) by marginalizing out. Constraints (4.4)-(4.7) are met, because for all four partitions D' is defined proportionally to D. □

Please recall from section 3.4.2 that the LIFT simply reflects the factor by which a class is overrepresented in a considered subset with respect to the class prior. For the example rule the LIFT is 5, because the risk for young drivers is 5 times higher than for the average driver. Hence, the probability to see a specific young driver who had an accident in the target sample is reduced by a factor of $1/5$ compared to the original data. Each of the four partitions, defined by a combination of prediction and true label, is rescaled in the same fashion.

Theorem 6 defines a new distribution to sample from, given a *single* rule r as prior knowledge. The same result allows to incorporate more complex forms of prior knowledge. As long as the prior probabilistic knowledge is used to make discrete predictions, a LIFT for each combination of prediction and true class can be computed. The prior knowledge can be reformulated as a classifier $h : \mathcal{X} \rightarrow \mathcal{Y}$ in this case, and its contingency matrix allows to compute the LIFTs and to refer to the same definition of the new distribution. Hence, separating hyperplanes or ensembles of base classifiers like Horn logic rules are supported background theories.

Please note that no assumptions about the considered utility function were made. In fact, any utility function can be used in combination with knowledge-based sampling, although section 4.4.2 will reveal reasons to prefer metrics that scale at most linearly with the coverage of rules.

4.4. A knowledge-based rejection sampling algorithm

This section will demonstrate that the knowledge-based sampling technique can easily be realized by a specific kind of rejection sampling (cf. subsection 3.4.2). The algorithm presented for

this task applies to more general settings than those meeting constraints (4.1)-(4.7): First, nominal class labels are supported, which are only for parts of the subsequent sections assumed to be boolean. Second, $\mathcal{X}$ may be continuous and the functional dependency of $\mathcal{Y}$ on $\mathcal{X}$ is no longer assumed, so we refer to the general case of a probability density function $D : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^+$. Third, any prior knowledge in the form of a hypothesis *partitioning* the instance space may be utilized. This kind of hypothesis space is chosen, because it applies to a broad number of potential settings:

Crisp classifiers: If the prior knowledge is given in the form of a classifier $h : \mathcal{X} \to \mathcal{Y}$, then h is used to define equivalence classes with respect to the predictions; each subset of $\mathcal{X}$ for which h predicts the same class constitutes a separate partition. The total number of partitions is hence equal to the number of classes $|\mathcal{Y}|$, in this case.

Models partitioning $\mathcal{X}$: Another supported case are models that explicitly partition the instance space, e.g., following the objective to minimize an impurity criterion like entropy or the Gini index. For example, the leaves of a decision tree (cf. p. 19) partition $\mathcal{X}$, because each example is propagated to exactly one leaf. For each leaf a separate estimate of the target attribute's class distribution can be computed, which may indicate high impurity at some leaves, and almost perfect class discrimination at others. This difference can be exploited by the knowledge-based rejection sampling algorithm; the partition indices defined by the tree (e.g., the leaf numbers) may be used instead of the class predictions $y \in \mathcal{Y}$ above.

Soft classifiers: As an alternative to analyzing the partitioning behavior of a classifier, e.g. to identify leaf numbers of decision trees, one might prefer to define partitions as subsets of $\mathcal{X}$ for which the same *soft* prediction is made. For a decision tree the resulting number of partitions is bounded by the number of leaves. In more general cases, similar predictions can be aggregated (discretization), so that the number of resulting partitions becomes tractable. The impact of the number of partitions on the proposed algorithm will become clear from an analysis in section 4.4.2.

These different settings are unified by considering models of the form $h : \mathcal{X} \to \{1, \ldots, v\}$, where v denotes the number of partitions. No estimates associated with the partitions are required. The only hard constraint for knowledge-based sampling is that none of the partitions may be "pure" with respect to conditional class distributions. More precisely, for none of the partitions the conditional probability to sample an example from any of the classes may be 0, because otherwise the LIFT of a corresponding rule would also be 0, leading to an undefined inverse LIFT. In section 4.4.3 a solution to this problem will be proposed and applicability aspects of knowledge-based sampling will be discussed.

4.4.1. The Algorithm

Examples are again just assumed to be sampled i.i.d. with respect to any unknown (initial) probability density function D. Algorithm 1 shows a knowledge-based rejection sampling technique that allows to sample example sets i.i.d. from pdf D' as defined in theorem 6. It employs a given subroutine EX_D that provides classified examples sampled from D; for any given training set, EX_D just needs to operationalize uniform sub-sampling (cf. subsection 3.2.1).

If all LIFT values of the prior knowledge with respect to pdf D were known, then it would be easy to directly realize an efficient step of rejection sampling. This is usually not the case, so the algorithm takes an indirect approach. It exploits that under D' the class priors and the

Algorithm 1 Knowledge-based rejection sampling

Given:

- A subroutine EX_{D} providing classified samples $e \sim \mathsf{D}$.
- Prior knowledge $h : \mathcal{X} \to \{1, \ldots, v\}$ partitioning $\mathcal{X}$.
- A set $\pi_{\mathcal{Y}}$ that specifies the class prior π_y of each $y \in \mathcal{Y}$.
- A number m of examples to be sampled from D'.

Output:

- An example set $\mathcal{E} \sim (\mathsf{D}')^m$, where D' is defined as in theorem 6 (p. 74)

Procedure $\mathbf{EX}_{\mathbf{D}'}(\mathbf{h}, \mathbf{m}, \pi_{\mathcal{Y}})$:
Initialize a queue $Q_{i,y} := \emptyset$ for each $1 \leq i \leq v$ and class $y \in \mathcal{Y}$.
for $i = 1$ to m **do**
  sample $e = (x, y)$ by calling EX_{D}
  randomly choose label $y^* \sim \pi_{\mathcal{Y}}$
  if $(y^* \neq y)$ **then** // Find an example of class y^*, instead.
    let $i := h(x)$ // New example from partition i required.
    enqueue$(Q_{i,y}) \leftarrow e$ // Store for later usage.
    while $(Q_{i,y^*} = \emptyset)$ **do** // Wait for example with label y^* and $h(x) = i$.
      sample $e' := (x', y')$ by calling EX_{D}
      enqueue$(Q_{h(x'),y'}) \leftarrow e'$ // Store at end of corresponding queue.
    end while
    $e \leftarrow$ **dequeue**(Q_{i,y^*}) // Remove first example from queue.
  end if
  output e.
end for

probability that a randomly sampled example falls into a specific partition are defined as under D (constraints (4.2) and (4.3)), and that labels are independent of partitions (constraint (4.1)). The consequence of the indirect approach is a high rejection rate, which is compensated by storing and re-using rejected examples. To this end, the algorithm makes use of *queues*, an abstract data structure that provides two constant-time operations, *enqueue* and *dequeue*. The former stores an example in a queue, the latter removes the example from a queue that has been stored first, and returns it. In the initialization step, an empty queue is prepared for each combination of partition (e.g., prediction) $i \in \{1, \ldots, v\}$ and (true) label $y \in \mathcal{Y}$.

Technically seen, knowledge-based rejection sampling establishes a procedure $\mathsf{EX}_{\mathsf{D}'}(m)$ that outputs example sets of size m from D'. For simplicity, the class priors π_y are assumed to be known in the pseudo-code. Besides the target sample size m and the model h that defines D', the procedure $\mathsf{EX}_{\mathsf{D}'}$ hence has $\pi_{\mathcal{Y}}$ as a further input parameter. Class priors can well be estimated from large data sets, unless one class is extremely rare.

The idea of algorithm 1 is to *de-correlate* partitions and labels by means of an internal random experiment. The partitions of examples sampled from D are combined with randomly assigned labels. Each iteration of the outer loop of the algorithm yields one example e independently sampled from D', so m iterations are required. Loops start by requesting a *reference example* e from EX_{D}. Then, a target class label y^* is randomly chosen, using the probability distribution imposed by the class priors. If the class of e is y^*, then e can be output and the loop ends. Otherwise, the algorithm waits for an example from class y^* that shares the partition of the reference example defined with respect to model h. If e.g., h is a crisp classifier, then the predictions of h

for the reference and for the target example need to be identical. The queues serve as example caches. Each example that cannot be output is appended to a queue "responsible" for the specific combination of true label and partition. The algorithm does not have to wait for a specific kind of example if there are examples in the corresponding queue. In this case, the first example is retrieved and returned. If the queue for a required label and partition is empty, then the algorithm reads (and caches) examples from EX_{D} until one is found that matches the specification.

The fact that examples are requested sequentially from EX_{D} and that each iteration yields one example sampled from D' illustrates the applicability to streaming data. The capacity of queues can be constrained, in this case. When sub-sampling from a database instead, a *set* containing all output examples may as well be returned after termination of algorithm 1.

As an advantage, the algorithm does not require any probability estimates, except for the class priors; still it *exactly* transforms D into D' or EX_{D} into $\mathsf{EX}_{\mathsf{D}'}$, respectively. This will be shown in section 4.4.2. For completeness it should be mentioned, that even the assumption of known class priors can be avoided without losing the property of having an exact transformation. It is sufficient to replace the random selection of a target class y^* by defining it to be the class of a second, independent reference example requested from EX_{D}.

An interesting aspect is, that knowledge-based rejection sampling can be simplified considerably when simultaneously considering a step of stratification. In this case, the class priors for guiding the random target class assignment can be replaced by uniform priors. For boolean labels this means to assign the target class y^* by simply flipping a fair coin. This variant of the algorithm samples exactly from the stratified random sample density function of D' (cf. Def. 31, p. 56). This is a useful property, because theorem 5 (p. 59) states that stratification transform the optimization problem of subgroup discovery with the WRAcc metric into a common classifier induction problem. This technical simplification will be utilized in section 4.5.

4.4.2. Analysis

In the next paragraphs the correctness of the presented algorithm in terms of the previously motivated constraints is shown. Crucial properties of the algorithm are its runtime and sample complexity. Obviously, these quantities are asymptotically identical for the presented algorithm if assigning uniform costs ($O(1)$) to each call of EX_{D}. After proving its correctness, the sample complexity of the algorithm will be analyzed.

Correctness

Proposition 3 *Algorithm 1 returns a sample from the probability density function* D' *that satisfies constraints* (4.1)-(4.7).

Proof
Each iteration starts by sampling a reference example $e = (x, y)$ and randomly selecting a label. A first property of the algorithm is that the example $e = (x^*, y^*)$ that is returned at the end of each iteration satisfies the constraints $h(x) = h(x^*)$ and $y = y^*$: If the class of the reference example e is identical to the randomly selected class, then e satisfies both constraints and is returned; if the classes y and y^* differ, then an example taken from queue $Q_{h(x),y^*}$ is returned. Only examples sharing partition $h(x)$ and label y^* are appended to this queue, so the returned example again satisfies both constraints.

The probability to sample an example of any specific class $y^* \in \mathcal{Y}$ in any iteration is (by construction of the internal random experiment) identical to the probability that an example e

returned by EX_D has label y^*. This implies constraint (4.3), that the class priors under D' – implicitly defined by the algorithm – need to be identical to the class priors under D:

$$(\forall y \in \mathcal{Y}) \quad : \quad Pr_{D'}[y] = Pr_D[y]$$

Analogously, the probability to sample an example from any specific partition, e.g., so that $h(x) = i$ for a given $i \in \{1, \ldots, v\}$, is equivalent to the probability of sampling an example from that partition from EX_D. In more formal terms this means

$$(\forall i \in \{1, \ldots, v\}) \quad : \quad Pr_{(x,y)\sim D'}[h(x) = i] = Pr_{(x,y)\sim D}[h(x) = i],$$

which is the required generalization of constraint (4.2) for $D : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^+$ and an arbitrary (finite) number of partitions.

Selecting y^* at random guarantees that the label is independent of the partition (or predicted label, respectively), so constraint (4.1) is also met.

The last constraints (4.4)-(4.7) require, that the conditional distributions of examples, given a specific pair of partition and label, do not change from D to D'. Obviously, the resulting distribution D' is not affected by switching the first two lines, so that the target label y^* is chosen first. If the reference example $e = (x, y)$ sampled from D shares the chosen class ($y = y^*$), then e is directly propagated and the constraint is met. If the class of reference example e is not y^*, then the first example (x^*, y^*) with $h(x^*) = h(x)$ is returned from $Q_{h(x),y^*}$. It is an example that belongs to partition $(h(x), y^*)$ and did not match a randomly selected class. This obviously does not skew the conditional distribution either, because (i) when enqueuing, the random experiment is independent of the specific examples in each partition, and because (ii) when dequeuing, the i.i.d. property of EX_D implies that each permutation of examples in $Q_{h(x),y^*}$ is equally likely. Hence, the example finally returned by the algorithm has been sampled from D conditioned on $h(x)$ and y^*, as required. This completes the proof. □

As shown in subsection 4.3.2, constraints (4.1)-(4.4) induce pdf D' uniquely.

Corollary 1 *The* knowledge-based rejection sampling algorithm *(algorithm 1) operationalizes theorem 6 properly by means of rejection sampling.*

Computational Complexity

The runtime complexity of the algorithm is linear in the number of instances requested from its subroutine EX_D, because in each inner loop exactly one example is requested. For this reason the analysis focuses on sample complexity.

Some abbreviations will ease notation in the following paragraphs. For each tuple (i, j) with $i \in \{1, \ldots, v\}$ being a partition defined with respect to h and $j \in \{1, \ldots, |\mathcal{Y}|\}$ being the index of a label from $\mathcal{Y}$, the term[1]

$$\text{lift}_{i,j} := \text{LIFT}_D((h(x) = i) \to y_j),$$

refers to the corresponding LIFT under pdf D. The minimum of all these LIFTs under D is denoted as lift_{min}. Further, p_{min} denotes the probability of the least probable combination of class $y \in \mathcal{Y}$ and partition $i \in \{1, \ldots, v\}$ with respect to pdf D, and analogously, p'_{min} refers to the probability of the least probable combination with respect to D'.
The following lemma provides an upper bound on the sample complexity of algorithm 1:

[1] By construction, the corresponding LIFTs under D' are all 1, so no subscript "D" is required.

Lemma 5 *Let* $h : \mathcal{X} \to \{1, \ldots, v\}$ *be a model partitioning the instance space* $\mathcal{X}$*, let* $\mathcal{Y}$ *denote the target label, and let* D *denote a corresponding probability density function.*
If, for a given confidence parameter $\delta \in (0, 1)$*,*

$$m \geq 3 \cdot \frac{\ln(v) + \ln(|\mathcal{Y}|) + \ln 2 + \ln(1/\delta)}{p'_{min}}$$

examples are requested from D' *defined by theorem 6, then with a probability (confidence) of at least* $(1 - \delta)$ *the knowledge-based rejection sampling algorithm does not request more than* $(4m/\text{lift}_{min})$ *examples from* EX_D*.*

Proof
For a given confidence parameter δ and given target sample size m, we identify sufficient criteria for the algorithm not to request more than n examples from EX_D. To this end we assume that only a sample of size n is available, which is sampled from D in advance, and that no more examples are provided afterwards. When sampling from D', examples are simply taken from the corresponding queues. The algorithm basically just deviates from this scenario in that it reads the examples from EX_D on demand. It "succeeds" if it does not read more examples from any of the queues than are available after the n examples have been distributed to the queues. Considering the reference example as a sample candidate is the second point in which the algorithm differs from the setting above. This will usually lead to a small advantage regarding expected sample sizes, but as a consequence, we may not assume independence between the number of samples in each queue and the corresponding number of requests.

For $h : \mathcal{X} \to \{1, \ldots, v\}$ there are $v \cdot |\mathcal{Y}|$ queues. Each example $(x, y_j) \in \mathcal{X} \times \mathcal{Y}$, $i := h(x)$, is stored in queue $Q_{i,j}$, where y_j simply denotes the j^{th} class from $\mathcal{Y}$, referring to an arbitrary but fixed order. The set of all $q := v \cdot |\mathcal{Y}|$ queues used by the algorithm is denoted as $\mathcal{Q}$. For any queue $Q_{i,j} \in \mathcal{Q}$ we have a fixed probability $p_{i,j}$ of an example randomly sampled from D to be *appended* to $Q_{i,j}$, and another probability $p'_{i,j}$ of sampling an example from D' that triggers a corresponding *request* from $Q_{i,j}$.

Let $Y^{(t)}_{i,j}$ denote the binomially distributed random variable that specifies the fraction of randomly sampled examples that belong to queue $Q_{i,j}$ if the absolute number of sampled examples is t. The corresponding distribution, D or D', is given in the context, but also by the superscript: A sample size of n always refers to pdf D, while m indicates pdf D'.

We are going to investigate the probability of two events, in order to derive probabilistic bounds on the sample complexity of the algorithm. The first event occurs, if after distributing the n examples requested from EX_D to queues, there are *at least half the expected number* of examples in each of the queues. The second event occurs, if m examples are sampled from D' by the algorithm, but from all the queues *at most double the expected number* is requested. We can give bounds on the probabilities of both events according to Chernoff (cf. section 3.2.2, p. 45), with $\lambda = 1$ and $\lambda = 1/2$, respectively:

$$(\forall Q_{i,j} \in \mathcal{Q}) : Pr_D\left(Y^{(n)}_{i,j} \leq \mathbb{E}_D[Y^{(n)}_{i,j}]/2\right) \leq e^{-(n \cdot p_{i,j})/8}$$
$$(\forall Q_{i,j} \in \mathcal{Q}) : Pr_{D'}\left(Y^{(m)}_{i,j} \geq 2\mathbb{E}_{D'}[Y^{(m)}_{i,j}]\right) \leq e^{-(m \cdot p'_{i,j})/3}$$

The following inequality is an upper bound on the risk that *any* of the queues does not contain at least half as many examples as expected after distributing n examples sampled from D:

$$\sum_{Q_{i,j} \in \mathcal{Q}} Pr_D\left(Y^{(n)}_{i,j} \leq \mathbb{E}_D[Y^{(n)}_{i,j}]/2\right) \leq \sum_{Q_{i,j} \in \mathcal{Q}} e^{-(n \cdot p_{i,j})/8} \leq q \cdot e^{-(n \cdot p_{min})/8}$$

To upper-bound the risk by $\delta/2$ that any of the queues contains too few elements, the union bound allows to derive the following sufficient sample complexity n:

$$q \cdot e^{-(n \cdot p_{min})/8} \leq \delta/2 \quad \Leftrightarrow \quad n \geq 8 \cdot \frac{\ln q + \ln 2 + \ln(1/\delta)}{p_{min}} \tag{4.8}$$

Analogously, a risk of at most $\delta/2$ for the event of requesting more than $2\mathbb{E}_{D'}[Y_{i,j}^{(m)}]$ examples from any queue when sampling from D' can be guaranteed when the target sample size m is at least

$$q \cdot e^{-(m \cdot p'_{min})/3} \leq \delta/2 \quad \Leftrightarrow \quad m \geq 3 \cdot \frac{\ln q + \ln 2 + \ln(1/\delta)}{p'_{min}}. \tag{4.9}$$

The risk is bounded by $\delta/2$ for both kinds of failure, so with a probability of at least $(1-\delta)$ none of these events occurs. Please note that we again make use of the union bound at this point, which does not require any of the variables $Y_{i,j}^{(m)}$ to be independent of the vector of variables $Y_{i,j}^{(n)}$. We hence just exploit that the algorithm samples from D' (Prop. 3) to derive eqn.(4.9).

If none of the two kinds of failures occurs, the number of examples requested from each queue is at most twice as high as expected, while at the same time each queue contains no less than half the expected number of examples. Hence, it is sufficient if, after distributing the n examples sampled from EX_D, the expected number of examples in each queue is 4 times higher than the expected number of requests when sampling m examples from D'. For each individual queue this criterion can be reformulated in terms of n, m, and the corresponding LIFT, because theorem 6 implies $p'_{i,j} = p_{i,j} \cdot (lift_{i,j})^{-1}$:

$$\begin{aligned} & n \cdot p_{i,j} = \mathbb{E}_D[n \cdot Y_{i,j}^{(n)}] \geq 4 \cdot \mathbb{E}_{D'}[m \cdot Y_{i,j}^{(m)}] = 4m \cdot p'_{i,j} \\ \Leftrightarrow \quad & n \geq 4m \cdot \frac{p'_{i,j}}{p_{i,j}} = 4m \cdot \frac{p_{i,j} \cdot (lift_{i,j})^{-1}}{p_{i,j}} = \frac{4m}{lift_{i,j}}. \end{aligned}$$

This set of constraints can obviously be simplified; any sample size n that meets the constraint for queues having a LIFT of $lift_{min}$ will meet the constraint for all queues:

$$(\forall Q_{i,j} \in \mathcal{Q}) \quad : \quad \mathbb{E}_D[n \cdot Y_{i,j}^{(n)}] \geq 4 \cdot \mathbb{E}_{D'}[m \cdot Y_{i,j}^{(m)}] \qquad \Leftrightarrow \qquad n \geq \frac{4m}{lift_{min}} \tag{4.10}$$

The minimal sample size n for any given confidence level δ is hence constrained by eqn. (4.8) and eqn. (4.10). From theorem 6 we can derive

$$p'_{i,j} = p_{i,j} \cdot (lift_{i,j})^{-1} \quad \Rightarrow \quad p'_{i,j} \leq p_{i,j} \cdot (lift_{min})^{-1}$$

for each queue $Q_{i,j}$, and hence

$$p'_{min} = \min_{i,j} \ (p'_{i,j}) \quad \leq \quad \min_{i,j} \ (p_{i,j}) \cdot (lift_{min})^{-1} = p_{min} \cdot (lift_{min})^{-1}.$$

If m is lower-bounded by eqn. (4.9), as assumed, then eqn. (4.10) is the dominating constraint, and eqn. (4.8) is redundant:

$$\begin{aligned} & p'_{min} \leq p_{min} \cdot (lift_{min})^{-1} \\ \Leftrightarrow \quad & \frac{1}{lift_{min} \cdot p'_{min}} \geq \frac{1}{p_{min}} \\ \Leftrightarrow \quad & \frac{1}{lift_{min}} \cdot \frac{\ln q + \ln 2 + \ln(1/\delta)}{p'_{min}} \geq \frac{\ln(q) + \ln 2 + \ln(1/\delta)}{p_{min}} \\ \Rightarrow \quad & \frac{4m}{lift_{min}} \geq 8 \cdot \frac{\ln q + \ln 2 + \ln(1/\delta)}{p_{min}} \end{aligned}$$

Combining eqn. (4.10) with eqn. (4.9) and the definition of q yields the lemma. □

The factor of 4 in lemma 5 is pessimistic, but as shown next, a linear dependency on $(\text{lift}_{\min})^{-1}$ holds in fact. In contrast to lemma 5, the following proposition directly addresses the *expected* sample complexity.

Proposition 4 *The asymptotic expected sample and runtime complexity of the knowledge-based rejection sampling algorithm for a target sample size of* m *is*

$$\mathbb{E}(n) = \Theta\left(\frac{m}{\text{lift}_{\min}}\right).$$

Proof
The proof of lemma 5 implicitly provides an asymptotic bound of $O(m/\text{lift}_{\min})$ for $\mathbb{E}(n)$: With a growing target sample size m the probability to require more than $(4m/\text{lift}_{\min})$ samples from D vanishes; all considered random variables $Y_{i,j}^{(t)}$ are binomially distributed and converge to the normal distribution with a variance of 0 in the limit. Hence, in the limit the fraction of examples associated to each individual queue meets the expected value, and the expected sample size "converges" to values no larger than $(4m/\text{lift}_{\min})$. Formally we have:

$$\lim_{m\to\infty}\left(\frac{\mathbb{E}(n)}{4m\cdot(\text{lift}_{\min})^{-1}}\right) \leq 1.$$

Convergent sequences are bounded, so deviations from the expected value $\mathbb{E}(n)$ vanish as a constant factor in the asymptotic notation.

To derive a loose lower-bound on $\mathbb{E}(n)$ it suffices to consider a queue Q_{i^*,j^*} with $\text{lift}_{i^*,j^*} = \text{lift}_{\min}$. On average, sampling m examples under D' requires $\mathbb{E}_{D'}[Y_{i,j}^{(m)}]$ examples from each queue $Q_{i,j}$. For a given m we consider a sample size $\tilde{n}$ that is characterized by having in expectation the same number of examples stored in Q_{i^*,j^*} as are expected to be requested:

$$\mathbb{E}_D[\tilde{n}\cdot Y_{i^*,j^*}^{(\tilde{n})}] = \mathbb{E}_{D'}[m\cdot Y_{i^*,j^*}^{(m)}] \quad\Leftrightarrow\quad \tilde{n}\cdot p_{i^*,j^*} = m\cdot p'_{i^*,j^*} \quad\Leftrightarrow\quad \tilde{n} = m\cdot\frac{p'_{i^*,j^*}}{p_{i^*,j^*}}.$$

The algorithm "fails" for a sample size $\tilde{n}$, if $\tilde{n}\cdot Y_{i^*,j^*}^{(\tilde{n})} < m\cdot Y_{i^*,j^*}^{(m)}$, that is, if more examples are requested than are available. For large m we can again refer to an approximation of these binomially distributed random variables by normal distributions. Because of the symmetric form of the normal distribution we can conclude that the risk δ of failure is at least $1/2$, if $Y_{i^*,j^*}^{(\tilde{n})}$ and $Y_{i^*,j^*}^{(m)}$ are independent.

Let the random variable n reflect the number of actually required samples for any fixed target sample size m. Since $n \geq 0$ and $Pr(n > \tilde{n}) \geq 1/2$, we have $\mathbb{E}(n) \geq \tilde{n}/2$. In the limit, the number of examples requested from Q_{i^*,j^*} is determined only by the corresponding weight under D' (cf. Prop. 3), and $Y_{i^*,j^*}^{(m)}$ is in fact independent of $Y_{i^*,j^*}^{(n)}$. This implies

$$\lim_{m\to\infty}\left(\mathbb{E}(n)/\left(\frac{\tilde{n}}{2}\right)\right) \geq 1 \quad\Rightarrow\quad \lim_{m\to\infty}\left(\mathbb{E}(n)/\left(\frac{m}{2}\cdot\frac{p'_{i^*,j^*}}{p_{i^*,j^*}}\right)\right) \geq 1.$$

After rewriting the denominator in terms of $\text{lift}_{\min} = p_{i^*,j^*}/p'_{i^*,j^*}$ we get the result

$$\mathbb{E}(n) = \Omega\left(\frac{m}{\text{lift}_{\min}}\right),$$

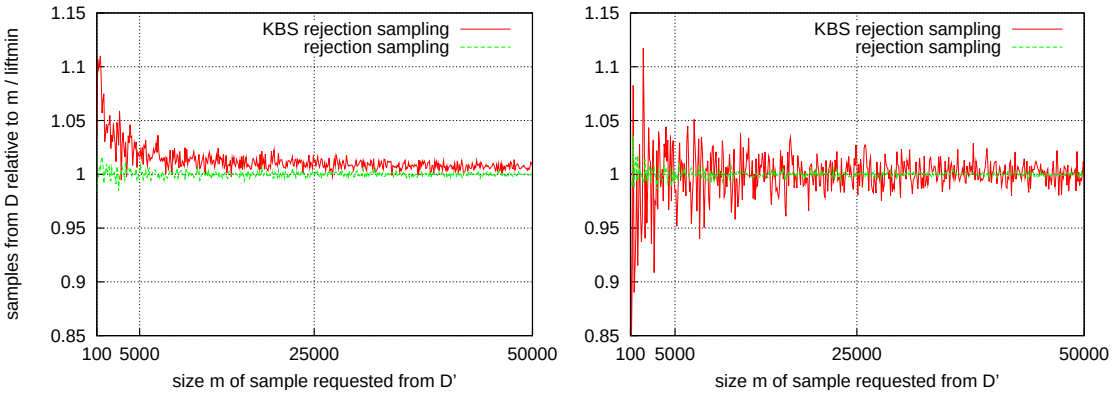
which completes the proof. □

Proposition 4 states that the asymptotic sample complexity of the knowledge-based rejection sampling algorithm is linear in the number of requested samples and the reciprocal minimal LIFT. Clearly, for practitioners the constant factors not visible in the asymptotic notation are also of interest. The factor of 4 derived in lemma 5 seems a bit pessimistic, while the factor of $1/2$ seems to be too optimistic. This is discussed in the following paragraphs.

First of all, the results are compared to much simpler kinds of rejection sampling. As mentioned above, if the true LIFT values of the prior knowledge were known, then we could directly apply rejection sampling in a straightforward manner, i.e without relying on reference examples. Optimal rejection sampling applies the maximal scaling factor c (cf. subsection 3.4.2) to D'/D that meets the constraint to yield well-defined acceptance probabilities. It is easily seen that this is the case for $c := \text{lift}_{\min}$, because for each x contained in the corresponding subset we have the highest value $D'(x)/D(x) = (\text{lift}_{\min})^{-1}$; after multiplying with c, the corresponding probabilities of acceptance are 1, and no larger than 1 for all other subsets. The pdf $D'(x) := D(x) \cdot (\text{LIFT}(x, h))^{-1}$ for any given model h has a total weight of 1. By rescaling we increase the expected sample complexity by a factor of $c^{-1} = (\text{lift}_{\min})^{-1}$, because the average probability to accept an example is c afterwards. This implies that an optimal direct rejection sampling algorithm has an expected sample complexity of $\mathbb{E}(n) = m/\text{lift}_{\min}$.

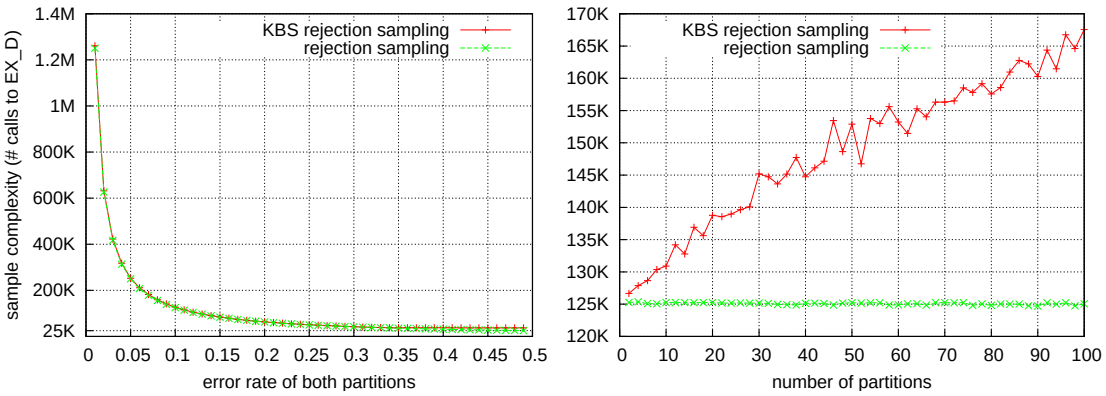
Figure 4.1 illustrates that knowledge-based rejection sampling has a very similar sample complexity. The expected complexity only depends on the contingency matrix of the prior knowledge, so the knowledge-based sampling algorithm was evaluated based on a synthetic procedure EX_D and partitioning models h with fixed performances. Please keep in mind that the sample complexities of optimal rejection sampling are only depicted for reference. As discussed, this approach is not feasible in practice, because it requires the true contingency matrix to be known. For the experiments this information was supplied to the procedure. Each point in the figures is the result of averaging 10 repetitions of sampling a fixed number of examples from D'. The y-axis refers to the numbers of examples requested from EX_D for this purpose.

Figure 4.1(a) and 4.1(b) show the sample complexities n divided by the expected values of $m/\text{lift}_{\min}$ to illustrate that the true factor "missing" in the asymptotic result of Prop. 4 is very close to 1. In Fig. 4.1(a), prior knowledge in the form of a boolean classifier is sampled out. Its precision is 90% when predicting the positive and the negative class, and both of these predictions are equally likely. The additional constant factor of knowledge-based rejection sampling is about 1.1 for a very small target sample size m of 100 examples, and it converges to 1 with growing m. In Fig. 4.1(b) the performance of the model is 95% for the covered, and 55% for the uncovered subset. The coverage is only 10%. In this case, knowledge-based rejection sampling performs differently; its average sample complexity is about $m/\text{lift}_{\min}$ (factor: 1) for all evaluated sample sizes, but the variance for small m is much higher than for larger m. Optimal rejection sampling has a lower average sample complexity in the former experiment, but the advantage vanishes with an increasing value of m. Its variance is significantly lower in both figures.



(a) Sampling out a boolean classifier with 90% precision on both covered and uncovered subset. The coverages are both 50%.

(b) Sampling out a boolean classifier with 95% precision and 10% coverage. The precision on the uncovered subset is 55%.



(c) Error rate for both predictions (balanced coverage) vs. absolute number of samples required from D to sample 25.000 examples from D'.

(d) Varying number of partitions with precision of 90% vs. absolute number of samples required from D to sample 25.000 examples from D'.

Figure 4.1.: Empirical evaluation of the sample complexity of knowledge-based rejection sampling (algorithm 1). The plots depict the complexities for prior knowledge with different precisions and for a varying number of partitions.

Figure 4.1(c) illustrates the impact of model accuracy on the sample complexity measured in absolute terms. To this end, the experimental setting underlying Fig. 4.1(a) was adapted: Both predictions of a boolean model are equally likely in the experiment, and have the same precision (or error rate, respectively). The target sample size was chosen to be m=25.000. It shows that both algorithms have almost the same performance.

For error rates above 35%, the rejection sampling algorithm has slightly lower sample complexities, but the knowledge-based algorithm is also very close to the value of 25.000 examples. The plot confirms the theoretical finding that very accurate prior knowledge increases the required number of samples significantly. For example, if class priors are uniform, then an accuracy of 99% leads to acceptance rates of one out of 50 samples. This is still tractable when mining from very large databases, especially when comparing to the super-linear time consumption of rule or model induction algorithms.

Figure 4.1(d) confirms another theoretical finding: The number of partitions has a low impact on the algorithm. Even if each partition has the same high precision of 90%, the sample

complexity of knowledge-based rejection sampling increases moderately, e.g. about 30% when increasing the number of partitions from 2 to 100. By construction, the number of partitions has no effect on optimal rejection sampling. It is worth noting that substituting estimated performances, e.g. estimated class distributions at the leaves of decision trees, may easily increase the complexity of direct rejection sampling. Optimistic estimates based on the training data naturally result in a lower value of lift_{min}.

How knowledge-based sampling affects subgroup discovery

Algorithm 1 changes the distribution underlying a data set so that the label is conditionally independent of any given prior knowledge. This is a consequence of sampling from D', a pdf which meets eqn. (4.1) (p. 73). The following proposition helps to understand the effect of changing the distribution on subsequently applied subgroup discovery algorithms in more detail.

Proposition 5 *Let S_i denote partition $i \in \{1,\ldots,v\}$ defined with respect to a model $h : \mathcal{X} \to \{1,\ldots,v\}$, let C be a specified target class, and let $\overline{C}$ refer to an aggregation of the remaining classes to a single negative class. If in a subset $\text{Ext}(A \cap S_i) \subseteq \text{Ext}(S_i)$ the odds ratio under D is $s \in \mathbb{R}^+$ times higher than in the complete partition $\text{Ext}(S_i)$, formally*

$$\frac{\text{Pr}_D(C \mid A, S_i)}{\text{Pr}_D(\overline{C} \mid A, S_i)} = s \cdot \frac{\text{Pr}_D(C \mid S_i)}{\text{Pr}_D(\overline{C} \mid S_i)}, \tag{4.11}$$

then the subset $\text{Ext}(A, S_i)$ has a LIFT *ratio of*

$$\beta_{D'}[(A, S_i) \to C] := \frac{\text{LIFT}_{D'}(A, S_i \to C)}{\text{LIFT}_{D'}(A, S_i \to \overline{C})} = s.$$

under the probability density function D' defined by theorem 6 (p. 74).

Proof
First we define

$$s^{(+)} := \frac{\text{Pr}_D(C, A, S_i)}{\text{Pr}_D(C, S_i)} \text{ and } s^{(-)} := \frac{\text{Pr}_D(\overline{C}, A, S_i)}{\text{Pr}_D(\overline{C}, S_i)}.$$

Rearranging terms in eqn. (4.11) we see that $s = s^{(+)}/s^{(-)}$. According to theorem 6 we have $D'(x) = D(x)/\text{LIFT}_D(S_i \to C)$ for $x \in \text{Ext}(S_i, C)$. This allows to rewrite probabilities with respect to pdf D in this subset:

$$\begin{aligned} \text{Pr}_{D'}[C, A, S_i] &= \frac{\text{Pr}_D[C, A, S_i]}{\text{LIFT}_D(S_i \to C)} = \frac{s^{(+)} \cdot \text{Pr}_D(C, S_i)}{\text{Pr}_D[C \mid S_i]/\text{Pr}_D[C]} \\ &= s^{(+)} \cdot \text{Pr}_D[C] \cdot \text{Pr}_D[S_i] \end{aligned} \tag{4.12}$$

From $D'(x) = D(x)/\text{LIFT}_D(S_i \to \overline{C})$ for $x \in \text{Ext}(S_i, \overline{C})$ we can analogously derive

$$\text{Pr}_{D'}\left[\overline{C}, A, S_i\right] = s^{(-)} \cdot \text{Pr}_D\left[\overline{C}\right] \cdot \text{Pr}_D[S_i]. \tag{4.13}$$

Combining the definition of the LIFT, eqn. (4.12), and (4.13) yields the result:

$$\beta_{D'}[(A, S_i) \to C] = \frac{\text{LIFT}_{D'}(A, S_i \to C)}{\text{LIFT}_{D'}(A, S_i \to \overline{C})} = \frac{\text{Pr}_{D'}[C, A, S_i]/\text{Pr}_{D'}[C]}{\text{Pr}_{D'}\left[\overline{C}, A, S_i\right]/\text{Pr}_{D'}\left[\overline{C}\right]} = \frac{s^{(+)}}{s^{(-)}} = s$$

□

As proposition 5 illustrates, each rule $r : A \rightarrow C$ with $Pr_D(C \mid A, S_i) = Pr_D(C \mid S_i)$ for a partition S_i of a prior model has a LIFT ratio $\beta_{D'}[(A, S_i) \rightarrow C] = 1$ in the intersection of $Ext(A)$ and $Ext(S_i)$. This implies a BIAS of 0 in $Ext(A, S_i)$ with respect to D'. The weighted relative accuracy (WRACC) is a weighted combination of the biases of subsets. For subsets S_i, $1 \leq i \leq \nu$, the WRACC of a rule candidate $r : A \rightarrow C$ can be rewritten as

$$\text{WRACC}(A \rightarrow C) = \text{COV}(r) \cdot \text{BIAS}(r) = \sum_{i=1}^{\nu} Pr[A, S_i] \cdot \text{BIAS}((A, S_i) \rightarrow C).$$

This shows that extending a subgroup by an additional subset with no bias does not change the utility score, so whenever the conditional class distribution in a partition S_i meets the corresponding distribution in S_i when conditioning on A (or "A performs on S_i as predicted"), this subset is implicitly ignored by the WRACC metric under D'.

In turn, proposition 5 states that an increasing odds ratio for a partition S_i under D when conditioning on A leads to a proportional increase in the LIFT ratio under D' for $Ext(A, S_i)$. The BIAS is a fundamental quantity of most utility functions. It is identical to the precision ($Pr(C \mid A, S_i)$) up to an additive constant term. WRACC maximizes the BIAS $Pr(C \mid A) - Pr(C)$ weighted by the individual coverages of partitions, so it implicitly maximizes the average precision, and hence the odds ratio ($Pr(C \mid A)/Pr(\overline{C} \mid A)$). This means that – when mining based on WRACC under D' – the degrees of deviation between predicted and true conditional class distributions induce the preference order for rule selection. This incorporates deviation from prior knowledge into the concept of optimizing with respect to unexpectedness; conditioning on (a subgroup) A should change the class distribution as much as possible. Due to the other constraints that motivated the choice of D', the resulting samples are not skewed unnecessarily. In particular, the coverages of all partitions remain, so the individual deviations from expectation are weighted as by D when averaging[2] over the WRACC of all partitions. In subsection 4.5.1 this will be further investigated for sequential subgroup mining, and it will be complemented by an analysis for predictive settings in subsection 5.3.4.

Other utility functions may also be used in combination with knowledge-based sampling. An attractive choice is the function $Q^{(0.5)}(r)$ which is factor-equivalent to the binomial test function (cf. eqn. (2.2), p. 23). The difference to WRACC can be seen after rewriting

$$Q^{(0.5)}(r) = \sqrt{\text{COV}(r)} \cdot \text{BIAS}(r) = \frac{\text{WRACC}(r)}{\sqrt{\text{COV}(r)}}.$$

This connection to WRACC reveals that including unbiased subsets into the extension of a rule r even *decreases* the utility score of this metric, given an initial score above 0: As discussed above, WRACC is invariant against including unbiased subsets, but such a step increases the coverage. Apart from this difference both metrics are identical, so $Q^{(0.5)}$ also favors rules for which conditioning on the antecedent changes the class distribution.

The situation is different for $Q^{(2)}$, which can easily be seen to *multiply* WRACC with $\text{COV}(r)$; including unbiased subsets hence increases the utility score of $Q^{(2)}$. Moreover, even including subsets with a negative BIAS, which decreases the average BIAS, may often increase the score

[2] Please recall from lemma 2 (p. 53), that WRACC is an instance averaging function, so the total WRACC is a coverage-weighted average of the WRACC of all partitions.

due to an over-compensation by an increasing coverage. This is a counter-intuitive behavior in the scope of knowledge-based sampling, because it compromises the discovery of unexpected patterns. It seems reasonable to exclude functions like $Q^{(\alpha)}$ with $\alpha > 1$ from consideration in this context. The experiments in this chapter will focus on WRAcc, as it provides the most straightforward criterion for unexpectedness.

4.4.3. Discussion

Knowledge-based sampling can be considered to preprocess the data by "sampling out" the prior knowledge. In section 4.4.2 rejection sampling (cf. section 3.4.2) has been shown to be capable of constructing samples that correspond to the previously defined distribution D'. This distribution is very intuitive due to the set of motivating constraints, but also due to the simple function that transforms the initial into the new distribution. The resulting samples of knowledge-based sampling do not contain any example weights, so arbitrary data mining algorithms may be applied subsequently; this includes rule or decision tree learners, as discussed in subsection 4.3.2 or in (Scholz, 2005a), respectively.

To discover subgroups in the presence of prior knowledge, a straightforward way to proceed after knowledge-based sampling is to apply an algorithm like MIDOS to the constructed sample. MIDOS (Wrobel, 1997) is based on the WRAcc metric, so it will ignore rules with an extension for which the class distribution can be predicted from prior knowledge; the LIFT of any rule having exactly the expected precision will be 1 after knowledge-based sampling, leading to a BIAS and hence utility of 0. Rules that e.g., overlap with previously found rules have their odds ratios divided by the expected odds ratios, which means that only their additional contribution will be accounted for. In other words, rules are ranked according to their deviation from expectation.

In a sub-sampling scenario in which the complete data set is an i.i.d. sample itself, we may be interested in giving probabilistic guarantees. In particular, we may want to provide confidence bounds and guarantees that the rules are ϵ-close to the best rules in our hypothesis space. In this case, we may simply make use of the data streaming capability of knowledge-based rejection sampling, and "pipe" the examples sampled from the new distribution into an adaptive sampling algorithm (cf. subsection 3.3.2), e.g. GSS (Scheffer & Wrobel, 2002). This algorithm solves the approximately k-best rules problem (Def. 28, p. 51) regardless of the underlying distribution, so it also yields reliable confidence bounds for the altered subgroup discovery task.

A remarkable property of knowledge-based rejection sampling is, that it does not require the LIFTs or any other performance indicators of models with respect to the unknown underlying distribution to be known, but it still exactly provides samples from D', a pdf for which the label is uncorrelated with prior knowledge.

As the analysis in subsection 4.4.2 illustrates, knowledge-based sampling with respect to discrete prediction models, but also with respect to functions partitioning the instance space based on impurity-criteria, is *efficiently* possible. From proposition 5 it can be seen, that the numbers of partitions and classes have a neglible influence for (reasonably) large target sample sizes. Intuitively, the main bottleneck for knowledge-based rejection sampling is the combination of class and prediction (or partition) which is biased the most, in the sense that the label is highly underrepresented in the corresponding partition; the asymptotic sample complexity is proportional to the largest reciprocal LIFT value.

From an application point of view, the main constraint is hence the accuracy of the prior knowledge. The higher the accuracy of our knowledge or model is, the harder it is to construct uncorrelated samples. This aspect of the sampling technique is consistent with our intuition: If the prior knowledge is highly accurate, then the required evidence (sample complexity) for

any significant novel findings is higher than in the case without any prior knowledge. More precisely, since the sampling algorithm is only concerned with partitions and the true classes, problems arise if partitions are almost pure, or if at least one of the classes does not appear in one of the partitions at all. In such cases the LIFT of the corresponding queue is close to or equals 0, so $(\text{lift}_{\min})^{-1}$ becomes very large, or even undefined. A straightforward solution is to follow the sequential covering approach for binary target attributes in this case, which means to simply remove the "covered" examples, because they are fully "explained" by the model. Removing the covered examples changes the marginal probability of the corresponding partition to 0, which violates constraint (4.2) (p. 73). In chapter 5 it will be shown that, with respect to predictive performance, this violation is not critical.

In the presence of precise base models, the rejection rates of algorithm 1 may become high, resulting in small sample sizes, unless sampling from very large data sets. An attractive alternative to rejection sampling is to use example weights, as discussed in subsection 3.4.2. Weighting has been reported in the literature to work well for applications like boosting (cf. chapter 5), but a drawback of this solution is, that it requires all subsequently applied machine learning algorithms to be capable of using weight-annotated data. Another drawback is, that single points, including noise, may receive high weights. This results in poor approximations of the target distribution. In fact, a known problem of ADABOOST (Freund & Schapire, 1997) is that weights may increase exponentially, which has been reported to occur in particular for noisy data (Domingo & Watanabe, 2000). Hence, re-sampling is a more precise alternative and helps to avoid overfitting, but the amount of remaining training data should correspond to a point close to convergence in the learning curve of the model induction algorithm (cf. section 3.3.1). Otherwise, any serious sample size reduction due to rejection sampling might be detrimental to the quality of models, so reweighting is preferable. Learning curves are usually unknown, and detecting convergence remains a crucial problem for real-world applications. A reasonable heuristic is to use subsampling if the data does not fit into main memory or the computational costs for learning need to be reduced, and to use weights, otherwise.

4.5. Sequential subgroup discovery algorithms

Previous sections of this chapter introduced the notion of *knowledge-based sampling*, in particular for prior knowledge that implicitly partitions the instance space. This important class of prior knowledge contains all discrete classifiers, impurity minimizing decision trees, and classification rules. All these models can be sampled out exactly, without requiring any performance estimates. This section discusses an extension of the presented framework, that allows to sequentially mine unexpected patterns.

4.5.1. KBS-SD

The crucial observation underlying the subsequently proposed sequential subgroup discovery is, that each time a new pattern is identified it can be assumed to refine the prior knowledge. When mining the next pattern, all subsequently identified patterns may be referred to as *given*. This fosters diversity in reported rule sets. The focus of this section is on how transforming the underlying pdf and combining rules to an ensemble may be utilized to improve data mining results. This discussion can be decoupled from scalability aspects. The technique proposed next may be referred to as a representative of a larger family of algorithms, all sharing the underlying goal of sequentially mining relative to all previously discovered patterns. More complex alternatives

Algorithm 2 Algorithm KBS-SD

Input:
• $\mathcal{E} = (x_1, y_1), \ldots, (x_n, y_n)$
• number of iterations k
Output:
• Set of k classification rules

KBS-SD($\mathcal{E}$, k):
Let D_0 denote the uniform distribution over $\mathcal{E}$.
for each $y \in \mathcal{Y}$ **do**
  $\pi(y) := \frac{1}{n} \sum_{i=1}^{n} I[y_i = y]$ // compute class priors, I denotes indicator function
end for
Let $D_1(x_i) := \pi(y_i)^{-1}$ for $i \in \{1, \ldots, n\}$. // D_1: stratified version of D_0
for $t = 1$ to k **do**
  $r_t \leftarrow$ RULEINDUCTION(D_t, $\mathcal{E}$)
  Compute contingency matrix for r_t // ... from $\mathcal{E}$ weighted wrt. D_t
  Compute LIFT values for r_t // ... from contingency matrix (Def. 33, p. 74)
  Let $D_{t+1}(x_i) := D_t(x_i) \cdot (\text{LIFT}_{D_t}(r_t, x_i))^{-1}$ for $i \in \{1, \ldots, n\}$.
end for
Output the set of rules $\{r_1, \ldots, r_k\}$ and their LIFTs.

may be considered if e.g., predictive performance is of higher importance than intuitive distributions. A variant tailored towards high predictive accuracy will be discussed and analyzed in chapter 5.

For mainly three reasons the following algorithm is presented only for the weighted relative accuracy (WRACC, Def. 20, p. 24). First, WRACC is the most popular utility function for subgroup discovery in practice. It is used as the default evaluation metric for subgroup discovery since the early systems EXPLORA (Klösgen, 1996) and MIDOS (Wrobel, 1997). Second, for this function theorem 5 (p. 59) provides a simple way of transforming the corresponding formal data mining problem into a classifier induction problem. This simplifies the evaluation in practice, because it allows to tackle the problem by using a common rule induction algorithm from any of the machine learning libraries. The third reason is, that it is sufficient to compare the presented approach to the reweighting schemes proposed for subgroup discovery so far (Lavrac et al., 2004b).

Description of the algorithm

The following paragraphs describe a straightforward extension of knowledge-based sampling for sequential subgroup discovery. To ease presentation, this algorithm is stated in terms of an example reweighting rather than sampling strategy, because the sampling step can be realized as discussed before, but adds additional complexity to the pseudo-code. The discussion will point out how sampling techniques can be substituted.

The *knowledge-based sampling for sequential discovery* algorithm (KBS-SD) depicted in algorithm 2 iteratively selects a single rule that corresponds to a subgroup with high WRACC, updates the distribution according to theorem 6, and selects the next rule based on the updated distribution. The function[3] $D_{t+1} : \mathcal{X} \rightarrow \mathbb{R}^+$ implicitly defined in iteration t by this strategy

[3] Again, the deterministic dependency of $\mathcal{Y}$ of $\mathcal{X}$ is not required, but eases notation.

for the complete instance space $\mathcal{X}$ is the result of transforming the most recent function D_t by applying theorem 6 to sample out the rule r_t that was discovered in iteration t. The reweighting strategy of the pseudo-code changes the weights of all available examples accordingly in each iteration; the weight of example (x, y) after iteration t is hence chosen as $D_{t+1}(x)$, following the discussion in subsection 3.4.2 (p. 66) on the relation between rejection sampling and example weights.

Theorem 5 (p. 59) implies that high predictive accuracy on stratified samples directly translates into high WRAcc on the original data. To utilize this simplification, the first lines of the algorithm compute a distribution D_1 by reweighting the training examples $\mathcal{E}$ with respect to Def. 31 (p. 56). Each of the subsequent loops induces a single rule with high predictive accuracy by calling a procedure RuleInduction(D_t, $\mathcal{E}$). This procedure is assumed to be capable of processing an example set $\mathcal{E}$ that is weighted with respect to a supplied function D_t. Each resulting rule characterizes a separate subgroup.

As KBS-SD computes distribution updates according to theorem 6, all constraints defined in Subsec. 4.3.1 hold. Constraint (4.3) implies that all subsequently defined distributions share the stratification property of D_1. It is mathematically equivalent to (i) compute the example weights with respect to (generally unstratified) distributions $D_1, \ldots, D_k$, which leaves the stratification step to the rule induction algorithm, and to (ii) integrate stratification into the definition of D_1, as the algorithm does, and to compute subsequent distributions based on this stratified version. In the latter case, the corresponding unstratified counterparts can simply be reconstructed by rescaling the class fractions, so that the original class priors are re-established. This corresponds to an inversion of the initial step of stratification.

To operationalize KBS-SD by sampling, we may build upon the knowledge-based rejection sampling (algorithm 1) presented in section 4.4.2. It was shown how to realize the transformation of D into D' as described by theorem 6 by constructing a procedure $EX_{D'}$ that utilizes another procedure EX_D. This idea can be applied recursively. Each $EX_{D_{t+1}}$ required in iteration $t+1$ can be realized by sampling from D_t utilizing the procedure EX_{D_t} established in iteration t. In this case, no example weights are required in algorithm 2; each iteration t starts with a step of (stratified) sampling based on procedure EX_{D_t}, followed by a step of inducing a rule and estimating Lifts on the sample. As a final step, it is required to prepare a procedure $EX_{D_{t+1}}$ to be used in the next iteration.

For predictive purposes, rules selected by KBS-SD annotated by their Lifts can be combined to an ensemble classifier applying any of the techniques discussed in section 2.6 (p. 32). All rules r_t are of the form $A^{(t)} \rightarrow C$, or $A^{(t)} \rightarrow \overline{C}$, respectively, if the property of interest is boolean[4], with each $A^{(i)}$ denoting an antecedent. The following definition eases notation for the prediction rules used in this chapter.

Definition 34 *The* Lift ratio of an example $x \in \mathcal{X}$ *for a rule* $(A \rightarrow C)$ *is defined as*

$$LR(A \rightarrow C, x) := \begin{cases} \frac{\text{Lift}(A \rightarrow C)}{\text{Lift}(A \rightarrow \overline{C})} & \text{, for } x \in \text{Ext}(A) \\ \frac{\text{Lift}(\overline{A} \rightarrow C)}{\text{Lift}(\overline{A} \rightarrow \overline{C})} & \text{, for } x \in \text{Ext}(\overline{A}) \end{cases}$$

Using this notation, an application of the hybrid NaïveBayes / logistic regression strategy for

[4]This restriction is only made for simplicity, as it is always possible to substitute one-against-all estimates. In such a case, the resulting probability estimates just need to be normalized appropriately.

combining predictions (cf. Subsec. 2.6.2, p. 34) yields

$$\hat{\beta}(x) = \frac{Pr_{D_0}(C)}{Pr_{D_0}(\overline{C})} \cdot \prod_{1 \leq t \leq k} LR_{D_t}((A^{(t)} \rightarrow C), x) \tag{4.14}$$

as an estimate for the odds ratios (eqn. (2.8), p. 35).

Estimating each LR with respect to the corresponding pdf D_t prevents poor approximations of the conditional target distribution in case of violated conditional independence (cf. Subsec. 2.6.3, p. 36). For descriptive settings, performance criteria of discovered rules evaluated with respect to the *original* distribution can be reported to end users, which is more intuitive; the estimates based on the altered pdfs D_t above may be used for making predictions, and for illustrating the individual impact of each rule in the context of an ensemble.

Eqn. (4.14) is not applicable whenever a selected rule has a precision of 1. This should not pose a problem, however; following the argument on page 88, fully explained subsets may simply be removed from further consideration.

In prediction scenarios, significance tests help to avoid overfitting, in a descriptive settings they avoid to report rules that could easily appear interesting just by chance. For simplicity, significance tests during rule selection are avoided in the subsequent experiments, but a similar effect is achieved by controlling the number of iterations k; rules are selected with respect to WRAcc, a metric proportional to coverage, so rules are not expected to overfit for small k. Alternatively, the binomial test function could be used in combination with a reasonable confidence threshold. Please note that precise significance tests require more efforts, i.e. to (i) evaluate on a hold-out set, which allows to utilize the confidence bounds for $q^{(0.5)}$ depicted in table 3.1 (p. 53), or to (ii) consider the expressiveness of the hypothesis space in terms of the VCdim (Def. 4, p. 15), respectively.

Properties of KBS-SD

The subsequent paragraphs discuss the proposed combination of rules to an ensemble classifier in the context of knowledge-based sampling. Obviously, the ensemble of rules described above makes *soft* predictions, while KBS-SD and the proposed knowledge-based rejection sampling algorithm are designed to "sample out" models that make *discrete* predictions, or define boolean assignments to partitions, respectively. The most crucial question is hence, whether the distributions defined by KBS-SD match the prediction strategy above. In other terms: Does the sampling procedure realize a step of sampling out the soft predictions of the ensemble, and do additional rules help to improve the predictions of the resulting ensemble?

First of all it should be mentioned, that the prediction rule above is a compromise between model complexity, computational complexity, and model accuracy. As discussed in section 2.6, there are many alternatives for combining model predictions for predictive data mining tasks. A precise (but naïve) combination of the k individual rule estimates based on Bayes' theorem requires exponentially (2^k) many estimates, one for each intersection of subsets. Another naïve approach, NaïveBayes, assumes conditional independence, which results in models that consider the performance of each rule separately. This model class is tractable and can easily be interpreted by human analysts. In practice the performance will usually be compromised by the independence assumption not being met, however. Please recall from the discussion in subsection 2.6.3 (p. 36) that a more precise logistic regression-like framework requires higher computational efforts, but in return, it yields a maximum likelihood weight vector as a tractable compromise between model complexity and accuracy. Computing the weights of rules sequentially, without reconsidering previously chosen weights, is a very cheap greedy approximation

strategy. In combination with i.i.d. data streams and single-pass algorithms (random permutation assumed) it has another appealing property worth being mentioned: It is possible to sample exactly from the target distributions without knowing the performances of rules, simply by utilizing rejection sampling-like algorithms as discussed above. Performances required for making predictions may be refined continuously based on subsequently read data. The robustness of this approach, but realized in terms of example weighting, will be exploited in chapter 6, where it is applied to data streams inhibiting concept drift.

For some kinds of tasks, other combination strategies than those discussed here may be preferable. As will be outlined at a later point in this section, they can easily be substituted, but at the price of requiring performance estimates *before* being able to select additional models.

The analysis starts with sets of conditionally independent rules. Clearly, this is the most desirable case: The individual contribution of each rule to the ensemble can be identified without considering the context of other rules, and the simple NAÏVEBAYES prediction rule yields optimal conditional class estimates. The following result shows, that this case is well supported by the algorithm. In detail it illustrates, that (i) the pdfs sequentially defined by KBS-SD do not interfere with conditional independence, and that (ii) we may always refer to the altered pdfs D_i when estimating LIFTs, or LIFT ratios, respectively, because for conditional independent rules the estimates are identical to the estimates with respect to D_0.

Proposition 6 *If a sequence of rules* $R := (r_1, \ldots, r_k)$ *is conditionally independent with respect to an initial distribution* D_0*, then we have*

$$(\forall r_i \in R)(\forall x \in \mathcal{X}) : \text{LIFT}_{D_i}(r_i, x) = \text{LIFT}_{D_0}(r_i, x),$$

where $D_1, \ldots, D_k$ *denote the pdfs sequentially defined by* KBS-SD *based on* R.

Proof
First, we observe that the stratified distribution D_1 is defined by rescaling D_0, changing the example weights of each class by a constant factor. Considering the definition of the LIFT (Def. 16, p. 22) we see, that this metric is invariant to changing class skews, because of the class prior in the denominator. To be precise, this property requires that the constraints defined in subsection 4.3.1 – except for maintaining the class priors – are met, that is, that we *only* change the class priors. This is given, so the invariance implies that the LIFTs of rules in R with respect to D_0 and D_1 are identical. It is hence sufficient to prove the equivalence of LIFTs under D_i, $1 < i \leq k$, to the LIFTs under D_1. Clearly, changing the class priors does not introduce conditional dependencies between rules, so the rules in R are also conditionally independent with respect to D_1.

The proof is shown for two rules $r_1 : A^{(1)} \to C$ and $r_2 : A^{(2)} \to C$ only; the case of more rules can be handled analogously. By construction of D_2 we have

$$\Pr_{D_2}[A_s] = \frac{\Pr_{D_1}[A_s]}{\text{LIFT}_{D_1}(A^{(1)} \to C)} \quad \text{and} \quad \Pr_{D_2}\left[A'_s\right] = \frac{\Pr_{D_1}[A'_s]}{\text{LIFT}_{D_1}(\overline{A}^{(1)} \to C)}$$

for each A_s and A'_s with $\text{Ext}(A_s) \subseteq \text{Ext}(A^{(1)} \wedge C)$ and $A'_s \subseteq \text{Ext}(\overline{A}^{(1)} \wedge C)$, respectively.

Now we show, that the probability of $A^{(2)} \wedge C$ under D_2 is identical to the probability of the same event under D_1, if $A^{(2)}$ is conditionally independent of $A^{(1)}$ under D_1:

$$\begin{aligned}
\Pr_{D_2}\left[A^{(2)}, C\right] &= \Pr_{D_2}\left[A^{(1)}, A^{(2)}, C\right] + \Pr_{D_2}\left[\overline{A}^{(1)}, A^{(2)}, C\right] \\
&= \frac{\Pr_{D_1}\left[A^{(1)}, A^{(2)}, C\right]}{\text{LIFT}_{D_1}(A^{(1)} \to C)} + \frac{\Pr_{D_1}\left[\overline{A}^{(1)}, A^{(2)}, C\right]}{\text{LIFT}_{D_1}(\overline{A}^{(1)} \to C)}
\end{aligned}$$

Exploiting the conditional independence and $\text{LIFT}(A \to C) = \Pr[A \mid C] / \Pr[A]$ we get

$$\begin{aligned}
\Pr_{D_2}\left[A^{(2)}, C\right] &= \frac{\Pr_{D_1}\left[A^{(1)} \mid C\right] \cdot \Pr_{D_1}\left[A^{(2)}, C\right]}{\text{LIFT}_{D_1}(A^{(1)} \to C)} + \frac{\Pr_{D_1}\left[\overline{A}^{(1)} \mid C\right] \cdot \Pr_{D_1}\left[A^{(2)}, C\right]}{\text{LIFT}_{D_1}(\overline{A}^{(1)} \to C)} \\
&= \Pr_{D_1}\left[A^{(2)}, C\right] \cdot \left(\frac{\Pr_{D_1}\left[A^{(1)} \mid C\right]}{\Pr_{D_1}\left[A^{(1)} \mid C\right] / \Pr_{D_1}\left[A^{(1)}\right]} + \frac{\Pr_{D_1}\left[\overline{A}^{(1)} \mid C\right]}{\Pr_{D_1}\left[\overline{A}^{(1)} \mid C\right] / \Pr_{D_1}\left[\overline{A}^{(1)}\right]} \right) \\
&= \Pr_{D_1}\left[A^{(2)}, C\right] \cdot \left(\Pr_{D_1}\left[A^{(1)}\right] + \Pr_{D_1}\left[\overline{A}^{(1)}\right] \right) = \Pr_{D_1}\left[A^{(2)}, C\right].
\end{aligned}$$

By substituting other antecedents $A^{(2)}$ and conclusions C we can show, that the remaining entries of the contingency matrix (e.g., $\Pr\left[\overline{A}^{(2)}, \overline{C}\right]$) are also identical under D_1 and D_2. This implies equal LIFTs of r_2 under both distributions. □

Conditionally dependent rules are much less convenient, but should still be supported well by data mining algorithms. The following results describe properties of KBS-SD if conditional independence is lacking. First, Prop. 5 is adapted to ensembles of rules reported by KBS-SD in combination with prediction rule eqn. (4.14) from page 91.

Definition 35 *A subset $S \subseteq \mathcal{X}$ is called an* atomic subset *with respect to a set of classification rules R, if for each $(A \to C) \in R$ either $S \subseteq \text{Ext}(A)$ or $S \cap \text{Ext}(A) = \emptyset$.*

Atomic subsets have the property, that the prediction rule eqn. (4.14) yields the same estimate for each of its instances.

Theorem 7 *Let $R = \{r_i \mid 1 \leq i \leq k\}$ be a set of rules output by the* KBS-SD *algorithm. Let further antecedent S refer to an* atomic subset $\text{Ext}(S)$ *with respect to R, and let $\hat{\beta}(S)$ denote the corresponding odds ratio estimate of prediction rule eqn.* (4.14)*. If the (true) odds ratio in $\text{Ext}(S)$ under D for a target class C is $s \cdot \hat{\beta}(S)$ for any $s \in \mathbb{R}^+$, then under the final pdf D_{k+1} constructed by* KBS-SD *the odds ratio of $\text{Ext}(S)$ is*

$$\frac{\Pr_{D_{k+1}}[C \mid S]}{\Pr_{D_{k+1}}\left[\overline{C} \mid S\right]} = s.$$

Proof
We start with an analysis of prediction rule eqn. (4.14). The predicted odds ratio $\hat{\beta}(x)$ for an example $x \in \mathcal{X}$ is:

$$\hat{\beta}(x) = \frac{\Pr_{D_0}(C)}{\Pr_{D_0}(\overline{C})} \cdot \prod_{1 \leq i < k} \text{LR}_{D_i}(r_i, x). \tag{4.15}$$

For notational simplicity we assume without loss of generality that the antecedents of all rules $r_i : A^{(i)} \to C$ *apply* for each $x \in S$. In this case we may substitute

$$\text{LR}_{D_i}(r_i, x) = \frac{\text{LIFT}_{D_i}(A^{(i)} \to C)}{\text{LIFT}_{D_i}(A^{(i)} \to \overline{C})}$$

in eqn. (4.15) to receive a product of fractions. Let

$$\gamma^+(S) := \Pr_{D_0}(C) \cdot \prod_{1 \le i < k} \text{LIFT}_{D_i}(A^{(i)} \to C)$$

denote the product of all the resulting enumerators, and

$$\gamma^-(S) := \Pr_{D_0}(\overline{C}) \cdot \prod_{1 \le i < k} \text{LIFT}_{D_i}(A^{(i)} \to \overline{C})$$

be the product of denominators.

The algorithm starts with a step of stratification. Up to a constant factor this is equivalent to dividing the initial weights of positives by $\Pr_{D_0}(C)$ and the weights of negatives by $\Pr_{D_0}(\overline{C})$.

In each subsequent iteration i the weight of all positive examples in $\text{Ext}(S)$ are divided by $\text{LIFT}_{D_i}(A^{(i)} \to C)$, while weights of negatives are divided by $\text{LIFT}_{D_i}(A^{(i)} \to \overline{C})$. For a constant $c > 0$, the final weight of each positive example in $\text{Ext}(S)$ is hence $c/\gamma^+(S)$, while the weight of each negative example is $c/\gamma^-(S)$.

From $\beta(S) = \gamma^+(S)/\gamma^-(S)$ we can conclude that the odds ratio in $\text{Ext}(S)$ changes from D to D_{k+1} by a factor of $(\beta(S))^{-1}$:

$$\frac{\Pr_{D_{k+1}}[C \mid S]}{\Pr_{D_{k+1}}[\overline{C} \mid S]} = \frac{\Pr_{D_0}[C \mid S]/\gamma^+(S)}{\Pr_{D_0}[\overline{C} \mid S]/\gamma^-(S)} = \frac{\Pr_{D_0}[C \mid S]}{\Pr_{D_0}[\overline{C} \mid S]} \cdot \left(\hat{\beta}(S)\right)^{-1}$$

In particular, an odds ratio of $s \cdot \hat{\beta}(S)$ under D translates into an odds ratio of s. □

As this result shows, the reweighting scheme of KBS-SD matches the proposed prediction rule. In each iteration the prediction rule is augmented by factors correcting the odds ratio estimates. The result of the reweighting step is a corresponding reduction of the odds ratios; for each atomic subset they are divided by the corresponding factor. As a result, the odds ratios of atomic subsets after reweighting reflect the residuals of the model in multiplicative terms. In turn, subsets for which the estimates of the model are correct can be recognized by having odds ratios of 1 after reweighting. This implies equally likely classes. The goal of KBS-SD in sampling scenarios can hence be formulated as to sample out all correlations that may be described in terms of probabilistic classification rules, until each subset covered by any of the rule candidates is stratified. If rule antecedents correspond to subsets that contain only a single class, then it is optimal to make deterministic predictions, regardless of the predictions of other rules; the total weight of such sets should be reduced to 0 in the subsequently defined pdf, as for sequential covering.

Before addressing the question in which sense KBS-SD samples out the soft predictions of its rule ensembles, it should be clarified why selecting rules by WRACC is reasonable, although the reweighting scheme and the prediction rule both refer to the LIFT of rules. The argument follows the discussion on page 86, but can be simplified for stratified data.

Proposition 7 *For stratified example sets and boolean target attributes any rule that maximizes* ACC *or* WRACC *also maximizes the* LIFT *weighted by coverage.*

Proof
The equivalence between maximizing ACC and WRACC for stratified data sets has been shown before (cf. theorem 5), so we may refer to ACC and hence assume that the rule predicts the

opposite class if does not apply. ACC is defined as

$$\begin{aligned}\text{ACC}(A \to C) &= \Pr[A, C] + \Pr\left[\overline{A}, \overline{C}\right] = \Pr[A] \cdot \Pr[C \mid A] + \Pr\left[\overline{A}\right] \cdot \Pr\left[\overline{C} \mid \overline{A}\right] \\ &= \Pr[A] \cdot \Pr[C] \cdot \text{LIFT}(A \to C) + \Pr\left[\overline{A}\right] \cdot \Pr\left[\overline{C}\right] \cdot \text{LIFT}(\overline{A} \to \overline{C}) \\ &= 1/2\left(\Pr[A] \cdot \text{LIFT}(A \to C) + \Pr\left[\overline{A}\right] \cdot \text{LIFT}(\overline{A} \to \overline{C})\right).\end{aligned}$$

The constant factor of $1/2$ can be ignored in the context of optimization. The remaining term is the average weighted LIFT of the rule. □

Clearly, since the LIFT is proportional to the precision, the odds ratios (and LIFT ratios) increase monotonically with the LIFT of covered subsets. Hence, KBS-SD selects rules by their average deviation from expectation (that is, form stratification), and corrects the corresponding residuals of the ensemble by adapting the predictions accordingly. A detailed analysis of the predictive performance of a KBS-SD variant will be provided in the next chapter.

Regarding sample construction with respect to the soft predictions of the ensemble, we should first note that the concept of "sampling out" probabilistic predictions is not supported by the constraints (4.1)-(4.7) (p. 73). We will hence briefly discuss a straightforward generalization of the pdf defined by theorem 6 (p. 74) for soft predictions, and then compare it to KBS-SD.

Definition 36 *Let* $D : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^+$ *denote a target pdf, and let the prior knowledge* θ *be associated to a function*

$$\widehat{\Pr}(y \mid x, \theta) \approx \Pr_D[y \mid x] = \frac{D(x, y)}{\sum_{y \in \mathcal{Y}} D(x, y)},$$

estimating the conditional distribution of the label for each example $x \in \mathcal{X}$*. For class priors* $\Pr_D(y)$*,* $y \in \mathcal{Y}$*, the* estimated LIFT *with respect to prior knowledge* θ *is defined as*

$$\widehat{\text{LIFT}}_D(x \to y \mid \theta) := \frac{\widehat{\Pr}(y \mid x, \theta)}{\Pr_D(y)}.$$

Based on the estimated LIFT we can generalize the transformation described in theorem 6 by defining

$$D'(x, y) = D(x, y) \cdot \left(\widehat{\text{LIFT}}(x \to y \mid \theta)\right)^{-1} \tag{4.16}$$

Despite its syntactical similarity, this rule shares another important property with the transformation rule used so far.

Proposition 8 *For a label* $\mathcal{Y} = \{0, 1\}$ *the predictions* $h(x) := \widehat{\Pr}(y = 1 \mid x, \theta) \in [0, 1]$ *for class* $y = 1$ *are uncorrelated with* $\mathcal{Y}$ *with respect to* D' *defined by eqn.* (4.16)*, formally*

$$\mathbb{E}_{D'}[y \cdot h(x)] = \mathbb{E}_{D'}[y] \cdot \mathbb{E}_{D'}[h(x)],$$

as long as D *and* D' *share the same class priors.*

Proof
It is sufficient to show that the expected prediction $\mathbb{E}_{D'}[h(x) \mid y]$ of the ensemble when conditioning on either of the classes is identical. In this case, the proposition holds because

$$\begin{aligned}\mathbb{E}_{D'}[y \cdot h(x)] &= \Pr_{D'}(y = 0) \cdot 0 + \Pr_{D'}(y = 1) \cdot \mathbb{E}_{D'}[h(x) \mid y = 1] \\ &= \Pr_{D'}(y = 1) \cdot \mathbb{E}_{D'}[h(x)] \;=\; \mathbb{E}_{D'}[y] \cdot \mathbb{E}_{D'}[h(x)]\end{aligned}$$

It can be seen that the expected predictions for the correct class are equal to the corresponding class prior when plugging the definition of the empirical LIFT into eqn. (4.16):

$$D'(x,y) \;=\; D(x,y)\cdot\left(\widehat{\text{LIFT}}(x \to y \mid \theta)\right)^{-1} \;=\; D(x,y)\cdot\frac{Pr_D(y)}{\widehat{Pr}(y \mid x,\theta)}$$

The estimates cancel out when switching from D' to D conditioning on any class $y^* \in \mathcal{Y}$:

$$\begin{aligned}
\mathbb{E}_{D'}\left[\widehat{Pr}(y^* \mid x,\theta) \mid y^*\right] &= (Pr_{D'}(y^*))^{-1}\cdot\int_{D'} I[y=y^*]\cdot\widehat{Pr}(y^* \mid x,\theta)\,dx\,dy \\
&= (Pr_{D'}(y^*))^{-1}\cdot\int_{D} I[y=y^*]\cdot\widehat{Pr}(y^* \mid x,\theta)\cdot\frac{Pr_D[y^*]}{\widehat{Pr}(y^* \mid x,\theta)}\,dx\,dy \\
&= (Pr_{D'}(y^*))^{-1}\cdot\int_{D} I[y=y^*]\cdot Pr_D(y^*)\,dx\,dy \\
&= \frac{Pr_D[y^*]}{Pr_{D'}[y^*]}\cdot Pr_D[y^*]\,.
\end{aligned}$$

Since class priors are assumed to be equal under both pdfs this implies

$$\begin{aligned}
&\mathbb{E}_{D'}\left[\widehat{Pr}(y \mid x,\theta) \mid y^*\right] = \begin{cases} Pr_{D'}(y^*) & \text{, for } y = y^* \\ 1 - Pr_{D'}(y^*) & \text{, for } y \neq y^* \end{cases} \\
\Rightarrow\quad &(\forall y^* \in \mathcal{Y}) : \mathbb{E}_{D'}[h(x) \mid y^*] = Pr_{D'}(y=1),
\end{aligned}$$

as required. □

This generalized pdf can be utilized when rules and prior knowledge were combined to a conditional class estimator function in arbitrarily complex ways. This allows e.g. to replace the NAÏVEBAYES-like strategy by logistic regression. To sample out such prior probabilistic knowledge, for example from a data stream, it is sufficient to subsample each example x with a probability of $(\widehat{\text{LIFT}}(x \to y \mid \theta))^{-1}$. This strategy is surprisingly simple and well suited to apply rejection sampling.

However, this technique also has some disadvantages. First of all, when relying on the predictions of an arbitrary soft classifier, the class priors will not necessarily remain after transforming the pdf. Hence, sampling orthogonal to expectation requires additional efforts in this setting. Second, the strategy lacks a pleasant property of KBS-SD, namely to be able to sample precisely without requiring any probability estimates. For example, when using the partitions induced by a decision tree for knowledge-based sampling, the corresponding conditional class distributions at the leaves based on a training sample are usually biased. KBS-SD allows to continuously refine all LIFT estimates on the fly as new examples are sampled. The reason is, that it defines and operationalizes pdfs with respect to the correct LIFTs, which in section 4.4.2 has been shown to work even if these LIFTs are not known precisely. Hence, any LIFT estimate can be refined independently, without affecting performances of other models. This is different when defining pdfs based on arbitrary soft classifier estimates; subsequent pdfs that are constructed with respect to eqn. (4.16) depend on the empirical estimates, so changing an estimate for predictive purposes affects all subsequently defined pdfs and invalidates other estimates.

Proposition 8 describes a desirable property for sampling out soft classifier predictions in general, namely that for samples the target should be uncorrelated with predictions. When using the reweighting strategy of KBS-SD, all pdfs are stratified, so the prerequisite of the proposition

is met. The combination of reweighting scheme and rule combination proposed for KBS-SD is not identical with, but still very similar to eqn. (4.16). Enumerator and denominator of the product in (4.14) (p. 91) approximate the LIFTs of positive and of negative examples, respectively; if θ denotes prior knowledge in the form of an ensemble of probabilistic rules with antecedents $\{A^{(1)}, \ldots A^{(k)}\}$, then we have

$$\widehat{\text{LIFT}}(x \to y \mid \theta) \approx \prod_{1 \le i < k} \text{LIFT}_{D_i}((A^{(i)} \to y), x).$$

As shown in the proof of theorem 7, KBS-SD weights examples inverse proportionally to these estimates, and hence, referring to Prop. 8, approximately yields samples that are independent of ensemble predictions. In fact, experiments by the author revealed no significant difference between the KBS-SD strategy and the empirical LIFT reweighting above on data sets that fit into main memory, when (i) using an example reweighting approach and (ii) without continuously refining LIFT estimates. For this reason only results for the simpler sequential KBS-SD algorithm are reported in this chapter.

4.5.2. Related work: CN2-SD

In the next section KBS-SD will be compared to the only two other reweighting strategies suggested in the subgroup discovery literature so far; both were published in the context of the CN2-SD algorithm (Lavrac et al., 2002a; Lavrac et al., 2004b).

The idea of CN2-SD is to adapt the classifier CN2 (Clark & Niblett, 1989) to the task of subgroup discovery, by changing the internal rule selection metric to WRACC. CN2 sequentially learns sets of classification rules, one rule per iteration, and removes the covered subsets each time before proceeding. Rule candidates are constructed based on a beam search-like strategy, and are selected in terms of a significance test metric. The improved variant (Clark & Boswell, 1991) referred to by (Lavrac et al., 2004b) removes only the *correctly* covered example in each iteration. A rule hence has to describe subsets in which the predicted class passes the significance test in the presence of all examples previously covered only by mistake. The result is an unordered set of rules, with each rule being annotated by the absolute number of covered positive and negative examples. The prediction strategy is equivalent to a weighted voting scheme (cf. subsection 2.6.1). The total prediction is an average of the individual class distributions (each normalized to sum up to 1) predicted by applicable rules. The absolute coverage of each such rule at induction time is chosen as its voting weight.

As a first adaptation, CN2-SD uses the metric WRACC to select rule candidates. It also yields unordered sets of rules, but combines them in terms of a uniform weighting scheme, rather than by using the weighted scheme of CN2. A second adaptation is, not to remove covered examples, but to reweight them. The reweighting strategy is different from that used by KBS-SD: After a positive example e has been covered by i rules, its new weight is computed as either

additive update: $w_i(e) := \frac{1}{i+1}$ or

multiplicative update: $w_i(e) := \gamma^i$ for a given parameter $\gamma \in [0, 1]$.

This step is meant to make the sequential procedure more suitable for subgroup discovery. However, there is no theoretical justification for this kind of reweighting, so this method does neither yield intuitive distributions, nor does it correspond to a framework that allows to handle prior knowledge and previously found patterns homogeneously.

A drawback in predictive settings is the lack of a theoretical foundation allowing to give guarantees, e.g., that additional rules help to improve the soft predictions of rule ensembles. It is easily seen that both reweighting schemes do not de-correlate the contingency matrix. This means that in general each subgroup – not necessarily with the same predicted label – will still receive a positive utility after reweighting. Consequently, multiple occurrences of rules in the rule set are allowed. Please note that a step of reweighting is necessary after each iteration of rule discovery, even if a rule has been discovered multiple times before; without reweighting, a subsequent iteration would yield the same rule again. The lacking correspondence between predictions and example weights do not seem to foster monotone behavior when incorporating additional rules.

4.6. Experiments

4.6.1. Implemented operators

To evaluate knowledge-based sampling for subgroup discovery three algorithms have been implemented in the learning environment YALE[5] (Mierswa et al., 2006) by the author of this thesis. The first of these is the knowledge-based sampling algorithm for sequential discovery (KBS-SD) shown in algorithm 2 on page 89. The implementation[6] uses example reweighting for training data that fits into main memory.

Additionally, two variants of an operator with the name *subgroup discovery rule set induction* (SDRI) have been implemented. The implementation exploits theorem 5, that is, the fact that after stratification *any* common rule induction algorithm can be used to identify rules maximizing the WRACC. One of the two reweighting strategies of CN2-SD can be specified as a parameter. The specific rule induction algorithm can be chosen from the library of classifiers supported by YALE. Hence, SDRI basically subsumes the two CN2-SD variants sketched above. The rule induction algorithm CONJUNCTIVERULE, part of the WEKA learning environment (Witten & Frank, 2000), was chosen as an embedded learner for KBS-SD and SDRI in the experiment section. CONJUNCTIVERULE iteratively constructs the body of rules comparing the information gain of each candidate literal, and it prunes rules applying the reduced error pruning heuristic.

The SDRI variant that applies CONJUNCTIVERULE on stratified samples after additive updates is referred to as SDRI$^+$, the one with multiplicative updates as SDRI*. The class explicitly predicted by a rule is defined to be the positive one, as fixing one of the classes as positive gave worse experimental results. This is coherent with the CN2 algorithm. SDRI combines individual rule predictions applying the same strategy as CN2-SD, a uniform voting scheme averaging soft predictions.

4.6.2. Objectives of the experiments

The goal of knowledge-based sampling is to support data mining in the presence of prior knowledge. Sequential subgroup discovery utilizes this form of sampling (or reweighting) in an iterative fashion to discover *sets* of rules. Sequentially changing the distribution underlying the data means to augment the prior knowledge by each new discovery before mining the next subgroup. Clearly, the sequential discovery procedure is more demanding than "sampling out" any static form of prior knowledge just once, before the data mining step. From a technical point

[5] `http://yale.sf.net/`

[6] `operator name: BayesianBoosting, deactivate the parameter allow_marginal_skews, activate rescale_label_priors`

of view, there is no substantial difference between any domain-specific prior knowledge and an ensemble of rules produced by early iterations of sequential subgroup discovery. For this reason, the subsequent experiments are only concerned with sequential subgroup discovery without any user-specified domain knowledge.

Two potential discovery tasks have been addressed in this chapter, descriptive and predictive subgroup discovery. The primary goal of subgroup discovery in predictive settings is to find a set of rules that characterize a target variable well. In more formal terms, the probabilistic classifiers built from sets of discovered rules should be accurate. This property is commonly evaluated in terms of the area under the ROC curve metric (AUC), see section 2.5.

A further desirable property for both predictive and descriptive tasks is, that the reported sets of rules are diverse; similar rules are neither interesting nor do they significantly improve the predictive performance (cf. p.33). In particular, we do not want the rule sets to contain any duplicates.

Moreover, the learning curves, e.g. measured in terms of the predictive AUC, should increase monotonically with an increasing number of rules. The reason is, that any decrease in predictive performance is a strong indicator of a false discovery, or of false performance estimates, respectively. For now, this property may just be considered to be derived from intuition; a theoretical investigation of the connections between performances of individual base models and corresponding changes in the AUC will be provided in the next chapter.

An important property of learning procedures in general is robustness to minor changes in the data. In predictive settings, using unstable learners may cause additional efforts, e.g. bagging (cf. subsection 2.6.1), in order to achieve reliable predictions; in descriptive settings, instability compromises the quality of studies in general. The robustness of ensembles will be evaluated based on the standard deviation of predictive AUC performances.

Finally, for descriptive tasks we want the resulting set of models to be small and understandable. The former is evaluated by reporting the number of discovered rules throughout the experiment section, the latter is assumed to hold due to the choice of classification rules as the class of models. For reporting results to end users, KBS-SD rule performances based on the initial distribution are appropriate. The corresponding averages of the metrics COV and WRACC for KBS-SD ensembles will be evaluated in the experiments. Compared to approaches that benefit from reporting highly overlapping sets of rules, KBS-SD focuses on novel patterns in each iteration; intuitively, this should reduce the average COV and WRACC, while increasing the diversity and predictive performance at the same time.

4.6.3. Results

The proposed idea of sequential sampling-based subgroup discovery has been evaluated on five data sets taken from the UCI Machine Learning Library (Blake & Merz, 1998) and a 10K sample taken from the KDD Cup 2004 Quantum Physics data set[7]. All data sets have boolean target attributes. Their size, the fraction of the minority class, and the number of discrete and continuous attributes are listed in table 4.1. The issue of scalability has been discussed before and can be decoupled from the question of whether changing the distribution and combining rules to an ensemble as proposed helps to increase the quality of data mining results. All well-suited publicly available benchmark data set known to the author of this thesis easily fit into main memory, so KBS-SD was consequently used in combination with example reweighting. This is more demanding, because all performances are estimated on the same sample (but for different weight distributions), which increases the risk of false discoveries and overfitting.

[7] http://kodiak.cs.cornell.edu/kddcup/

Dataset	Examples	Discrete	Continuous	Minority
KDD Cup	10.000	–	71	50.0%
Adult	32.562	8	6	24.1%
Ionosphere	351	–	34	35.8%
Credit Domain	690	6	9	44.5%
Voting-Records	435	16	–	38.6%
Mushrooms	8.124	22	–	48.2%

Table 4.1.: Data characteristics: size, number discrete/continuous attributes, class skew

Figure 4.2 to 4.7 show how the AUC performance changes with an increasing number of iterations. All values have been estimated by 10fold cross-validation[8]. The columns **k** and **auc** in table 4.2 list the average performances of rule sets according to the cross-validation experiments for the empirically best choice of k. For the KDD Cup data and the adult data set the number of iterations were constrained. To further evaluate the differences between the algorithms, another rule set was induced for each variant, using the same value of parameter k. The training set was also used for evaluation in this case, as common for descriptive learning tasks. Table 4.2 shows the resulting average coverages (**cov**) and average weighted relative accuracies (**wracc**). The ROC filter for rule sets based on the convex hull (discussed in section 4.2) was applied to both SDRI variants, denoted as RF in table 4.2.

The column **div** reflects the diversities of rule sets. The entropy of predictions is an appropriate measure for diversity of classifier ensembles in general (Cunningham & Carney, 2000). Each rule can be considered to predict the conditional distribution of the target, given whether it is applicable or not. The diversity was computed as the average entropy (see p. 33) over the complete example set after removing multiple occurrences of rules.

Throughout figures 4.2 to 4.7 the KBS-SD algorithm outperforms SDRI with both reweighting strategies, while none of the SDRI variants is clearly superior to the other one. In figure 4.2, all three algorithms manage to find useful rules repeatedly. SDRI$^+$ performs best for sets of 3 to 6 rules, but for larger rule sets and for any other data set and number of iterations KBS-SD is superior. In figures 4.3 to 4.5, KBS-SD improves the AUC much quicker than SDRI, although for the smallest data set (Fig. 4.4) it overfits after the 3$^{\text{rd}}$ iteration. For the credit domain data (Fig. 4.5) the AUC values of the SDRI rule sets improve non-monotonically. An inspection of the rule sets revealed many duplicates. For the voting-records (figure 4.6), SDRI effectively finds just 2 useful rules with both reweighting strategies, which improves the AUC by about 1% compared to the first iteration. KBS-SD selects 6 rules and improves the AUC by about 4%. Finally, in the experiment shown in figure 4.7, KBS-SD reaches 100% AUC with just 12 rules, while SDRI does not manage to improve over the performance of the first rule at all. For the smaller data sets, the ROC filter basically just removes duplicates from the rule sets, which has a marginal impact on the performance metrics. For the large data sets, the filter prunes the rule set at the price of a reduction in both AUC performance and diversity.

Finally, it is interesting to note that the KBS-SD rule sets outperform those of the SDRI variants in terms of predictive performance, although they often have a smaller coverage and WRAcc. Moreover, for all data sets the KBS-SD rule sets have the highest diversity, but according to the standard deviation of the AUC performance (column "$\pm$") they are nevertheless most robust to minor changes in the data. These changes are the result of sub-sampling during

[8]For SDRI* results are reported for the empirically best γ from the candidate set $\{.1, .2, .3, .4, .5, .6, .7, .8, .9, .95\}$.

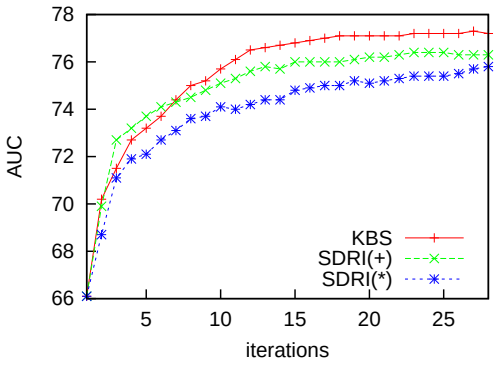


Figure 4.2.: KDD Cup

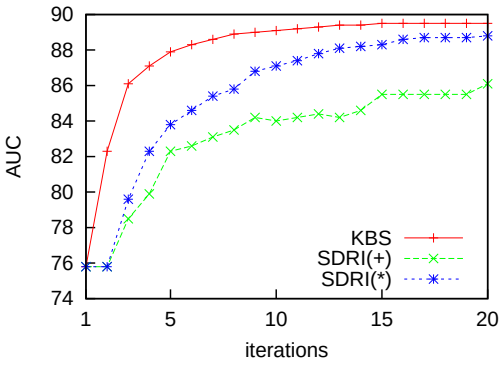


Figure 4.3.: Adult

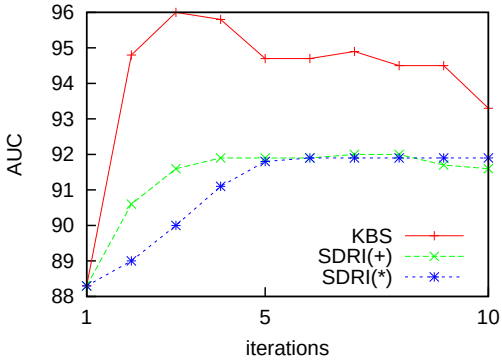


Figure 4.4.: Ionosphere

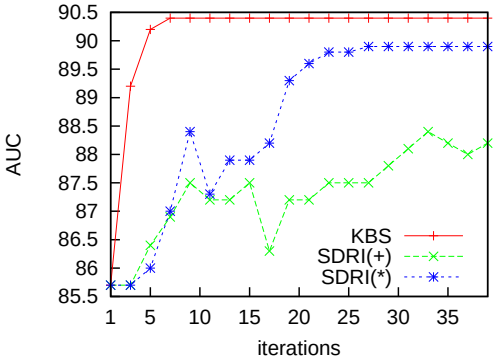


Figure 4.5.: Credit Domain

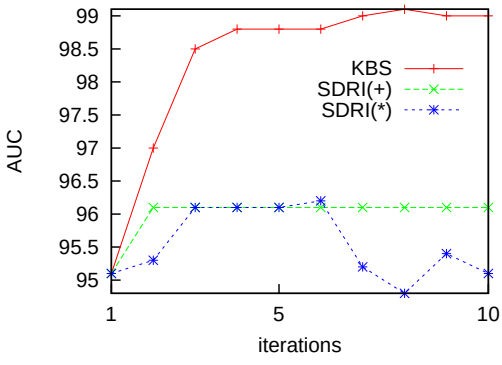


Figure 4.6.: Voting-Records

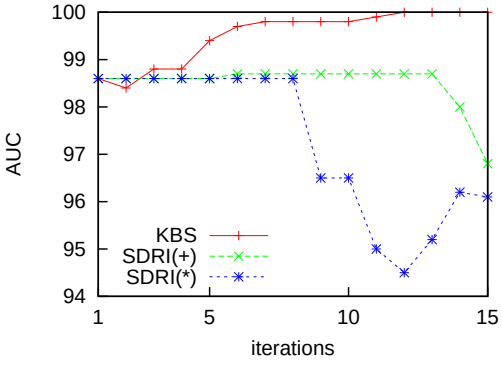


Figure 4.7.: Mushrooms

algorithm	k	auc	±	cov	wracc	div
KBS-SD	15	76.8	1.2	38.6%	0.023	0.972
SDRI$^+$	15	76.0	1.9	50.5%	0.054	0.932
SDRI$^+$, RF	12	74.3	2.0	50.0%	0.056	0.928
SDRI*	15	74.8	2.1	42.7%	0.071	0.917
SDRI*, RF	8	74.2	2.1	44.7%	0.074	0.914

KDD Cup

algorithm	k	auc	±	cov	wracc	div
KBS-SD	15	89.5	1.1	48.8%	0.036	0.739
SDRI$^+$	20	86.1	2.8	47.0%	0.053	0.703
SDRI$^+$, RF	7	83.6	2.6	49.8%	0.055	0.703
SDRI*	20	88.8	1.5	43.5%	0.051	0.719
SDRI*, RF	7	84.9	1.4	39.6%	0.050	0.706

Adult

algorithm	k	auc	±	cov	wracc	div
KBS-SD	3	96.0	3.0	42.7%	0.121	0.669
SDRI$^+$	7	92.0	7.4	37.6%	0.120	0.643
SDRI$^+$, RF	4	91.7	7.0	35.3%	0.120	0.643
SDRI*	6	91.9	7.3	60.1%	0.123	0.652
SDRI*, RF	3	91.0	6.7	40.6%	0.119	0.652

Ionosphere

algorithm	k	auc	±	cov	wracc	div
KBS-SD	7	90.4	3.4	42.2%	0.057	0.893
SDRI$^+$	31	88.4	4.2	56.8%	0.156	0.796
SDRI$^+$, RF	3	87.0	5.3	66.9%	0.139	0.668
SDRI*	27	89.9	4.0	55.8%	0.164	0.739
SDRI*, RF	2	85.7	5.3	66.9%	0.139	0.668

Credit Domain

algorithm	k	auc	±	cov	wracc	div
KBS-SD	8	99.1	1.0	46.2%	0.142	0.615
SDRI$^+$	2	96.1	2.0	50.0%	0.215	0.244
SDRI$^+$, RF	2	96.1	2.0	49.8%	0.215	0.244
SDRI*	6	96.2	2.0	50.3%	0.214	0.244
SDRI*, RF	2	96.4	1.9	51.4%	0.216	0.254

Voting-Records

algorithm	k	auc	±	cov	wracc	div
KBS-SD	12	100	0.0	69.5%	0.086	0.843
SDRI$^+$	6	98.7	0.2	43.4%	0.195	0.470
SDRI$^+$, RF	1	98.6	0.1	43.4%	0.195	0.470
SDRI*	1	98.6	0.1	43.4%	0.195	0.470
SDRI*, RF	1	98.6	0.1	43.4%	0.195	0.470

Mushrooms

Table 4.2.: Performance of different subgroup discovery algorithms.

the cross-validation procedure.

The interested reader may want to refer to a recent publication containing further related experiments. Subsequent to the publication (Scholz, 2005e) on which large parts of this chapter are based, and given the knowledge-based rejection sampling algorithm by the author of this thesis, Dach (2006) successfully combined the generic sequential sampling algorithm (GSS) by Scheffer and Wrobel (2002) (cf. subsection 3.3.2) with KBS-SD for sequential subgroup discovery. The work[9] contains an empirical study of the benefits and drawbacks of different utility functions. Further, the difference between example weighting and rejection sampling was compared empirically. As mentioned earlier, even the largest publicly available benchmark data sets suited for subgroup discovery easily fit into main memory. For such small data sets, Dach (2006) reports a reduced sample complexity of GSS, a significant gain in runtime, and a small advantage in terms of predictive performance when using an example reweighting technique rather than rejection sampling. For this kind of experiment, GSS was made capable of incorporating example weights. From all the functions in the $Q^{(\alpha)}$ family, the binomial test function performed best in predictive settings, but, comparing to WRAcc, GSS required much larger samples for this choice. WRAcc performed comparably well, until the ensembles reached a certain level of saturation. As expected, the binomial test function was more robust against overfitting, which mainly affects later iterations of KBS-SD.

4.7. A connection to local pattern mining

Despite its applicability to sequentially mine diverse rules (Scholz, 2005e), the presented sampling technique has originally been proposed in the context of mining *local patterns* (Scholz, 2005b). A framework for this tasks has recently been established in two targeted workshops (Hand et al., 2002; Morik et al., 2005), by collecting and integrating definitions from the statistical and machine learning community. Patterns are defined as subsets of an instance space that show an unusually high density. A consensus, first suggested by Hand (2002), distinguishes local patterns from global patterns and from noise. Global patterns are easily recognized, because their main characteristic is that they show as "high distortions" of the distribution underlying the data. Typically a model of low complexity (e.g., having a low VCdim, p. 15) suffices to describe global patterns. On the other side, local patterns need to be distinguished from random noise. Random fluctuations and noise may easily lead to false discoveries, which are not supported by the unknown distribution underlying the data, but are only valid by chance. The characteristics of local patterns are, that they either cover only a small subset of the overall population, that shows deviations from the predictions of a global model, or that they are of higher complexity than the global model, and hence require hypotheses from a richer space to be captured. As illustrated by the experiments in the last section, the KBS-SD algorithm allows to find a small set of diverse rules. Each of them can be regarded as (covering) a local pattern. Finding deviations from a user-given or previously trained global model is explicitly addressed by this technique. The risk of false discoveries due to noise and statistical fluctuations can be controlled by (i) employing the binomial test function (p. 23) in combination with a conservative minimal confidence level, or by (ii) considering the confidence bounds discussed in subsection 3.3.2 (p. 51) for resulting samples.

An approach related to knowledge-based sampling for rule discovery is described by Morik (2002). Global and local patterns are discovered by a learner that selects rules in accordance to a user-specified utility function and threshold. In a presented application, removing all examples

[9] The described work is a master's thesis supervised by the author of this thesis.

that were correctly covered by a global model learned in the first step, and changing to a finer grained hypothesis space in the second, allowed to find global and local patterns in an iterative fashion. Three possible adaptations for the second learning step were proposed: Choosing a larger hypothesis space by weakening the syntactical restrictions on rules, increasing the dimensionality of examples by adding attributes, or changing the overall search strategy in favor of a more exhaustive one.

Finding a global model first, with a set of coarse hypotheses, is an idea shared by the approach presented in this chapter. The main difference lies in the preprocessing step for finding local patterns in the next run. Iteratively removing examples covered by a set of rules follows a classical sequential covering approach. Each example is *explained* once in this setting, and can be removed from further consideration. This is reasonable for "strong" rules, that is, for rules with a precision close to 100%. For weak rules this is not attractive, because it is not possible to correct the false positive errors of rules in subsequent iterations.

In this thesis, patterns are considered to be reflected by biased class distributions. Thus, finding patterns goes beyond covering examples. The classical covering approach fails to detect several kinds of overlapping patterns, especially global patterns covering local ones. An example is to detect that the cancer rate of smokers is high (global pattern), but also, that smokers of a specific strong cigarette brand have an even higher cancer rate. Following the KBS approach, covered examples are not removed, but subsequent learners are "made aware" of mining local patterns under the presence of these strong biases. They rather aim at explaining remaining deviations in class distributions than at explaining single examples. It is still possible to choose different model classes for mining local and global patterns. Relations between sequential covering (alias separate-and-conquer) strategies and KBS will be pointed out in chapter 5.

Mining local patterns with models of different complexity, with a focus on kernel methods and interpretability is a line of research elaborately discussed by Rüping (2006).

4.8. Summary

In this chapter, the idea of knowledge-based sampling has been presented, a generic technique to make rule selection metrics sensitive to prior knowledge. The interestingness of rules is often relative to a user's expectation or previously found patterns. Still, even learning tasks that explicitly address the problem of finding novel and unexpected patterns lack a support for mining in the presence of prior knowledge. A set of intuitive constraints has been proposed, that formalizes how to construct samples that are similar to the original distribution, but "orthogonal" to expectation, so that subsequently applied rule induction techniques focus on novel patterns. These constraints have been shown to uniquely define a new probability density function, which can be operationalized by re-sampling or by introducing example weights.

The black-box sampling approach does not require to adapt the applied data mining algorithms. It has been shown that it is easy to construct algorithms for this kind of sampling, which, surprisingly, sample from the target distribution exactly, without requiring any estimates. The runtime requirements have been shown to mainly depend on the accuracy of the provided prior knowledge. This reflects the intuition gained from chapter 3, that a precise model requires additional patterns to be supported by a larger number of examples in order to guarantee significance of findings.

Knowledge-based sampling applies to a broad variety of supervised learning tasks. Exemplarily, incorporating prior knowledge has been shown to be beneficial for the learning task of subgroup discovery in this chapter. It can be used to make common subgroup discovery algo-

rithms, like MIDOS, sensitive to prior knowledge, without requiring an adaptation of the algorithm itself. However, evaluating rules globally, as commonly done by subgroup discovery algorithms, is known to result in overlapping patterns. To cover various aspects of a data set it is more appropriate to construct sets of smaller rules, each of which captures a novel pattern of its own. Knowledge-based sampling is a way to shift the focus of subgroup discovery to undiscovered patterns, which allows to construct small sets of rules or other kinds of models with high diversity. A novel sequential subgroup discovery technique based on stratification, iterative reweighting, and an arbitrary embedded rule learner has been presented. It sequentially constructs intuitive distributions based on prior knowledge and any previously discovered patterns. Unlike other sequential subgroup discovery algorithms, it is not biased towards correlated sets of rules, but it tends to report approximately independent patterns. The results are hence useful for predictive and descriptive analysis tasks. The sampling-based nature of the novel algorithm allows to combine this technique with adaptive sampling, which allows to give probabilistic (PAC-like) guarantees when sub-sampling from a large database. Combinations with cost-proportionate rejection sampling and corrections of known sample selection biases in the same framework are also straightforward.

Experiments with six real world data sets were provided, showing that the algorithm outperforms state of the art subgroup discovery based on alternative reweighting strategies. Using a lower number of rules, the predictive performance of knowledge-based sequential subgroup discovery still tends to be higher. Individual rules have a lower coverage, because each rule captures a new aspect of the data. Although the diversities of the resulting rule sets are high, the results are still robust to minor changes in the data.

The next two chapters adapt the knowledge-based sampling technique to the predictive task of boosting classifiers.

5. Boosting as Layered Stratification

5.1. Motivation

When dealing with todays databases and data warehouses, KDD experts face a paradox situation: On the one hand, PAC theory and statistical learning theory teach us that huge amounts of data are required in order to identify complex accurate models with high confidence. On the other hand, impractical runtime requirements prevent the application of popular learning algorithms to very large databases.

In the presence of huge amounts of data approximately meeting the commonly made i.i.d. assumption, sub-sampling provides a genuine way of learning in sub-linear or even constant time. As discussed in section 3.3.1 (p. 48), the expected accuracy typically increases monotonically with a growing sample size, but the *gain* in accuracy decreases rapidly with each additional example. Hence, any learning algorithm scaling super-linearly may utilize sub-sampling in order to decrease runtime complexities significantly at a comparably small sacrifice in accuracy.

This chapter will illustrate how knowledge-based sampling techniques may be used to combine several models, each induced in sub-linear or even constant time with constant main memory requirements. For base learners like top-down induction of decision trees, such ensembles of models can often even be expected to outperform a single model trained from all the data. To allow for this kind of application, the techniques presented in chapter 4 are first adopted to general classification tasks. Please note that optimizing accuracy is a complementary goal to subgroup discovery, the task studied in chapter 4. Subgroup discovery aims at finding interesting patterns that are meaningful to a human analyst. The focal question of knowledge-based sampling is how to preprocess the data, so that subsequent iterations of data mining do not report similar patterns several times, but rather focus on uncorrelated new patterns. This fosters diversity (cf. p. 33) in the resulting sets of models. It is important to change the underlying distribution in an intuitive way, so that subsamples are meaningful and interpretable. The objective of boosting is "simply" to increase predictive performance, while interpretability of models is desirable, but of minor importance. The diversity of rule sets induced by KBS-SD and the close connection between subgroup discovery and classifier induction illustrated by theorem 5 (p. 59) are indicators that knowledge-based sampling may nevertheless be useful in a boosting context. Obviously, the critical property is not scalability in this context, but how well the transformations of underlying density functions match the combination of classifier predictions in terms of predictive performance.

Boosting is one of the most popular learning techniques in practice, but not yet fully understood in terms of its selection metrics and convergence behavior. This chapter analyzes a boosting strategy derived from knowledge-based sampling. The KBS-SD algorithm 2 (p. 89) can be adapted to the task of boosting in a straight-forward manner, simply by removing the initial step of stratification. Basically any crisp base classifier can be employed. In (Scholz, 2005a) this very simple boosting strategy has empirically been shown to be competitive to state of the art boosting algorithms for two-class problems. Further evidence has been provided by Foussette et al. (2004) who report successful experiments with the KDD Cup[1] data of 2004. A slightly

[1] http://kodiak.cs.cornell.edu/kddcup/

improved version of the boosting algorithm sketched above is analyzed in this chapter. It shares the basic ideas of KBS-SD, e.g., of stratifying the target attribute, leading to equal priors of all classes and to equal priors when conditioning on the prediction of a model that is "sampled out". The main difference is, that the algorithm analyzed here introduces stronger skews to the marginal distribution $\mathrm{D} : \mathcal{X} \to \mathbb{R}^+$ than necessary, violating constraint (4.2) (p. 73) proposed in subsection 4.3.1. The skews are introduced to make the base learner focus on subsets with a lower accuracy; the resulting target distributions are less intuitive, but are better suited to achieve high predictive performance. This modified version is analyzed in order to show that it eases the analysis of the algorithm drastically. Besides, it empirically tends to yield better results in terms of predictive performance.

In order to point out similarities and differences of the novel KBS boosting strategy to existing algorithms, this chapter starts with a brief review of the well known ADABOOST algorithm. An improved reweighting strategy is proposed that allows ADABOOST to take more advantage of its base models, which is shown to generally increase the learning rate and hence the generalization performance of boosting. The algorithm still searches a space of linear base classifier combinations by repeatedly applying a crisp boolean classifier. The modified ADABOOST variant is then reformulated in terms of stratification and shown to be identical to KBS-SD up to the mentioned introduction of marginal skews. As another contribution, this chapter provides the first illustration of a boosting algorithm in ROC spaces, more precisely, in their unscaled counterpart which are referred to as coverage spaces. The analysis shows that the novel boosting technique shares some properties of separate-and-conquer rule learning. Furthermore, it allows to derive a new upper bound for the AUC metric for a broader class of boosting algorithms based on the reduction of the total example weight of a given training set.

This chapter is organized as follows: Section 5.2 introduces the specific coverage space notation required for the analytical part of this chapter, and it points out properties of stratification in coverage spaces that go beyond those described in subsection 3.4.1 (p. 56). ADABOOST is discussed and adapted in section 5.3. Proofs on the improved learning rate and an interpretation in coverage spaces illustrate the benefits of the adapted version, but also provide a better understanding why boosting works. In section 5.4 the theoretical results are complemented by an empirical evaluation of generalization performances. Section 5.5 summarizes and concludes.

5.2. Preliminaries

This section provides the background required to analyze boosting techniques in coverage spaces. Subsection 5.2.1 introduces the coverage space notation. Basic properties of stratifying complete example sets in coverage spaces are discussed in subsection 5.2.2.

5.2.1. From ROC to coverage spaces

ROC analysis has become a popular tool for analyzing classifier performances and to study evaluation metrics (see section 2.5). The unscaled counter-part, referred to as coverage spaces, are well suited to visualize the nested subspaces of separate-and-conquer rule learning (Fürnkranz & Flach, 2005). The only difference to ROC spaces is that the axes of coverage spaces show the absolute numbers of positives and negatives, while in ROC spaces both axes are scaled to a range of $[0, 1]$. This chapter confines itself to boolean classification problems, so models are functions mapping an instance space $\mathcal{X}$ to a boolean target label, for mathematical simplicity chosen as $\mathcal{Y} = \{+1, -1\}$. The following definitions used throughout this chapter are chosen to be similar to those used in (Fürnkranz & Flach, 2005) for rule learning in coverage spaces. In

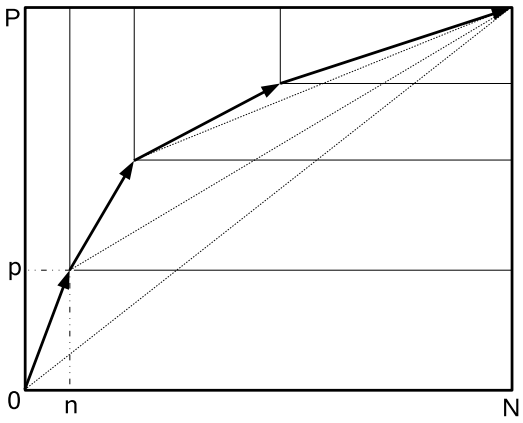


Figure 5.1.: Nested coverage spaces

the subsequent analysis, the symbols are more convenient than TP, FP, TN, and FN (cf. p. 25). The letter always reflects the true label of an example in this notation.

Definition 37 *For a model* $h : \mathcal{X} \to \mathcal{Y}$ *and a given data set* $\mathcal{E}$ *with an absolute number of* P *positive examples and* N *negative examples, the absolute number of true positives is denoted as* p*, the number of false positives as* n*. Analogously, the absolute number of false negatives is denoted as* $\overline{p}$*, the absolute number of true negatives as* $\overline{n}$*.*

Figure 5.1 shows nested coverage spaces from specific to general as obtained when adding a single rule to a decision list in each iteration. The $p + n$ examples for which the rule applies are removed from further consideration and the remaining examples are represented by nested rectangles of shrinking size. Analogously, refining a rule adding one literal at a time shrinks the covered subsets from general to specific.

As discussed in section 2.5, a typical application of *ROC* analysis in machine learning is to visualize soft classifiers that yield continuous confidence scores of how likely examples are considered to be positive. Each crisp classifier obtained by applying a threshold to a soft classifier is represented in a ROC diagram as a point depicting the resulting true positive rate p/P and false positive rate n/N (see section 2.5 or (Fawcett, 2003)). The area under the resulting ROC curve (AUC) equals the probability that a randomly selected positive example is ranked higher (higher confidence) than a randomly selected negative example. Maximizing the AUC is a learning task of its own right and has also been shown to lead to a competitive but more robust selection of models for maximizing predictive accuracy (Rosset, 2004). In this chapter, a quantity closely related to the AUC is analyzed, the area *over* the curve in coverage spaces. An asterisk indicates unscaled quantities in coverage space rather than ROC space. A weighted version of the AOC^* will turn out to be useful at a later point.

Definition 38 *For a given soft classifier and example set* $\mathcal{E} = \mathcal{E}^+ \cup \mathcal{E}^-$ *of positive examples* $\mathcal{E}^+$ *and negative examples* $\mathcal{E}^-$*, the* area over the curve in coverage spaces (AOC^*) *is defined as the number of misranked tuples* $(e^+, e^-) \in \mathcal{E}^+ \times \mathcal{E}^-$ *of a positive example* e^+ *and a negative example* e^-*, that is the number of pairs for which* e^- *is predicted positive with higher confidence than* e^+*. Each pair* (e^+, e^-) *with weights* w^+ *and* w^- *is accounted for by* $w^+ \cdot w^-$ *misranks.*

Ties are considered to be broken randomly, so that half of all equally ranked pairs are considered to be misranked. It is easily seen that $\text{AOC}^* = (1 - \text{AUC}) \cdot P \cdot N$ for the unweighted case, which

is a special case of the weighted case with $w^+ := 1$, $w^- := 1$. An example with a weight of w naturally represents an example *set* of size w.

5.2.2. Properties of stratification

As discussed in subsection 3.4.1, stratification is a preprocessing step commonly applied in machine learning for tasks like incorporating misclassification costs depending only on the true label, or to overcome known sample selection biases in the data. ROC spaces can be interpreted as *stratified coverage spaces*. Before addressing the question of how to boost base classifiers by stratifying subsets of the data, the step of stratification in coverage spaces is discussed for complete data sets.

Definition 31 (p. 56) captures the transformation of a distribution D underlying the data for the case in which classes are required to be equally likely. A simplified version of algorithm 1 (p. 77) proposed in section 4.4 allows to sample from D' exactly, e.g., for i.i.d. data streams or very large databases. Alternatively, stratification can simply be realized by weighting, choosing class-dependent factors with the property that afterwards the total weights of classes meet specified target proportions, e.g. 50% for a boolean target attribute $\mathcal{Y}$.

Class ratios can be skewed arbitrarily at any time, either by weighting or by sub-sampling. It is clear, that this does not change the performance of random classifiers for the original data. As discussed in subsection 2.5.1, the slope of the ROC diagonal represents the precision $Pr(y = +1 \mid h(x) = 1)$ of all random classifiers that do not incorporate any data at all, but just predict $y = +1$ with a fixed probability. Although the term "random classifiers" suggests a bad choice, any finite combination of models making discrete decisions ends up with atomic subsets (cf. Def 35, p. 93) of the instance space $\mathcal{X}$. These subsets contain all examples for which the same predictions or decisions are made by each base classifier. If we would visualize each atomic subset in a separate ROC plot, then the classifier predictions would all lie on the corresponding ROC diagonals, because they do not depend on further properties of examples. This holds for the leaves of a decision tree as well as for a weighted majority vote of discrete base learners (cf. Subsec. 2.6.1).

The subsequently presented learning technique can be considered to continuously improve the performance of a random classifier by stratification. In order to preserve fundamental semantics, it is suggested not to change the AOC^* during the process of stratification, because it reflects the absolute ranking error. This constraint is further justified at a later point. For the original data, the random classifier AOC^* equals half the area of the corresponding coverage space, so $AOC^* = (P \cdot N)/2$. The constraint translates into $P' \cdot N' = P \cdot N$ for the new values P' and N' obtained by skewing P and N, respectively.

The learning algorithms used in this chapter are assumed to implicitly normalize the training set so that the weights describe a distribution. The total weight is hence irrelevant for learning; only the class ratios P/N and P'/N' are of interest for stratification.

Proposition 9 *The reweighting rule for changing the ratio of* P/N *by a factor of* c *while meeting the constraint* $P \cdot N = P' \cdot N'$ *is unique:*

$$w'(x,y) := w(x,y) \cdot \begin{cases} \sqrt{c}, \textit{ for positive } y \\ \frac{1}{\sqrt{c}}, \textit{ for negative } y \end{cases}$$

Proof
It directly follows that after reweighting we have

$$P' = \sqrt{c}P \,,\, N' = \frac{N}{\sqrt{c}} \,,\, \frac{P'}{N'} = c \,,\, P' \cdot N' = P \cdot N.$$

Multiplying positives with a constant of c' requires to divide negatives by the same constant to satisfy the constraint. Only $c' = \sqrt{c}$ is valid, since $P'/N' = c'^2$. □

As required, the weighting does not change the AOC* (Def. 38). Stratification is a specific case of skewing the data, leading to equal class proportions. It has a further important property in the context of boosting. The utility of the following result will become clear in Sec. 5.3.

Proposition 10 *Among all skewing operations preserving* $P \cdot N$*, stratifying the data by choosing* $c = N/P$ *leads to the minimal total example weight.*

Proof
As a constraint, all reweighted sets share the property $P' \cdot N' = P \cdot N$. The total weight to be minimized is $P' + N' = \sqrt{c}P + N/\sqrt{c}$. Setting the derivate of $P' + N'$ as a function of c to zero yields the desired result. □

5.3. Boosting

This section starts with a review of the well known AdaBoost algorithm in 5.3.1. This algorithm is modified and shown to take more advantage of the base classifiers in subsection 5.3.2. Subsection 5.3.3 simplifies the resulting algorithm by reformulating it in terms of stratification, which leads to an algorithm that is very close to the KBS-SD algorithm proposed in the last chapter. Based on a coverage space analysis, an upper bound for the AUC and further properties are shown for the original AdaBoost and for the new algorithm in subsection 5.3.4. Finally, based on the similarity between the new technique and KBS-SD, it is shown in subsection 5.3.5 how to naturally integrate biased classifiers into boosting.

5.3.1. AdaBoost

The ensemble method of boosting allows to induce a set (or ensemble) of classifiers by repeatedly running a "weak" base learner. In each iteration the training set is slightly altered, assigning higher weights to subsets that have been misclassified in previous iterations. Ensembles predict in terms of voting (cf. Subsec. 2.6.1). Boosting is an effective way to increase predictive accuracy and other metrics like the AUC of a weak learner. It allows to reach arbitrarily accurate combined classifiers, as long as each weak hypothesis is slightly better than random guessing in each iteration.

The best known, most studied, and probably the most frequently applied ensemble method is AdaBoost (Freund & Schapire, 1997), see algorithm 3. It fits a sequence of base models $h_t : \mathcal{X} \to \mathcal{Y}$, each to a reweighted version of the training set, or to an analogously constructed subsample, respectively. The term $I[\cdot]$ used in the algorithm refers to the indicator function, α_t denotes the weight assigned to base model t, and ϵ_t is the error rate of this model. To simplify subsequent analysis, the algorithm is formulated without the weight normalization step, assuming that this is implicitly handled by the base learner if required. For a (weighted) error rate of ϵ_t of the base classifier in iteration t and

$$\beta_t(x) := (1 - \epsilon_t)/\epsilon_t = \exp(2\alpha_t) \tag{5.1}$$

Algorithm 3 ADABOOST for $y \in \{+1, -1\}$

Initialize weights $w_1(x_i, y_i) := 1$ for $(x_i, y_i) \in \mathcal{E}$
for $t = 1$ to k **do**
  $h_t \leftarrow$ `base learner`$(\mathcal{E}, w_t)$ // train base classifier, use weights w_t
  Compute $\epsilon_t := \sum_{i=1}^{|\mathcal{E}|} w_t(x_i, y_i) I[h_t(x_i) \neq y_i]$ // error rate of classifier
  Let $\alpha_t := \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
  $w_{t+1}(x_i, y_i) := w_t(x_i, y_i) \cdot \exp(-y_i \alpha_t h_t(x_i))$ // reweight all examples
end for
Output classifier: For example x predict label $\hat{y} := \text{sign}\left(\sum_{i=1}^{k} \alpha_t h_t(x)\right)$

the reweighting strategy computes

$$w_{t+1}(x, y) = w_t(x, y) \cdot \exp(-y \alpha_t h_t(x_i)) = \prod_{t=1}^{k} (\sqrt{\beta_t})^{-y \cdot h_t(x)} \tag{5.2}$$

as the new weight for each example $(x, y) \in \mathcal{E} \subseteq \mathcal{X} \times \mathcal{Y}$, starting with uniform weights. All examples with a final weight w_{t+1} of less than 1 will be classified correctly, because

$$\begin{aligned}
& \sum_{t|h_t(x)=y} \alpha_t > \sum_{t|h_t(x)\neq y} \alpha_t \\
\Leftrightarrow \quad & 1/2 \left(\sum_{t|h_t(x)=y} \ln \frac{1-\epsilon_t}{\epsilon_t} - \sum_{t|h_t(x)\neq y} \ln \frac{1-\epsilon_t}{\epsilon_t} \right) > 0 \\
\Leftrightarrow \quad & \prod_{t=1}^{k} (\sqrt{\beta_t})^{y \cdot h_t(x)} > 1 \\
\Leftrightarrow \quad & w_{t+1}(x, y) < 1.
\end{aligned}$$

In turn, examples with a weight of greater than 1 are misclassified. Clearly, the total weight upper-bounds the training error. For this reason one of the most important properties of ADABOOST is that it reduces the total weight quickly if the base learner provides useful models $h_t : \mathcal{X} \to \mathcal{Y}$, i.e. models better than random guessing.

Intuitively, the reweighting scheme of ADABOOST gives higher weight to the "hard" examples of the training set, and finally predicts based on a weighted majority vote. However, ADABOOST can also be explained in different ways, for example in terms of entropy projection (Kivinen & Warmuth, 1999). A statistical perspective has been fostered by Friedman et al. (2000). They point out ADABOOST's similarity to additive logistic regression; this method is based on the same principle as logistic regression (cf. Subsec. 2.6.3), but it *iteratively* fits functions to the data. Starting with the original input data, it adds another "base function" fit to the residuals of the current model in each iteration. From an optimization point of view, ADABOOST fits into a broader framework proposed by Mason et al. (1999), namely the so-called AnyBoost framework. The optimization strategy of ADABOOST is a gradient descent search in function space, following the objective to minimize the exponential loss function

$$L_{\text{exp}}(F, \mathcal{E}) = \sum_{(x,y)\in\mathcal{E}} \exp(-yF(x)), \quad \text{with } F(x) := \sum_{t=1}^{k} \alpha_t h_t(x)).$$

This function has the example-wise minimizer

$$F(x) = \frac{1}{2} \log \left(\frac{Pr(y = +1 \mid x)}{Pr(y = -1 \mid x)} \right)$$

which is approximated by the prediction rule of ADABOOST.

Finally, the good performance of ADABOOST was also explained by its property of maximizing the margin (Schapire et al., 1998). This property has recently been shown not to hold in the limit, which can be corrected by a minor modification of the algorithm, however (Rätsch & Warmuth, 2005).

Several extensions have been proposed in the last years. Examples include confidence-rated variants (Schapire & Singer, 1999), a version that incorporates prior knowledge (Schapire et al., 2002), and another one that exploits input-dependent properties more rigorously (Jin et al., 2003). Oza and Russell (2001) presented a variant capable of mining from i.i.d. data streams. Quinlan (2001) and Hoche and Wrobel (2002) proposed adaptations for multi-relational data.

The classical ADABOOST algorithm has been presented more than one decade ago, but is still on the agenda of research. For example, ADABOOST's excellent ranking behavior in terms of the area under the ROC curve metric has just recently been explained by its similarity to another well known boosting algorithm called RANKBOOST (Freund et al., 2003), if equal loss is suffered from positive and negative examples (Rudin et al., 2005). In combination with a calibration method proposed by Platt (1999), boosted decision trees have recently been reported by Niculescu-Mizil and Caruana (2005) to yield excellent probability estimates, outperforming e.g, support vector machines.

Despite some short-comings, the original version proposed in (Freund & Schapire, 1997) is probably still the best-known and most popular boosting algorithm in practice. Top-down induction of decision trees, e.g., the C4.5 algorithm by Quinlan (1993), is a well suited base learning technique. In its most simple form, nowadays referred to as DECISIONSTUMPS, they induce trees consisting of a single node only (Holte, 1993). Large empirical studies with decision tree base learners have consistently shown that for most data sets boosting outperforms single trees and bagging (cf. subsection 2.6.1) unless the noise rate is very high (Quinlan, 1996; Bauer & Kohavi, 1999; Dietterich, 2000), and that boosting DECISIONSTUMPS is often competitive to boosting larger trees (Freund & Schapire, 1996). As an important property for practical applications, ADABOOST has been reported to hardly ever overfit to the training data.

5.3.2. ADA^{2}BOOST

One disadvantage of ADABOOST is that it does not take full advantage of its models. For illustration we consider a classification rule covering significantly less than half of the examples (respecting weights), but having a low error rate for this subset. Such a model is generally useful for ensemble learning. However, it is not necessarily useful for ADABOOST, because the error rate of the large uncovered part might be significantly higher, resulting in a value of $\alpha_t \approx 0$.

Such asymmetric cases can be handled by using separate estimates of the error rate for the *covered* part, defined as $C_t := \{(x, y) \in \mathcal{E} \mid h_t(x) = +1\}$, and the *uncovered* part $\overline{C}_t := \{(x, y) \in \mathcal{E} \mid h_t(x) = -1\}$, respectively. Both local error rates, denoted as

$$\epsilon^+ := n/(p + n) \text{ for } C_t, \text{ and } \epsilon^- := \overline{p}/(\overline{p} + \overline{n}) \text{ for } \overline{C}_t$$

can easily be computed from the contingency matrix and will usually differ. Please note, that for $\overline{C}_t$ the *negative* examples are the correctly classified ones. We will replace the static values of β_t by functions $\beta_t(h_t(x))$ that depend only on the prediction of their corresponding base model

$h_t(x) \in \{+1, -1\}$. This leads to two separate factors, the odds ratio for C_t, and the reciprocal odds ratio for $\overline{C}_t$:

$$\beta(+1) := \frac{1-\epsilon^+}{\epsilon^+} = \frac{p}{n} \quad , \qquad \beta(-1) := \frac{1-\epsilon^-}{\epsilon^-} = \frac{\overline{n}}{\overline{p}} \tag{5.3}$$

With $\alpha(h(x)) := (\ln \beta(h(x))/2$ the weight update of ADABOOST changes to

$$w_{t+1}(x_i, y_i) := w_t(x_i, y_i) \cdot \exp\left[-y_i \cdot \alpha_t(h_t(x_i)) \cdot h_t(x_i)\right].$$

The rule for predicting a label $\hat{y} \in \{+1, -1\}$ is changed accordingly:

$$\hat{y} := \text{sign}\left(\sum_{t=1}^{k} \alpha_t(h_t(x))h_t(x)\right) \tag{5.4}$$

This adapted version of AdaBoost is referred to as ADA^{2}BOOST in this chapter. It does not require regression-capabilities of its base learners, as e.g. LogitBoost (Friedman et al., 2000) does when using working responses and weights at the same time. ADA^{2}BOOST is only analyzed in combination with plain boolean base classifiers.

ADA^{2}BOOST is similar to the confidence-rated REAL ADABOOST (Schapire & Singer, 1999), which allows for continuous predictions $h_t : \mathcal{X} \to \mathbb{R}$. REAL ADABOOST reweights examples using the same rule

$$w_{t+1}(x_i, y_i) := w_t(x_i, y_i) \cdot \exp(-y_i \alpha_t h_t(x_i))$$

as used by ADABOOST. The more general setting of continuous functions h_t requires to optimize α_t "manually" based on standard optimization techniques, however. Again, the objective is to minimize the total example weight. The final prediction rule of REAL ADABOOST is identical to the one shown in algorithm 3 for ADABOOST.

If each h_t takes only values from $\{+1, -1\}$ the choice of asymmetric model weights made by ADA^{2}BOOST reduces weights optimally (see Prop. 10). Differences to REAL ADABOOST are that ADA^{2}BOOST (i) uses / supports boolean *crisp* base classifiers, adding confidence-like scores as part of the boosting procedure, and (ii) that it incorporates confidence ratings only in a very moderate form, which constrains the potential to overfit to the training data; when using the same fixed number of base models, ADA^{2}BOOST selects ensemble models from almost the same search space as ADABOOST:

Proposition 11 *If estimated error rates are bounded away from zero, the search spaces of* ADABOOST *and* ADA^{2}BOOST *for boolean classification tasks are identical up to a constant additive offset.*

Proof
A model of the form given by eqn. (5.4) can be transformed into another classifier of the form

$$\hat{y} := \text{sign}\left(\alpha'_0 + \sum_{t=1}^{k} \alpha'_t h_i(x)\right), \tag{5.5}$$

with offset α'_0 and model weights $\alpha'_1, \ldots, \alpha'_k \in \mathbb{R}$ by computing for each model the averages

$$\text{avg}_{0,t} := \frac{\alpha_t(+1) + \alpha_t(-1)}{2},$$

and setting

$$\alpha'_t := \alpha_t(1) - \text{avg}_{0,t} \qquad \alpha'_0 := \sum_{t=1}^{k} \text{avg}_{0,t}.$$

The transformed model (5.5) is obviously equivalent to the original model. □

Aiming to minimize generalization error, it is quite natural to bound the error rates away from 0, e.g., by using Laplace or m-estimates (Cestnik, 1990) for pure subsets. In other boosting variants, similar techniques are referred to as "smoothing".

Although the difference in expressiveness seems marginal, it allows ADA^{2}BOOST to take more advantage of its base models:

Theorem 8 *If $\epsilon = \epsilon^+ = \epsilon^-$ then the reweighting strategies of* ADABOOST *and* ADA^{2}BOOST *are identical. Otherwise* ADA^{2}BOOST *reduces the weights more efficiently.*

Proof
ADABOOST reweights the $p + \overline{n} = 1 - \epsilon$ correctly classified examples multiplying with $\sqrt{\beta}$, and misclassified examples dividing by the same term. Hence, the total weight

$$W_{t+1} = \sum_{i=1}^{|\mathcal{E}|} w_{t+1}(x_i, y_i)$$

in iteration $t + 1$ can be computed as

$$\begin{aligned} W_{t+1} &= \frac{1}{\sqrt{\beta}}\left[(1-\epsilon)W_t\right] + \sqrt{\beta}\left[\epsilon W_t\right] \\ &= 2W_t\sqrt{\epsilon\cdot(1-\epsilon)} \\ &= 2W_t\sqrt{(\overline{p}+n)(p+\overline{n})}. \end{aligned} \tag{5.6}$$

ADA^{2}BOOST reweights the $p + n$ covered examples ($\mathcal{C}$) multiplying their weights with

$$\sqrt{\beta(+1)}^{(\pm 1)} = \left(\sqrt{\frac{\epsilon^+}{1-\epsilon^+}}\right)^{\pm 1} = \left(\sqrt{\frac{n/(p+n)}{1-n/(p+n)}}\right)^{\pm 1} = \left(\sqrt{\frac{n}{p}}\right)^{\pm 1}.$$

Since weights of positive examples are divided by and weights of negative examples are multiplied with $\sqrt{p/n}$, the weight of $\mathcal{C}$ is reduced from $W_t \cdot (p + n)$ to

$$W_t\left(p/\sqrt{p/n} + n\sqrt{p/n}\right) = 2W_t\sqrt{p\cdot n}.$$

The weight of $\overline{\mathcal{C}}$ changes analogously, applying the factor

$$\sqrt{\beta(-1)}^{(\pm 1)} = \sqrt{\overline{n}/\overline{p}}^{(\pm 1)}$$

instead, so the new total weight for ADA^{2}BOOST is

$$W_{t+1} = 2W_t \cdot \left(\sqrt{p\cdot n} + \sqrt{\overline{p}\cdot\overline{n}}\right).$$

We hence need to show

$$\sqrt{(p+\overline{n})(\overline{p}+n)} \geq \sqrt{p\cdot n}+\sqrt{\overline{p}\cdot\overline{n}},$$

where the term on the left reflects the weight decrease achieved by AdaBoost, and the term on the right the decrease achieved by Ada^{2}Boost. The inequality follows from

$$\begin{aligned}
&\sqrt{(p+\overline{n})(\overline{p}+n)} \geq \sqrt{p\cdot n}+\sqrt{\overline{p}\cdot\overline{n}}\\
\Leftrightarrow\ &(p+\overline{n})(\overline{p}+n) \geq pn+\overline{pn}+2\sqrt{pn\overline{pn}}\\
\Leftrightarrow\ &p\overline{p}+n\overline{n} \geq 2\sqrt{pn\overline{pn}}\\
\Leftrightarrow\ &(p\overline{p})^2+2pn\overline{pn}+(n\overline{n})^2 \geq 4pn\overline{pn}\\
\Leftrightarrow\ &(p\overline{p}-n\overline{n})^2 \geq 0.
\end{aligned}$$

Both strategies yield the same result if and only if $p\overline{p}=n\overline{n}$. This is equivalent to

$$\frac{p}{n}=\frac{\overline{n}}{\overline{p}} \Leftrightarrow \ \frac{p/(p+n)}{n/(p+n)}=\frac{\overline{n}/(\overline{n}+\overline{p})}{\overline{p}/(\overline{n}+\overline{p})} \Leftrightarrow \ \frac{1-\epsilon^+}{\epsilon^+}=\frac{1-\epsilon^-}{\epsilon^-} \Leftrightarrow \epsilon^+=\epsilon^-$$

If n or $\overline{p}$ are 0, then either *both* error rates need to be 0, or one of the local error rates is undefined, because the subset contains no examples. The fact that ϵ is a weighted average of ϵ^+ and ϵ^- completes the proof. □

As a consequence, for sequences of k base classifiers with fixed weighted error rates ϵ_i, $i \in \{1,\ldots,k\}$, the worst case for Ada^{2}Boost is to be identical to AdaBoost. If $\gamma_i := 1/2-\epsilon$ denotes the advantage over random guessing, recursively applying eqn. (5.6) yields

$$W_{k+1}=|\mathcal{E}|\cdot\prod_{i=1}^{k}\left(2\sqrt{\epsilon_i\cdot(1-\epsilon_i)}\right)=|\mathcal{E}|\cdot\prod_{i=1}^{k}\sqrt{1-4\gamma_i}$$

as the total final example weight in this case (Freund & Schapire, 1997). As mentioned before, this quantity upper-bounds the absolute number of misclassified training examples for both ensemble methods.

Usually AdaBoost tends to generate a diverse or approximately independent set of base classifiers. The reason is that when sub-sampling an example e weight-proportionally after reweighting in iteration i, then the boolean random variable "h_i classifies e correctly" is independent of the class label. Any remaining correlations of subsequent base classifiers with h_i are implicitly compensated by reducing the corresponding model weight appropriately. The following proposition helps to derive corresponding properties of Ada^{2}Boost.

Proposition 12 *After* Ada^{2}Boost *reweights for the first time, we have* $P'=N'$. *After each iteration* t, *the subsets* $\mathcal{C}_t$ *and* $\overline{\mathcal{C}}_t$ *corresponding to the model* h_t *are both stratified. The error rates* ϵ_t^+ *and* ϵ_t^- *of* h_t *are exactly* $1/2$ *with respect to* w_{t+1}.

This result can directly be derived from the weighting scheme. From these properties it can be concluded that the same kind of base classifier independence that holds for AdaBoost may also be assumed for Ada^{2}Boost ensembles; label and correct predictions are again independent after each reweighting step. Besides, there is evidence for an even stronger kind of independence: Ada^{2}Boost reweights the example set so that the individual labels and predictions of the model

are independent if interpreted as random variables. Since neither the base learner does have access to the predictions of previous iterations, nor are they considered in later reweighting steps, conditional independence emerges as a natural phenomenon in this setting.

The prediction rule of ADA^{2}BOOST suggests that it performs especially well for conditionally independent base classifiers. Denoting with $\Pr_{D_t}(\cdot)$ probabilities based on weights w_t, it uses a product of β_t terms as defined in eqn. (5.3) to compute odds-ratio estimates $\widehat{\beta}(x)$, exploiting that for the true odds $\beta(x)$ we have

$$\begin{aligned}
\beta(x) &= \frac{\Pr(y=+1 \mid h_1(x),\ldots,h_k(x))}{\Pr(y=-1 \mid h_1(x),\ldots,h_k(x))} \\
&= \frac{\Pr(h_1(x),\ldots,h_k(x) \mid y=+1)}{\Pr(h_1(x),\ldots,h_k(x) \mid y=-1)} \cdot \frac{\Pr(y=+1)}{\Pr(y=-1)} \cdot \frac{\Pr(h_1(x),\ldots,h_k(x))}{\Pr(h_1(x),\ldots,h_k(x))} \\
&\approx \frac{\Pr(y=+1)}{\Pr(y=-1)} \cdot \prod_{t=1}^{k} \frac{\Pr_{D_t}(h_t(x) \mid y=+1)}{\Pr_{D_t}(h_t(x) \mid y=-1)} \\
&= \frac{\Pr(y=+1)}{\Pr(y=-1)} \cdot \prod_{t=1}^{k} \left(\frac{\Pr_{D_t}(y=+1 \mid h_t(x)) \cdot \Pr(h_t(x)) \cdot \Pr(y=-1)}{\Pr_{D_t}(y=-1 \mid h_t(x)) \cdot \Pr(h_t(x)) \cdot \Pr(y=+1)} \right) \\
&= \frac{\Pr(y=+1)}{\Pr(y=-1)} \cdot \prod_{t=1}^{k} \frac{\mathrm{LIFT}_{D_t}(h_t(x) \to y=+1)}{\mathrm{LIFT}_{D_t}(h_t(x) \to y=-1)} \\
&= \prod_{t=1}^{k} \frac{\Pr_{D_t}(y=+1 \mid h_t(x))}{\Pr_{D_t}(y=-1 \mid h_t(x))}. \qquad (5.7)
\end{aligned}$$

This is again a NAÏVEBAYES-like combination of classifiers. The last equality holds because of the stratification property described by Prop. 12, holding for all but the first model. Hence, the first lift ratio (for $t = 1$) contains the reciprocal class skew, while for all other models the lift ratio is identical to the ratios of conditional class probabilities. This simple interpretation requires base models to have discrete prediction domains.

If the independence assumptions are met, then the weight subscripts in eqn. (5.7) may be ignored, because the odds ratios are equivalent for all distributions. Without these subscripts, eqn. (5.7) is an exact reformulation of NAÏVEBAYES on top of the base model predictions $h_t(x)$. This yields the Bayes' optimal decision rule in this setting, so it clearly succeeds no worse in selecting the most probable class than ADABOOST. Moreover, exploiting that for the true odds ratios $\beta(x)$ we have

$$\Pr(y=+1 \mid x) = \frac{\beta(x)}{1+\beta(x)},$$

each estimated value of $\widehat{\beta}(x)$ can easily be transformed into a probability estimate.

Even in cases where conditional independence is lacking, ADA^{2}BOOST fits an additive model to the log-odds. This suggests that it may yield good estimates of conditional class distributions in any case.

5.3.3. A reformulation in terms of stratification

The reweighting strategy of ADA^{2}BOOST is very similar to the stratification proposed in section 5.2.2. In each iteration, both the covered ($\mathcal{C}$) and the uncovered subset ($\overline{\mathcal{C}}$) are stratified separately in the sense of Prop. 9. This motivates a reformulation of the algorithm in terms of

Algorithm 4 ADA^{2}BOOST for $y \in \{+1, -1\}$

Let $w_1(x, y) := 1$ for all $(x, y) \in \mathcal{E}$ // Init with uniform weights
// Train k base classifiers:
for $t = 1$ to k **do**
  $h_t \leftarrow$ `base learner`$(\mathcal{E}, w_t)$
  $\beta'_t(+1) := p_t/n_t$, $\beta'_t(-1) := \overline{p}_t/\overline{n}_t$ // Compute odds ratios
  For all $(x, y) \in \mathcal{E}$ let
    $w_{t+1}(x, y) := \frac{w_t(x,y)}{\sqrt{\beta'_t(h_t(x))}^y}$ // Stratification
end for
Output soft classifier:
  $\hat{\beta}(x) := \prod_{t=1}^{k} \beta'_t(h_t(x))$ // Estimated odds ratio
  $\widehat{Pr}(y = +1 \mid x) = \hat{\beta}(x)/(1 + \hat{\beta}(x))$ // Estimated probability of $y = +1$

stratification. Algorithm 4 is equivalent to the previously discussed ADA^{2}BOOST, but the formalization is more intuitive. As a first difference, the use of exponential or logarithmic functions (α_t values) seems to be unnecessarily complicated and is avoided; ADA^{2}BOOST only requires the β_t values. Second, the algorithm uses β'_t, terms that *always* refer to the odds ratios, while the previously used β_t referred to the reciprocal odds ratios whenever a base classifier predicted the negative class.

The resulting algorithm boosts boolean base classifiers in a very simple fashion: In each iteration t another stratified (for $t > 1$, cf. Prop. 12) example set is presented to the base learner. The learner returns a base classifier $h_t : \mathcal{X} \to \{+1, -1\}$ that partitions the example set into "unstratified" subsets $\mathcal{C}$ and $\overline{\mathcal{C}}$; this automatically happens when maximizing accuracy, because we are implicitly maximizing WRACC (cf. theorem 5, p. 59). The terms p_t, $\overline{p}_t$, n_t, and $\overline{n}_t$ denote the true positives to false negatives of model h_t based on example weights w_t. For both partitions, $\mathcal{C}$ and $\overline{\mathcal{C}}$, the odds ratios are computed and stored for later predictions, before stratifying the subsets separately, respecting the constraint stated in Prop. 9. When predicting a label, the local odds are combined applying eqn. (5.7), which allows to derive conditional probability estimates.

After the reformulation, ADA^{2}BOOST can easily be seen to be very similar to KBS-SD (algorithm 2, p. 89). Both algorithms work by stratifying the subsets $\mathcal{C}$ and $\overline{\mathcal{C}}$ separately. ADA^{2}BOOST does not stratify in advance, but we also have equally probable classes after the first iteration. This implies that the base learner implicitly maximizes the WRACC metric, like the base learner of KBS-SD. Stratification could as well be performed before the first iteration.

In subsection 4.3.1 the reweighting scheme of KBS-SD was motivated in terms of constraints (p. 73). The reweighting rule of ADA^{2}BOOST also changes a probability density function underlying the data. This rule satisfies almost all constraints.

Proposition 13 *After an initial step of stratification the reweighting rule of* ADA^{2}BOOST *satisfies constraints* (4.1)*,* (4.3)*, and* (4.4) *to* (4.7)*.*

Proof
The first constraint (4.1), independence of true label y and predicted label $h_t(x)$ for $(x, y) \sim D_{t+1}$, is met because of Prop. 12 that states

$$(\forall y^*, \hat{y} \in \mathcal{Y}) : Pr_{D_{t+t}}(y = y^* \mid h_t(x) = \hat{y}) = 1/2.$$

Prop. 12 also directly implies constraint (4.3), not to change the class priors; all subsequent pdfs are stratified. Algorithm 4 does not stratify in advance, but the original class priors can still be

re-established (as for KBS-SD) by inverting the initial step of stratification, which is part of the first reweighting step.

Constraints (4.4) to (4.7) require altered density functions D' to be proportional to the original pdf D within each subset sharing a true and a predicted label. This property holds, because the densities of examples within each subset are defined by rescaling proportionally. □

There are further connections between ADA^{2}BOOST and KBS-SD. First, the square roots of the odds used by ADA^{2}BOOST cancel out when applying the model. As the class priors of both classes are identical, the odds ratios $\beta'_t(h_t(x))$ are identical to the LIFT ratios used by KBS-SD. Hence, both algorithms use the same prediction rule. Second, the property of atomic subsets (Def. 35) to reflect the residuals of ensemble predictions – stated by theorem 7 (p. 93) for KBS-SD – can be shown to also hold for ADA^{2}BOOST. For simplicity, this result is formulated with respect to the training error.

Proposition 14 *Let* $H = \{h_i \mid 1 \leq i \leq k\}$ *be a set of models output by the* ADA^{2}BOOST *algorithm. Let further* S *refer to a subset of examples, for which all rules in* H *make the same prediction, and let* $\hat{\beta}(S)$ *denote the corresponding odds ratio estimate of* ADA^{2}BOOST. *If the unweighted (true) odds ratio in* S *with respect to uniform weights is* $s \cdot \hat{\beta}(S)$ *for any* $s \in \mathbb{R}^+$, *then the corresponding odds ratio with respect to the final weights is* s.

Proof
The prediction $\hat{\beta}(S)$ is a product of individual base classifier odds ratios $\beta'_t(\cdot)$. Weights of positives are divided by $\sqrt{\beta'_t(\cdot)}$ in iteration t, while weights of negatives are multiplied with the same term. This implies that in iteration t the ratio of total positive to total negative example weight in S changes by a factor of $(\beta'_t(\cdot))^{-1}$. The final odds ratio in S is hence $\hat{\beta}(S)$ times lower after reweighting. □

The only constraint that is violated by the reweighting rule of ADA^{2}BOOST is (4.2). It requires to preserve the coverages of $\mathcal{C}$ and $\overline{\mathcal{C}}$. Please recall from theorem 6 (p. 74), that if constraint (4.2) would also hold, then the reweighting rule would necessarily be identical to the one used by KBS-SD. KBS-SD preserves the coverages of both sets, while ADA^{2}BOOST introduces a skew due to the square root transformation. This will be illustrated in the next paragraphs.

5.3.4. Analysis in coverage spaces

Figure 5.2 shows a step of stratification for two partitions, e.g. for a classification rule in coverage space. The two boxes represent the performances of two dual rules

$$(h_t(x) = +1) \rightarrow (y = +1) \quad \text{and}$$
$$(h_t(x) = -1) \rightarrow (y = -1),$$

where the slope of the diagonal is $\beta'_t(+1) = p/n$ for the former, and $\beta'_t(-1) = \overline{p}/\overline{n}$ for the latter. Stratification turns each of these boxes into a square (hence Ada^2), while preserving the enclosed areas. This step is illustrated on the left.

Stratification is a transformation of the underlying distribution (cf. Def. 31, p. 56) that can be inverted precisely when making predictions. In sub-sampling scenarios the stratification step

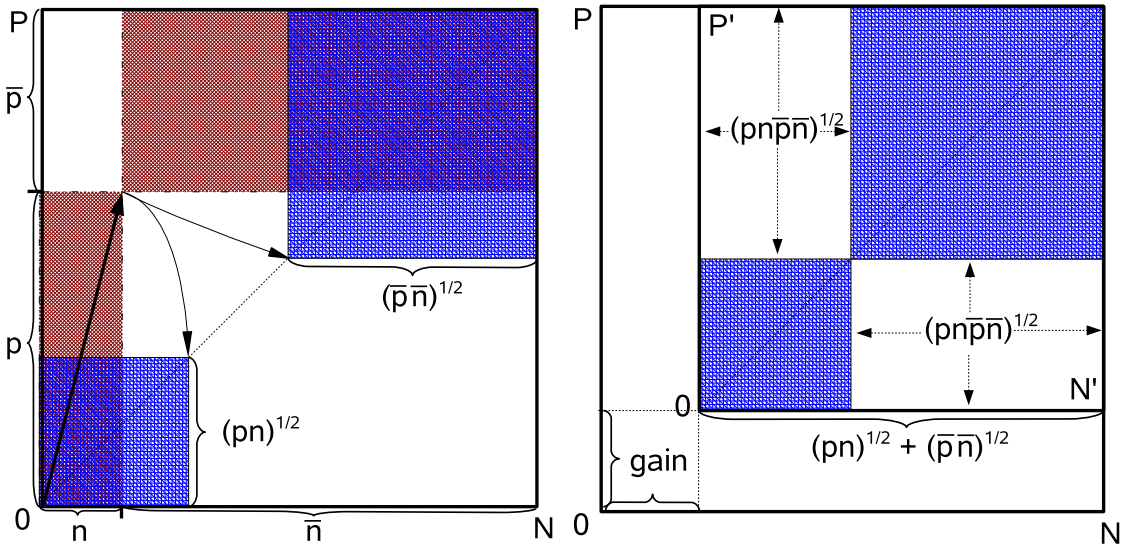


Figure 5.2.: ADA^{2}BOOST transforms the boxes representing $\mathcal{C}$ and $\overline{\mathcal{C}}$ into squares (left Fig.) by reweighting. Moving these squares to the upper right (right Fig.) yields a coverage subspace with the new origin 0 inside of the outer coverage space, and with the new axes P' and N'.

for both subsets can again be operationalized exactly by Monte Carlo techniques. A substantial difference to the reweighting strategy discussed in the last chapter is the dependency of the coverages of $\mathcal{C}$ and $\overline{\mathcal{C}}$ on the corresponding precisions of the model. Figure 5.3 illustrates the difference between the two reweighting schemes. KBS-SD also stratifies the data, but it preserves the coverage. For stratified data the isometrics of coverages are diagonal lines, connecting the points $(0,1)$ and $(1,0)$. Hence, without explictly addressing the changing coverages the simple but precise knowledge-based rejection sampling algorithm (p. 77) does not realize the transformations of the underlying pdf defined by ADA^{2}BOOST. Referring to the *true* values of p and n with respect to an underlying pdf, ADA^{2}BOOST changes the coverage of e.g. $\mathcal{C}$ from $p+n$ under D_t to $2\sqrt{pn}$ under D_{t+1}. That is, the coverage changes by a factor of $\sqrt{pn}/(0.5\cdot(p+n))$, the quotient of geometric and arithmetic mean. Sampling exactly from D_{t+1} seems to be considerably more complicated, because the true values of p and n are usually unknown[2]. However, for large data sets the target coverages can well be approximated based on samples. Alternatively, the aggregated ensemble estimates may be used to compute weights for rejection sampling, with the same disadvantage as discussed in the preceeding chapter: Refining the estimates (and weights) of any model h_t during the procedure will change subsequent pdfs and hence the performances of all models $h_{t'}$, $t' > t$. Refining estimates is not common in the context of boosting, however.

It is interesting to note that the role of the base learner can as well be stated in terms of stratification. Its objective is to divide $\mathcal{E}$ into "unstratified" subsets, which are continuously stratified (conquered) by the meta-algorithm, as long as the base learner succeeds. In separate-and-conquer *rule learning* examples that are covered are removed from subsequent learning iterations. Similarly, boosting can be considered to probabilistically discard examples. The main difference is, that it only discards a subset of the correctly predicted examples by turning boxes

[2]Surprisingly, without the square root term sampling precisely would still be easy. To sample from this pdf D' it is sufficient to sample example *pairs* from D, and to accept only those pairs which have one positive and one negative example. It seems hard to realize the square root transformation, however.

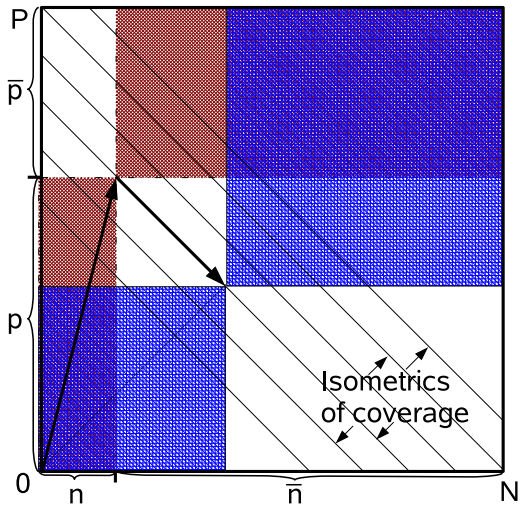


Figure 5.3.: The reweighting step of KBS-SD in coverage spaces.

in coverage spaces into squares.

Moving both squares in Fig. 5.2 to the upper right we reach at a visualization of boosting in *nested coverage spaces*. The weight lost by this transformation shows in the figure on the right as the part of the axes of the original coverage space below and left to the embedded coverage space. Since the complete data set is stratified itself, the nested coverage spaces share the proportions of ROC spaces.

Exactly those examples having a weight of greater than 1 are misclassified. Prop. 10 states, that ADA^{2}BOOST reduces the total weight as much as possible, while respecting the constraint to preserve the area of each subset in coverage space. Additionally, Fig. 5.2 illustrates that for a single step of stratification the AOC* of the classifier before stratification is upper-bounded by half the area of the nested coverage space: The AOC* is

$$\overline{p}n + (pn + \overline{pn})/2,$$

while the nested coverage space has a size of

$$2\sqrt{\overline{p}n \cdot p\overline{n}} + pn + \overline{pn}.$$

The following theorem generalizes this observation. It states, that the bound holds in general, i.e. for complete ensembles, and that it applies to both ADABOOST variants.

Theorem 9 *The absolute ranking error (*AOC**) of* ADABOOST *and* ADA^{2}BOOST *ensemble models for the original (unweighted) data is upper-bounded by the* AOC* *of the model after reweighting, that is the* AOC* *of the ensemble for the inner nested coverage space.*

Proof

The crucial observation is, that final confidences and weights are closely related. A pair of examples (e^+, e^-) with weights w^+ and w^- will be misranked, iff the estimated confidence of being positive is higher for e^- than for e^+. The confidences are monotone in the estimated odds ratios $\hat{\beta}(e^+)$ and $\hat{\beta}(e^-)$. Applying the reweighting scheme of ADA^{2}BOOST recursively for an ensemble of size k we find that

$$w_{k+1}(x, y) = \prod_{t=1}^{k} \sqrt{\beta'_t(h_t(x))}^{(-y)}.$$

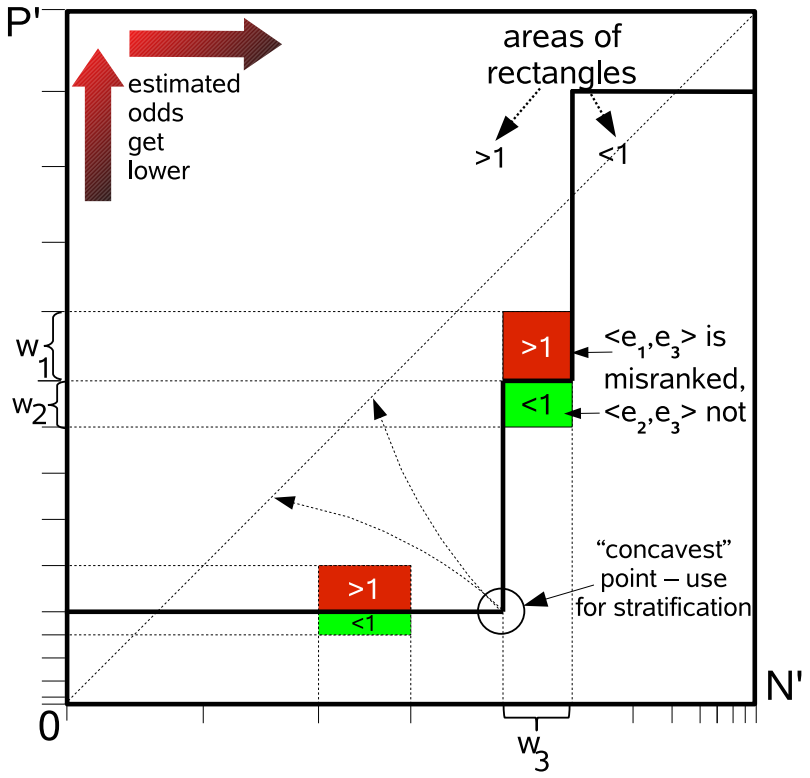


Figure 5.4.: Illustration for proof of theorem 9. Each example is visualized as a part of the corresponding axis. The length is proportional to the example weight. Weights of positives ascend along the P' axis, weights of negatives descend along the N' axis. Whether a pair of a positive and a negative example is misranked or not depends only on the area of the corresponding rectangle in the (inner) coverage space above. The border where areas become larger than 1 is depicted as a thick line. Concavities can be exploited for stratification.

At the same time we have

$$\hat{\beta}(x) = \prod_{t=1}^{k} \beta'_t(h_t(x)).$$

This implies

$$w^+ = \sqrt{\frac{1}{\hat{\beta}(e^+)}} \text{ and } w^- = \sqrt{\hat{\beta}(e^-)}. \tag{5.8}$$

For misranked example pairs (e^+, e^-) we have

$$\hat{\beta}(e^-) > \hat{\beta}(e^+) \Leftrightarrow (w^-)^2 > 1/(w^+)^2 \Leftrightarrow w^+ \cdot w^- > 1. \tag{5.9}$$

We can derive the same connection between $\hat{\beta}$ and example weights when substituting the AdaBoost reweighting rule.

It implies that each misranked pair (e^+, e^-) "occupies" a rectangle with an area of at least 1 of the inner nested coverage space. All rectangles representing different pairs (e^+, e^-) are disjoint.

Hence, if the nested coverage space has a size of $P' \cdot N'$, then this quantity upper-bounds the AOC* of the ensemble for the original data.

When ordering examples by confidence, as for soft classifier ROC plots, the weights of positives ascend along the P axis, while the weights of negatives descend along the N axis. Hence, the areas of rectangles representing example pairs grow monotonically when approaching the upper left corner $(0, P')$. The border where areas become larger than 1 is marked as a thick line in Fig. 5.4. Example pairs share their estimated odds along the border. Apart from the scale, Fig. 5.4 is a ROC plot of the ensemble for the *reweighted* example set; a threshold is associated to each point of the border. By construction we expect the ensemble to perform as good as random guessing after reweighting, leading to an AOC* of $(P' \cdot N')/2$. In this case only *half* of the total nested coverage space represents misclassified pairs (areas ≥ 1), which also halves the AOC* upper-bound for the original (unweighted) data. The same argument applies for any other AOC* score. □

The constraint formulated in Prop. 9 is required for the proof when deriving eqn. (5.9) for ADA^2-BOOST. More precisely, the important property is that the weights of positives are multiplied and the weights of negatives are divided by the same term for each subset. As a consequence, the upper-bound of theorem 9 does not only hold for ADABOOST, but also for any other reweighting scheme following Prop. 9, in particular for REAL ADABOOST-like ensemble classifiers. The derived bounds are tighter than those provided in (Schapire & Singer, 1999), based only on the worst case AOC* of $P' \cdot N'$. To the best of the author's knowledge theorem 9 provides the first AUC (AOC*) bound based on ranking performances after reweighting.

According to theorem 8, ADA^{2}BOOST reduces the total example weight and hence the size of the coverage space optimally within this setting, apart from the fact that it is still a greedy algorithm. The coverage spaces of ADABOOST are not necessarily stratified. However, by finally adding a single constant base classifier to each ensemble, which just stratifies the data, it is easily seen that the total weight also determines the area of coverage spaces for ADABOOST. This stresses the role of quick weight reductions even further.

Exploiting the fact that coverage spaces are stratified for ADA^{2}BOOST we can directly derive a corresponding upper-bound for the AUC:

Corollary 2 *For an example set $\mathcal{E} = \mathcal{E}^+ \cup \mathcal{E}^-$ a reduction of the initial weight of $|\mathcal{E}|$ to W results in an* AUC $\geq 1 - W^2/(8 \cdot |\mathcal{E}^+| \cdot |\mathcal{E}^-|)$ *if the* AUC *of the ensemble is at least $1/2$ (random guessing) for the reweighted example set.*

Serious concavities as shown in Fig. 5.4, i.e. AUCs $\ll 1/2$ after reweighting, seem not to occur in practice. However, they would not pose a problem anyway, but would rather be well suited to further reduce the total weight, e.g. by stratification, partitioning at a point most distant to the main diagonal. This step is illustrated in the figure.

Finally it should be mentioned, that the quicker weight reduction is also an advantage in the minimum description length (MDL) framework of Rissanen (1978). According to information theory (Shannon & Weaver, 1969), an optimal coding strategy requires $-\log_2(\widehat{\Pr}(y \mid x))$ bits to encode an example $(x, y) \in \mathcal{X} \times \mathcal{Y}$ if the predicted conditional class probability of the (true) label y is $\widehat{\Pr}(y \mid x)$ according to a given model. For $y = +1$ and example weight w the predicted conditional probability used by ADA^{2}BOOST can be rewritten (see eqn. (5.8)) as

$$\widehat{\Pr}(y = +1 \mid x) = \frac{\hat{\beta}(x)}{1 + \hat{\beta}(x)} = \frac{1/w^2}{1 + 1/w^2} = \frac{1}{w^2 + 1},$$

resulting in an encoding length of $\log_2(w^2+1)$. For $y=-1$ the reciprocal odds ratios are used instead of $\hat{\beta}$, which leads to the same term with respect to the example weights. Substituting the formulas used by original ADABOOST leads to the same encoding length with respect to the weights.

Corollary 3 *For a given example set $\mathcal{E}$ and an* ADABOOST *or* ADA^{2}BOOST *ensemble model producing example weights $w_1,\ldots,w_{|\mathcal{E}|}$ after training, the label can be encoded using*

$$\sum_{i=1}^{|\mathcal{E}|}\log_2(w_i^2+1)$$

bits according to information theory.

The main goal in the MDL framework is a reduction of the overall encoding length. This can first of all be justified by "Occam's razor" which suggests to prefer simpler models over complex ones. In more technical terms, if we assume that the encoding length of each model reflects its prior probability, a minimization of the accumulated encoding length of the selected model and training set leads to a maximum a posteriori model. See e.g. (Mitchell, 1990) for a simple derivation. The MDL principle has successfully been applied in various contexts, e.g. for pruning decision trees (Mehta et al., 1995).

The weight reduction of the ADABOOST variants is not directly backed by e.g., a proportional reduction of the encoding length for the training set, because of the constant offsets of 1 that are added to the (squared) weight terms. However, the corollary suggests a quick reduction of the encoding length as the total weight decreases. Please note that – due to the logarithmic transformation – the exponent alone would have no effect in an optimization framework. Corollary 3 illustrates that ADA^{2}BOOST tends to maximize the likelihood of ensembles, similar to logistic regression, but using a cheap sequential approximation strategy. The encoding length of the applied models needs to be accounted for separately, so if the models used by ADABOOST and ADA^{2}BOOST are of a comparable complexity (length), as e.g. given when boosting DECISION-STUMPS, then according to Prop. 11 the quicker weight reduction of ADA^{2}BOOST will generally lead to a more efficient overall reduction of the encoding length for any fixed number of models. Additional models used by ADABOOST to compensate the difference (if possible) increase the overall encoding length.

5.3.5. Learning under skewed class distributions

ADA^{2}BOOST is a boosting technique based on layered stratification. An interesting related and somehow diametric idea recently proposed by Khoussainov et al. (2005) is to *introduce* class skews artificially, in order to get highly confident predictions for a specific class. ADA^{2}BOOST allows to naturally integrate so-called *biased classifiers* trained on data sets that were skewed for an arbitrary constant $c>0$ in the sense of Prop. 9. The crucial observation is that skewing the data leads to exactly the kind of unbalanced decision rules that were used to motivate ADA^2-BOOST in subsection 5.3.2. As was shown, ADA^{2}BOOST handles asymmetric precisions well. It stratifies the sets $\mathcal{C}$ and $\overline{\mathcal{C}}$ separately. If a complete example set is skewed artificially, then the global skew can be used to adapt the odds ratios $\beta'_t(\cdot)$ of a model h_t to fit the skewed data. It is sufficient to redefine

$$\begin{aligned}\beta'_t(+1) &:= \frac{p}{n}\cdot\frac{N}{P}=\frac{p}{n}\cdot\frac{\Pr(y=-1)}{\Pr(y=+1)} \quad \text{and}\\ \beta'_t(-1) &:= \frac{\overline{p}}{\overline{n}}\cdot\frac{N}{P}=\frac{p}{n}\cdot\frac{\Pr(y=-1)}{\Pr(y=+1)},\end{aligned}$$

Algorithm 5 Skewed ADA^{2}BOOST for $y \in \{+1, -1\}$

Compute P_1/N_1 from training set $\mathcal{E}$
for $t = 1$ to k **do**
  Randomly select $\pi_+ \sim N(0.5, 0.25), 0 < \pi_+ < 1$
  Skew $\mathcal{E}$ so that // done via multiplying weights with constant $c^{\pm 1}$
    $P_t := Pr(y = +1) = \pi_+$
    $N_t := Pr(y = -1) = 1 - \pi_+$
  $h_t := \texttt{base learner}(\mathcal{E})$ // learning based on reweighted example set
  Compute
    $\beta'_t(+1) := (p_t/n_t) \cdot (N_t/P_t)$
    $\beta'_t(-1) := (\overline{p}_t/\overline{n}_t) \cdot (N_t/P_t)$
  Stratify $\mathcal{C}$ and $\overline{\mathcal{C}}$ // reweighting as for ADA^{2}BOOST
end for
Output: $\hat{\beta}(x) = (P_1/N_1) \cdot \prod_{t=1}^{k} \beta'_t(h_t(x))$
  $\widehat{Pr}(y = +1 \mid x) := \hat{\beta}(x)/(1 + \hat{\beta}(x))$

where the estimates for p, n, $\overline{p}$, $\overline{n}$, and $Pr(y = \pm 1)$ are computed from the *skewed* data sets. It is not necessary to keep track of introduced skews, only the initial value of P/N is required at prediction time. The odds ratios above reflect the values for stratified data. When the data has not been skewed "manually" the redefinition is identical to the one used by algorithm 4, so one can always apply the more general redefinition. Please note that the $\beta'_t(\cdot)$ values above are simply the LIFT ratios (LR) introduced in chapter 4 (p. 90) for KBS. The corresponding prediction rule is invariant to artificial skews.

Algorithm 5 illustrates how to integrate biased classifiers into the overall boosting framework, combining the layered stratification approach of ADA^{2}BOOST with the idea of Khoussainov et al. (2005). A classifier trained for a specific ratio of P_t/N_t can only improve over the default classifier by identifying a classifier h_t with a precision satisfying $Pr(y = +1 \mid h_t(x) = +1) > P_t/(P_t + N_t)$, so each skew can be interpreted as a confidence threshold for a specific class. The depicted variant simplifies matters by choosing the skews at random. It is meant as a proof of concept, that allows for preliminary empirical evaluation. Alternatively, the training set could be split into 3 or more partitions by running the base learner at different levels of skew. Using more than 2 partitions is a valid extension of ADA^{2}BOOST, following the same rationale, to stratify each individual subset separately.

5.4. Evaluation

In the previous sections ADA^{2}BOOST has been presented. It relies on classifiers to divide the example set into partitions having a ratio of p/n that differs from P/N. These partitions are iteratively stratified. Skewed ADA^{2}BOOST changes P/N randomly, which does not interfere with making predictions but allows to introduce confidence thresholds. Both variants reduce the total example weight more quickly than ADABOOST does, which leads to quicker improvements of the accuracy and area under the ROC curve metric (AUC). This section empirically evaluates the generalization performance with respect to these two metrics. Moreover, evidence has been provided, that ADA^{2}BOOST may perform well as a soft classifier for estimating conditional class probabilities. Hence, as a third metric the root mean squared error (RMSE, p. 20) is used in the

evaluation. For the experiments 4 benchmark data sets taken from the UCI library[3] (Blake & Merz, 1998), and a 10k sample of the quantum physics data set from KDD Cup 2004 were used.

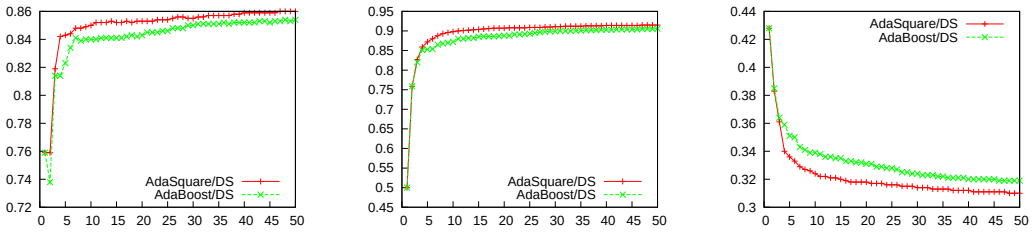
The WEKA library (Witten & Frank, 2000) provides ADABOOST and DECISIONSTUMPS as base learners. ADA^{2}BOOST was implemented in YALE[4] (Mierswa et al., 2006) without any optimizations like Laplace estimates or repairing concavities. DECISIONSTUMPS are popular base classifiers in practice, and they do not apply a greedy search themselves, which eases the evaluation of greedily operating boosting techniques. In the proposed experimental setting, REAL ADABOOST with "reasonable" confidence ratings for each decision stump yields the same predictions as ADA^{2}BOOST. The focus of the evaluation is hence on AUC maximization and its relation to ACC optimization; for error rate minimization please refer to (Schapire & Singer, 1999) or (Freund & Mason, 1999), because technically seen ADA^{2}BOOST is a specific confidence-rated boosting algorithm.

To give a full picture of the performances, full learning curves for different numbers of base models are shown in Fig. 5.5 to 5.9, where the red curve always refers to ADA^{2}BOOST and the green curve to ADABOOST. Each point in the plots is the result of a ten-fold cross-validation. The example sets are ordered by their number of attributes, which intuitively and empirically seems to be a good indicator of possible improvements over ADABOOST. The results illustrate that boosting in fact maximizes all three considered metrics simultaneously, with AUC and RMSE providing finer-grained indicators of progress than ACC. The moderate adaptation of ADABOOST does not only improve the ACC learning rate of the original algorithm, but also leads to similar improvements for the metrics AUC and RMSE.

The difference between ADABOOST and ADA^{2}BOOST for Credit-G (Fig. 5.6) containing very few examples *and* attributes (16) are the smallest (if any), lying within half a standard deviation. The adult data set contains even one attribute less, but 8K examples. This leads to a small, but still significant improvement: For all three metrics and e.g. 10 or 50 stumps it passes a t-test at a confidence level of 2%. For the remaining data sets the advantages are much clearer. E.g., for mushrooms ADA^{2}BOOST produces a perfect ranking with only 9 (Fig. 5.7(b)), and perfect *soft* predictions with 19 stumps (Fig. 5.7(c)). ADABOOST requires 24 stumps to rank perfectly and 100 stumps to reach an RMS of about 2% (ACC 99.9%). For musk, the curves differ most drastically (Fig. 5.9). It is the smallest data set, containing less than 500 examples, but about 170 attributes. The monotonicity of the AUC plots, well visible e.g. for the KDD Cup data, reflects a high robustness of this metric. AUC and RMSE behave similarly for all data sets. Even for data sets with minor improvements in the limit ADA^{2}BOOST seems to be the more economic choice whenever computational costs are an issue.

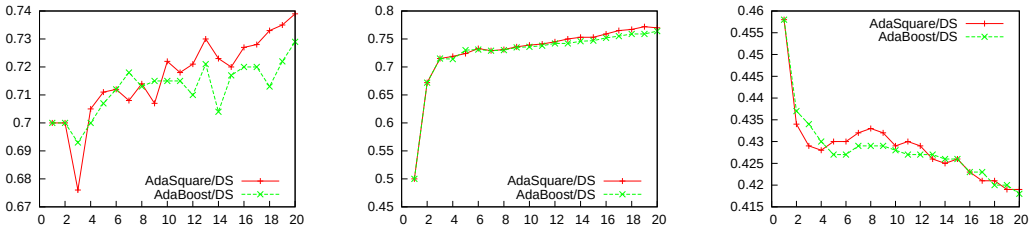
[3] `http://www.ics.uci.edu/~mlearn/MLRepository.html`

[4] `http://yale.sf.net/, operator name: BayesianBoosting, set allow_marginal_skews=true`



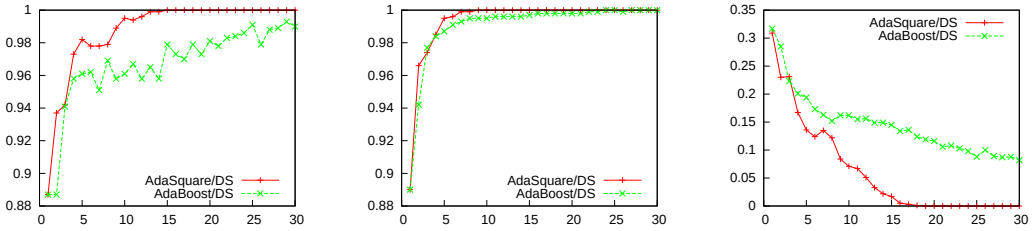
(a) predictive accuracy (b) area under the curve (c) root mean squared error

Figure 5.5.: Adult data set, 15 attributes, 32K examples



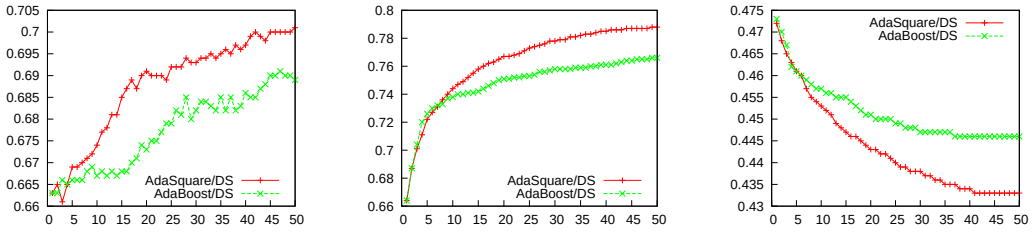
(a) predictive accuracy (b) area under the curve (c) root mean squared error

Figure 5.6.: Credit domain data set, 16 attributes, 690 examples



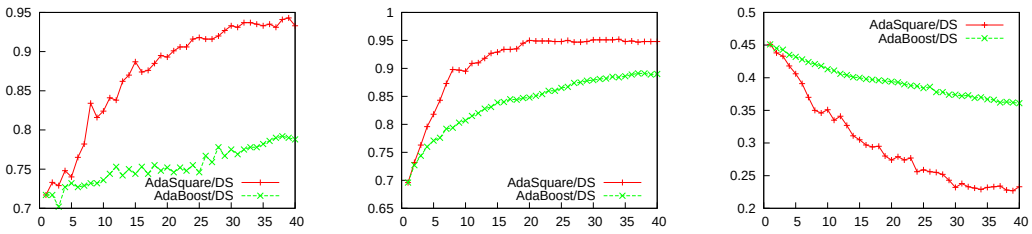
(a) predictive accuracy (b) area under the curve (c) root mean squared error

Figure 5.7.: Mushrooms data set, 23 attributes, 8K examples



(a) predictive accuracy (b) area under the curve (c) root mean squared error

Figure 5.8.: KDD Cup 2004 quantum physics data, 80 attributes, 10K sample



(a) predictive accuracy (b) area under the curve (c) root mean squared error

Figure 5.9.: Musk data set, 169 attributes, 476 examples

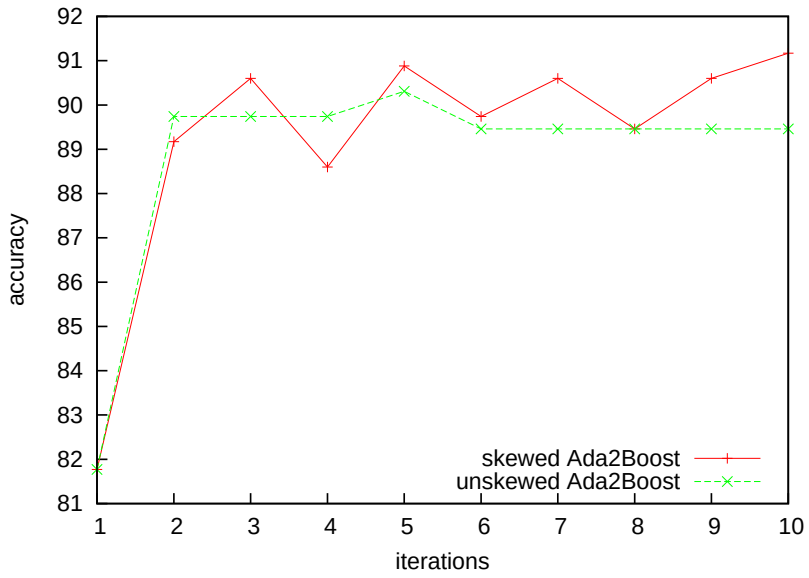


Figure 5.10.: A tenfold cross-validated run of skewed vs. unskewed ADA^{2}BOOST for the ionosphere data set, base learner is CONJUNCTIVERULE. When the data is not skewed artificially the base learner returns the default hypothesis after a few iterations. Skewing helps to overcome this problem and to further increase predictive accuracy.

Figure 5.10 shows the performances of skewed and unskewed ADA^{2}BOOST for the ionosphere data set, also part of the UCI repository. This is an example for which the skewed version of ADA^{2}BOOST proves useful, because the base learner, a rule induction algorithm, ends up returning the default rule after just a view iterations. In contrast, the random skews allow to focus on different subsets in each iteration. Skewing the data for classifier induction deserves further attention, as it provides a simple technique to maximize diversity, but also to increase the significance of findings.

5.5. Conclusions

Boosting is one of the practically most relevant data mining techniques. In this chapter, the knowledge-based sampling approach proposed in the preceeding chapter has been turned into a very efficient boosting strategy. It has been analyzed in coverage spaces, which helped to gain further insights, i.e. why boosting implicitly increases the AUC. Similarities between boosting and rule learning have been illustrated by visualizing the boosting process in the form of nested coverage spaces.

A novel upper-bound for the AUC has been derived based on the total example weight, It holds for ADABOOST, the novel ADA^{2}BOOST algorithm, but also for the confidence-rated REAL ADABOOST. The derived novel ADA^{2}BOOST has been shown to improve the learning rate of ADABOOST when optimizing accuracy and the AUC, and to optimally reduce the total example weight for given crisp base classifiers. The improvements over ADABOOST have been confirmed empirically for different metrics, including accuracy for crisp predictions, and AUC and the root mean squared error metric for estimated conditional class probabilities. ADA^{2}BOOST is

nevertheless a simplification of ADABOOST, as it simply stratifies subsets of the data.

As an additional benefit, it has been shown that ADA^{2}BOOST allows to naturally include biased base classifiers; class ratios can be skewed arbitrarily in order to enforce a minimum confidence of a base model. That way, ensemble methods based on biased classifier induction can be tightly integrated into the framework of boosting.

The similarity between KBS-SD and ADA^{2}BOOST pointed out in this chapter illustrates why sequentially identifying unexpected patterns is a viable approach to both descriptive and predictive data mining. Applicability to data stream mining have, up to now, only briefly been discussed in connection with sampling techniques, mainly addressing the issue of refining the performance estimates of models based on additional data. The next chapter presents a powerful extension of KBS-SD that even allows to boost classifiers induced from data streams suffering from concept drift.

Parts of this chapter are based on the publication (Scholz, 2006).

6. Boosting Classifiers for Non-Stationary Target Concepts

6.1. Introduction

Many real-world data sets are collected over an extended period of time. As shown in chapters 4 and 5, it is possible to induce highly accurate ensemble classifiers from huge amounts of data in constant time, as long as the training data approximately show the characteristics of an i.i.d. sample. Providing on-line predictions for streaming data is a typical application. In domains where the data generating process may change over time, the KDD expert meets with additional burdens. Model induction from outdated training data clearly lacks any justification. As a major problem, common cross-validation strategies may yield overly optimistic estimates if not explicitly taking concept drift into account. Both the training and the validation set are by construction bootstrap samples sharing a single underlying distribution in this case. This distribution is an average over the available training data. At application time, however, it will usually be outdated if the underlying distribution is non-stationary.

Unfortunately, for many domains it is natural that the data-generating distributions are non-stationary. Companies collect an increasing amount of data, like sales figures and customer data, to find patterns in the customer behavior and to predict future sales. As the customer behavior tends to change over time, the model underlying successful predictions should be adapted accordingly. Most recommender-like systems face a similar problem, for example when guiding a user's search on the World Wide Web (Joachims et al., 1997), or when filtering email (Cohen, 1996). The interest of a user, i.e. the concept underlying the classification of texts, and the document content change over time. A filtering system should be able to adapt to such concept changes. Another domain that obviously shares the property of non-stationary distributions is finance, i.e. the continuously changing stock market or business cycles of the global economy. As the last two examples illustrate, the speed of real-world concept drifts may range from sudden changes (crashing stock market) to concepts drifting slowly over years. The term business *cycles* suggests that the accuracy of individual models oscillates over time; later on, older examples may become useful again. To detect the kind of drift underlying a data stream helps to gain additional insights into the domain, and constitutes an analysis goal in its own right.

This chapter extends the knowledge-based sampling algorithm to data streams and makes it capable of adapting to drifting concepts. This will allow to quantify drifts in terms of previously induced base learners. After formalizing the concept drift problem in subsection 6.2.1, previous approaches addressing this problem are briefly outlined in subsection 6.2.2. Section 6.3 discusses related work on ensemble methods for data streams, and introduces a boosting-like algorithm for data streams that naturally adapts to concept drift. In section 6.4, the approach is evaluated on two real-world data sets with different simulated concept drift scenarios, and on one economic data set that exhibits real concept drift. Section 6.5 summarizes the results.

6.2. Concept drift

6.2.1. Problem definition

To a large degree the formal problem studied in this chapter follows the definitions from section 2.1. There is an instance space $\mathcal{X}$, a boolean target attribute $\mathcal{Y} = \{+1, -1\}$, and an underlying probability density function $D : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}^+$. D is changing over time, however: Following the framework of Klinkenberg and Joachims (2000), data arrives in batches. Without loss of generality, these batches are assumed to be of equal size, each containing m examples:

$$\underbrace{e_{(1,1)}, ..., e_{(1,m)}}_{\text{batch } 1}, \underbrace{e_{(2,1)}, ..., e_{(2,m)}}_{\text{batch } 2}, \cdots, \underbrace{e_{(t,1)}, ..., e_{(t,m)}}_{\text{batch } t}, \underbrace{e_{(t+1,1)}, ..., e_{(t+1,m)}}_{\text{batch } t+1}$$

$e_{(i,j)} = (x_{i,j}, y_{i,j})$ denotes the j-th example of batch i. For each batch i the data is sampled i.i.d. from D_i, a batch-dependent pdf. The difference between pdfs D_i and D_{i+1} reflects the amount and type of concept drift. These properties can only be observed indirectly from the corresponding samples. The learner aims to sequentially predict the labels of the next batch and to minimize the cumulated number of prediction errors. For example, after batch t the learner can use any subset of the training examples from batches 1 to t to predict the labels of the subsequent batch $t + 1$.

Learners designed for such batch scenarios can be applied to a continuous stream of examples by imposing batches artificially. Hence assuming batches is no major limitation.

Sudden or abrupt concept drifts, where the complete change of the target concept happens from one example (or batch) to the next, are often called *concept shifts*. In contrast, concept drifts with a slowly changing target concept and the change happening over a longer period of time are often referred to as *continuous* or *smooth concept drifts*.

6.2.2. Related work on concept drift

In machine learning, changing concepts are often handled by moving time windows of fixed or adaptive size over the training data (Mitchell et al., 1994; Widmer & Kubat, 1996; Klinkenberg & Renz, 1998; Hulten et al., 2001) or by weighting data or parts of the hypothesis according to their age and/or utility for the classification task (Taylor et al., 1997). The latter approach of weighting examples has already been used for information filtering in the incremental relevance feedback approaches of (Allan, 1996) and (Balabanovic, 1997).

For windows of fixed size, the choice of a *"good" window size* is a compromise between fast adaptivity (small window) and good generalization in phases without concept change (large window). A fixed window size makes strong assumptions about how quickly the concept changes. While heuristics can adapt to different speed and amount of drift, they involve many parameters that are difficult to tune. The basic idea of *adaptive window management* is to adjust the window size to the current extent of concept drift.

Drifting concepts can also be learned effectively and efficiently with little parameterization by an error minimization framework for adaptive time windows (Klinkenberg & Joachims, 2000) and example weighting or selection (Klinkenberg & Rüping, 2003; Klinkenberg, 2004). This framework makes use of support vector machines (SVMs) and their special properties, which allow for an efficient and reliable error estimation after a single training run (Joachims, 2000).

The methods of the framework either maintain an adaptive time window on the training data (Klinkenberg & Joachims, 2000), select representative training examples, or weight the training examples (Klinkenberg & Rüping, 2003; Klinkenberg, 2004). The key idea is to automatically

adjust the window size, the example selection, and the example weighting, respectively, so that the estimated generalization error is minimized. The approaches are unparameterized, have been analyzed theoretically, and have been shown to work well in practice.

6.3. Adapting ensemble methods to drifting streams

This section presents a novel ensemble method for data streams, which has some appealing properties in the presence of concept drift. Subsection 6.3.1 briefly reviews related work on ensemble methods for non-stationary distributions, 6.3.2 motivates the use of knowledge-based sampling in this context, before the KBS boosting algorithm is adapted along these lines in subsection 6.3.3.

6.3.1. Ensemble methods for data stream mining

In recent years, many algorithms have been specifically tailored towards mining from data streams. The goals of the algorithms vary, depending on the assumed scenarios. Apart from being able to cope with concept drift, scalability is one of the most important issues. As discussed in previous chapters, the induction of classifiers like decision trees is very efficiently possible, even for very large data sets. Using a sampling strategy based on Hoeffding bounds, the VFDT algorithm efficiently induces a decision tree in constant time (Domingos & Hulten, 2000). An extended version of this algorithm updates the tree based on a time window of fixed length, which allows to compensate concept drift up to a certain degree (Hulten et al., 2001). By combining several trees to an ensemble of classifiers, techniques like bagging (Breiman, 1996) and boosting (Freund & Schapire, 1997) have been shown to significantly improve predictions for many data sets (see also subsection 2.6.1 and 5.3.1). For some of these ensemble algorithms, corresponding online variants for *stationary* data streams have been suggested, see e.g. (Oza & Russell, 2001; Lee & Clyde, 2004).

The SEA algorithm (Street & Kim, 2001) induces an ensemble of decision trees from data streams and explicitly addresses concept drift. It splits the data into batches and fits one decision tree per batch. To predict a label, the base models are combined in terms of a uniform voting scheme. As soon as the number of base models exceeds a user specified constant, models are discarded using a heuristic approach. The authors do not report an increase in classification performance compared to a single decision tree learner, but state that the ensemble recovers from concept drifts. The recovery time of this approach seems unnecessarily long, however, as it only exchanges one model per iteration and does not make use of confidence weights.

Another interesting approach proposed by Fan (2004) is based on unweighted base learners, similar to Random Forests (Breiman, 2001). It exploits example selection to include only useful older data into the training set. Examples from earlier batches are only included if predicted correctly by both, the latest model and a recent (assumed optimal) one. Cross-validation experiments may still cause the learner to discard all old examples and to rely on a new model that has been learned from scratch. This allows to adapt more quickly to sudden drift than possible with SEA. However, as the author points out, it is a heuristic selection strategy. Further disadvantages are the required assumption of a fixed marginal distribution, that is a fixed probability to observe a specific instance, regardless of its label, and high computational costs if a different base learner is used.

A recent theoretical analysis suggests, that *weighted* base learners are a preferable alternative in domains with concept drift (Kolter & Maloof, 2005). The ADDEXP algorithm steadily updates

the weights of *experts* (base models in an ensemble), and adds a new expert each time the ensemble misclassifies an example. The new experts start to learn from scratch, using a weight that reflects the loss suffered by the ensemble on the current example. All experts are continuously trained on all new examples.

The main result reported for ADDEXP is a worst case error bound that can be stated in terms of the generalization performance of the best member of its ensemble (expert). This guarantee can only be given if a new base model is added to the ensemble whenever ADDEXP misclassifies a new example, however. This leads to a very large number of experts, which all have to be updated by ADDEXP for each new example. Hence, for practical applications the authors propose to use a computationally more efficient, heuristic variant instead. The goal of ADDEXP is to identify a single best expert, in order to upper-bound the worst case error rate. In contrast, for many applications it is more desirable to minimize the *expected* error rate. As discussed in previous chapters, boosting allows to achieve performances that go beyond those of any individual expert.

More practically oriented work that addresses learning from data streams is described by Stanley (2003) and Wang et al. (2003). The former of these approaches trains a weighted *fixed-size committee* of incremental decision trees, all of which are updated whenever a new example arrives. At each point in time the performance of all base models is estimated based on a *window of fixed size*. Poorly performing models are discarded and replaced by a new decision tree, trained from all subsequently read examples. A disadvantage of this "fixed number of base classifiers" approach is that it does not induce ensembles of *diverse* base models, like boosting algorithms do by appropriately reweighting examples. The approach rather leads to redundant ensembles, because the examples are not weighted individually for different base learners, and the most recent parts of the training sets are identical. Updating all incremental decision trees simultaneously may turn out to be expensive during concept drifts. It still cannot be expected to outperform adaptive time window approaches, as it implicitly maintains heuristically derived weights for the most recent examples.

The approach presented in (Wang et al., 2003) reads training data in batches, and it trains one classifier per batch. Even during stationary phases without drift no classifier is ever trained from more than a single batch. Consequently, large batch sizes are required. The performance of each classifier is estimated based on just the most recent classified batch in each iteration, and the inverse (estimated) error rate is used to weight the model. The authors prove that – given that the estimates are precise – this weighting scheme outperforms a single classifier trained from all the data during concept drift. This result is not surprising when comparing to boosting approaches, where an improvement in accuracy can be expected for each additional base model, especially if exact performance estimates can be provided. However, unlike boosting procedures, the presented approach does not weight examples. As a consequence, introducing diversity into ensembles is possible only by applying heuristics during a pruning procedure.

The algorithm that will be presented in subsection 6.3.3 is adopted from the procedure presented in section 4. It trades predictive performance for scalability. To this end, the online algorithm reads examples aggregated to batches and decides for each batch, whether to add a new expert to the ensemble or not. Unlike in SEA and similar algorithms, the base models of the ensemble are combined by a weighted majority vote. Subsequent models are trained after reweighting the examples of the new batch, and each new base classifier model is assigned a weight that depends on both, its own performance and the performance of the remaining ensemble. Adaptation to concept drift works by continuously re-estimating the weights of all ensemble members, similar to the procedure presented in (Wang et al., 2003), but with each weight fit to the residuals of the (already) weighted ensemble of previous base models. This last aspect is very similar to logistic regression, with the predictions of base models acting as constructed features.

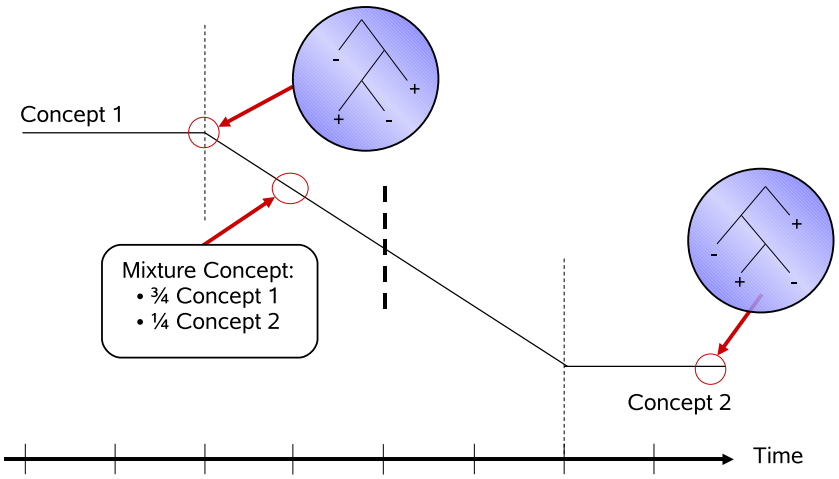


Figure 6.1.: Continuous concept drift, starting with a pure *Concept 1* and ending with a pure *Concept 2*. In between, the target distribution is a probabilistic mixture. It is optimal to predict *Concept 1* before the dotted line, and *Concept 2*, afterwards.

6.3.2. Motivation for ensemble generation by knowledge-based sampling

As a motivation for the subsequently presented knowledge-based sampling (or example weighting) technique, figure 6.1 illustrates the main idea for a simplified concept drift scenario. The underlying assumption is that during a concept drift all examples are sampled from a mixture distribution, which can be thought of as a weighted combination of two pure distributions characterizing the target concepts before and after the drift. In the figure, the initial target concept is simply referred to as *Concept 1*. Examples are sampled from a corresponding stationary distribution up to the first dotted vertical line. A learning algorithm may simply induce a model from all the data, which will predict *Concept 1*. Now, as the drift starts and a *Concept 2* overlaps *Concept 1*, this model will show a decreasing accuracy. Please note that in the intermediate batches the label can best be described as a probabilistic combination of different concepts. The best model during the drift can be derived in terms of Bayes' decision rule. For simplicity we assume diametric concepts in the figure, and a correctly trained model for *Concept 1* when the drift starts. In this situation there is a point in time (a batch), up to which the Bayes' optimal classifier predicts *Concept 1*. This can be concluded from the assumed generative model, because whenever there is a conflict between both concepts, then *Concept 1* is more likely to be correct. The point up to which knowing about *Concept 2* is useless for making predictions is shown as a thick dotted vertical line in figure 6.1. *After* that point it is optimal to predict *Concept 2*. It is not clear how to induce an appropriate model without seeing a few batches sampled from a pure corresponding distribution. Such batches are available to the learner only after the last dotted vertical line, but between the second and last line we would already like to predict *Concept 2*.

One of the main motivations for the boosting-like algorithm presented in this chapter is that, given an accurate model for *Concept 1*, it allows to *decompose* the mixture distribution during the concept drift. Thus, it is possible to construct a sample with respect to *Concept 2* as soon as the drift starts (first dotted line). This look-ahead strategy is inherently different from all approaches discussed in subsection 6.3.1, and it allows to adapt to drifts very quickly. The main reason is that it exploits more of the information encoded into the stream. Please note, that even between the thick and the final dotted lines the model for *Concept 1* is not useless, because it still

helps to "purify" subsequent batches by "subtracting" the deprecated *Concept 1*. The resulting model is a probabilistic ensemble classifier, for which Bayes' decision rule can explicitly be applied when crisp predictions are required.

As discussed in chapter 5, the KBS algorithm is at the same time a boosting algorithm, hence, as for other boosting approaches, the example weights anticipate the expectation given the predictions of previously trained models. This is especially useful for handling smooth concept drifts. While sudden drifts require a quick detection and a way to rapidly adjust the working hypothesis, for smooth drifts it is better to collect information on the new target concept over a period of time. Especially if the preceding concept has been identified accurately at the point in time when a drift starts, removing the knowledge about the current concept from the data allows to decompose mixture distributions as required.

6.3.3. A KBS-strategy to learn drifting concepts from data streams

In this chapter, the KBS-SD algorithm shown on page 89 is adapted, rather than ADA^{2}BOOST. The rationale for this choice is that the difference is small for the studied case of very small batch sizes, but it seems reasonable not to introduce additional marginal skews if the distribution of the data stream itself varies from batch to batch. As discussed in chapter 5, it is sufficient to remove the stratification step of KBS to reach at a boosting algorithm, see also (Scholz, 2005a). The pseudo-code of KBS-SD on page 89 assumes that the complete training set is available for training. The first step to adopt to data streams is hence to read and classify examples iteratively. For subsequent learning steps the reweighting strategy of KBS allows to compute example weights very efficiently. The data is assumed to arrive in batches, each one large enough to train an initial version of a base classifier.

The sizes of the training sets that are effectively used for each model are determined dynamically by the algorithm. Processing a new batch yields two ensemble variants. The first variant appends the current batch to the cache used for training in the last iteration, and it refines the latest base model accordingly. The second variant adds a new model, which is trained using the latest batch, only. Only the ensemble variant performing better on the next batch is kept.

The strategy serves two purposes. First, for stationary distributions a new model is trained only, if there is empirical evidence that this increases the accuracy of the resulting ensemble. This will generally happen if the learning curve of the latest model has leveled off (see Subsec. 3.3.1 or (John & Langley, 1996)), and the data set is well suited for boosting. Second, if sudden concept drift (concept shift) occurs, the same estimation procedure instantly suggests to add a new model, which will help to overcome the drift.

The second step of adopting KBS to data streams is to foresee a re-computation phase in which base model performances are updated with respect to the current distribution. In fact, the author of this thesis believes that this is a main advantage of using weighted ensembles in a concept drift scenario. For stationary distributions the weights vary marginally, while for smoothly drifting scenarios they are systematically shifted and even allow to quantify and interpret the drift in terms of previously found patterns or models. This will be discussed in subsection 6.3.4. Even sudden drifts do not pose a problem, as they automatically result in radically reduced weights of previously trained models, and in high weights of subsequently trained models, if these parameters are re-estimated from new data. The response time to drifts is very short. Since the streaming variant of KBS is closely coupled to the accurate KBS boosting algorithm, the predictive performance is expected to outperform those of single base models for many data sets. Pruning of ensembles can efficiently be addressed during weight re-computation; whenever a model does not lead to any advantage over random guessing on the latest batch, it is discarded. This is a

Algorithm 6 Algorithm KBS-Stream

Initialize empty ensemble $H := \emptyset$.
While not end of stream, do

1. Read next batch $\mathcal{E}_k$ in iteration k.
2. // Prediction rule as in algorithm 2 (p. 89):
   Predict $\hat{y}$ for all $x \in \mathcal{E}_k$ with current ensemble H
3. Read true labels of $\mathcal{E}_k$.
4. If alternative ensemble H^* exists:
   a) Compare accuracy of H and H^* wrt. $\mathcal{E}_k$.
   b) $H \leftarrow$ better ensemble, discard worse ensemble.
   c) If H^* is discarded: $\mathcal{E}^* \leftarrow \mathcal{E}_{k-1}$ (shrink cache to one batch).
5. Initialize D_1: Uniform distribution over $\mathcal{E}_k$.
6. For $i \in \{1, \ldots, |H|\}$, do:
   a) Apply h_i to make predictions for $\mathcal{E}_k$.
   b) Recompute $\text{LIFT}_{D_i}(h_i, x)$ for all $\langle x, y\rangle \in \mathcal{E}_k$ (Def. 33, p. 74).
   c) Update the LIFTs of h_i stored in H. // 4 LIFTs per h_i if $|\mathcal{Y}| = 2$
   d) $D_{i+1}(x, y) := D_i(x, y) \cdot (\text{LIFT}_{D_i}(h_i, x))^{-1}$ for all $\langle x, y\rangle \in \mathcal{E}_k$.
7. Call BASELEARNER($D_{|M|+1}$, $\mathcal{E}_k$), get new model $h_{|M|+1} : \mathcal{X} \to \mathcal{Y}$.
8. Compute $\text{LIFT}_{D_{|M|+1}}(h_{|M|+1}, x)$ for $x \in \mathcal{E}_k$ (Def. 33).
9. Add model $h_{|M|+1}$ (and its LIFTs) to ensemble H.
10. If this is the first batch, then $\mathcal{E}^* = \mathcal{E}_k$ (no alternative ensemble).
11. Else
    a) $\mathcal{E}^* \leftarrow \mathcal{E}^* \cup \mathcal{E}_k$, (extend cache by latest batch)
    b) $H^* \leftarrow$ clone(H)
    c) discard last base model of H^*
    d) repeat steps 5-9 for $\mathcal{E}^*$ and H^* instead of for $\mathcal{E}_k$ and H

natural and common pruning strategy for boosting algorithms, for example also found in the WEKA implementation (Witten & Frank, 2000) of ADABOOST (Freund & Schapire, 1997).

The algorithm is depicted in figure 6. It loops until the stream ends. Lines 1-2 apply the current ensemble to the new batch without knowing the correct labels. Lines 3-4 check whether continuing the training of the latest model with the latest batch outperforms adding a new model trained on that batch[1]. Only the better of these two ensembles is kept.

Lines 5 and 6 recompute the LIFT parameters of all base models. To this end, the models are iteratively applied to the new batch, and the weights are adjusted. This is similar to the learning phase. Finally, lines 8-11 train two variants of the ensemble again, H^* being the one extending the cache and updating the newest model appropriately, and H being the one that adds a new model, which is trained using only the latest batch.

One degree of freedom is left in line 2: The algorithm may use H or H^* to classify the new batch, as the performance of both is unknown at that time. For the experiments two variants have

[1] The pseudo-code does not assume an incremental base learner, but trains new models on cached data. For incremental base learners no caching is required.

been implemented. The first one (KBS_{stream}) always uses ensemble H^*, since models trained from larger batches are generally more reliable. The second variant (KBS_{hold_out}) uses a hold out set of 30% from the latest batch to decide which ensemble to use. Alternatively, one could perform more reliable (but also more expensive) cross-validation experiments, or apply the $\xi\alpha$-estimator for support vector machines (Klinkenberg & Joachims, 2000), which is as efficient as the validation approach suggested in (Fan, 2004). However, in the subsequent experiments the errors caused by simple hold-out estimation are much smaller[2] than those caused by the systematic delay of one batch between the distributions used for training and those underlying the validation data. For this reason more complex kinds of validation are postponed to future work.

If incremental base learners are used, then only the latest batch needs to be stored. The runtime is dominated by adjusting the most recent model to this data, and by applying all base models to it. This avoids the combinatorial explosion and memory requirements of advanced time windowing and batch selection techniques, respectively (see subsection 6.4.1). Unlike the ADDEXP (Kolter & Maloof, 2005) algorithm, which updates all the ensemble members for each new example, KBS_{stream} needs to update or create only two ensemble members for each batch. Incremental variants exist for many popular learning algorithms, in particular for decision trees (Utgoff, 1989) and support vector machines (Rüping, 2001).

6.3.4. Quantifying concept drift

An appropriate combination of several base classifiers often allows to increase predictive accuracy over that achieved by an average single classifier. As a disadvantage, the results lose interpretability to a certain extent. In principle, a similar argument also applies in the context of concept drift. It is interesting to see, however, that the proposed technique allows to extract a different kind of information in this setting at no additional costs: It allows to track the kind of drift underlying the data stream by analyzing the weights of individual base learners.

Please recall, that unlike methods that continuously retrain all models (Stanley, 2003; Kolter & Maloof, 2005), the KBS algorithm "freezes" all models but the latest one. The weights of all frozen models are re-estimated continuously by applying them in chronological order to the current batch, weighting examples accordingly, and by estimating the LIFT ratios of all models based on these weighted examples. This is the same procedure that is employed during the training phase, so any significant deviation from the initial model weights indicates a corresponding change in the underlying distribution. An advantage of using the LIFTs as performance estimates is that the corresponding weight vectors have clear semantics at different points in time, and are thus comparable without any additional artificial normalization. Only the weight of the latest model is not yet comparable between different iterations, because this model is still continuously refined by the boosting procedure. As model weights are re-estimated in chronological order, this has no effect on the remaining ensemble.

The idea of drift quantification is illustrated for the scenario sketched in figure 6.1, which was discussed in subsection 6.3.2 as a motivation for the knowledge-based sampling approach to overcome concept drift. For simplicity, a base learner like a support vector machine is assumed, that continuously improves with additional training data and does not benefit much from boosting. During the stationary distribution before the drift, the KBS algorithm fits a single model to the training data. Let this model capture the deterministic relation between features and label well. It may perform differently well when predicting a positive or a negative label,

[2] There is one exception that will be reported in subsection 6.4.4. It is an extreme case in which the batch size has been chosen much too small for the algorithm.

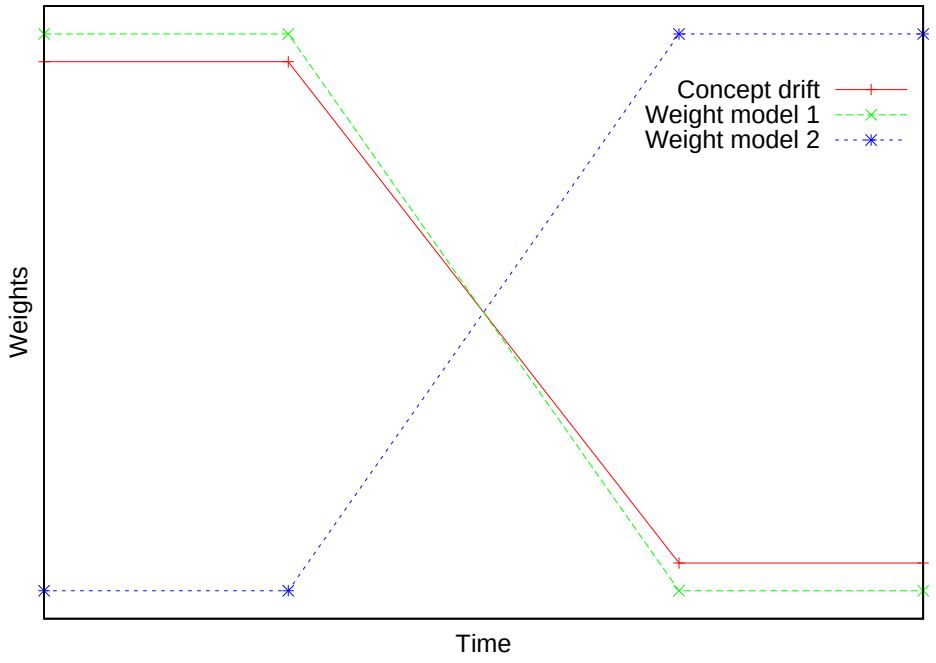


Figure 6.2.: An ideal change of model weights over time. The solid line depicts a drift from an initial to a new target concept. The dotted lines show how the base learner weights reflect the presence of their represented target concepts.

but the LIFTs will vary just marginally from batch to batch, as the underlying distribution does not change. When the drift starts, a significant change in the distribution makes the model perform worse, and the LIFT ratios (positive to negative) slowly approach 1. More interestingly, the reweighted batch suddenly allows to fit a separate model, which, together with the first model, has a higher estimated accuracy. The first model is frozen, but its estimates are continuously updated. The drift continues over several batches, which will first affect the latest model. The small LIFTs of the second model – estimated on the reweighted batch – cause an unbalanced ensemble of these two models, with the first one having a much higher overall impact. The second model is now refined throughout the drift and reflects an increasingly important aspect of the data, while the first model loses accuracy from batch to batch. Consequently, the LIFT ratios and hence the importance of the first model are decreased by the KBS algorithm, in favor of the second one. If the first model has become useless after the drift, it has no significant advantage over random guessing (LIFT ratios ≈ 1), so it is discarded automatically. This is not always the case, because different target concepts often overlap.

Figure 6.2 depicts an ideal change of ensemble weights over time for the sketched scenario. Until the drift starts, the first base model is the only one, assumed to be accurate, and it consequently receives a high weight. This weight is continuously changed based on estimates of the current accuracy. As this weight decreases, the importance and weight of model two increases. In this ideal situation the accuracy is reflected by the maximum of the two dotted lines, which is optimal with respect to Bayes' rule. Subsection 6.4.4 reports corresponding results of experiments with real-world data.

6.4. Experiments

6.4.1. Experimental setup and evaluation scheme

In order to evaluate the KBS learning approach for drifting concepts it is compared to the adaptive time window approach, to the batch selection strategy, and to three simple non-adaptive data management approaches.

Full Memory: The learner generates its classification model from all previously seen examples, i.e. it cannot "forget" old examples.

No Memory: The learner always induces its hypothesis only from the most recent batch. This corresponds to using a window of the fixed size of one batch.

Window of "Fixed Size": A time window of a fixed size of $n = 3$ batches is used on the training data.

Adaptive Window: A window adjustment algorithm adapts the window size to the current concept drift situation (cf. subsection 6.2.2 and (Klinkenberg & Joachims, 2000)).

Batch Selection: Batches producing an error less than twice the estimated error of the latest batch, when applied to a model learned on the latest batch only, are selected for the final training set. All other examples are deselected (cf. subsection 6.2.2 and (Klinkenberg & Rüping, 2003; Klinkenberg, 2004)).

The performance of the classifiers is measured in terms of their prediction errors. All results reported in subsection 6.4.2 and 6.4.3 are averaged over four runs, each based on a different random ordering of the examples in the stream. The results reported in subsection 6.4.4 are from a single run only, because the examples are taken in their real order and no artificial concept drift is simulated or imposed, but there is a real concept drift inherent to this real-world data set.

The experiments were conducted with the machine learning environment YALE (Mierswa et al., 2006), the SVM implementation MYSVM (Rüping, 2000), and two learners from the WEKA toolbox (Witten & Frank, 2000), namely a support vector machine (SMO-SVM) and a decision tree learner (J48), as well as the meta-learner ADABOOST provided by WEKA.

6.4.2. Evaluation on simulated concept drifts with TREC data

The first set of experiments is performed in an information filtering domain, a typical application area for machine learning methods that are able to handle drifting concepts. Text documents are represented as attribute-value vectors (*bag of words* model). Each distinct word corresponds to a feature, the value of which is the "ltc"-TF/IDF-weight (Salton & Buckley, 1988) of that word in each document. The experiments use a subset of 2608 documents of the data set of the *Text REtrieval Conference (TREC)*. Each of the real-world business news texts is assigned to one or several categories, five of which are considered here.

Three concept change scenarios are simulated following the experimental set-up in (Klinkenberg & Joachims, 2000; Klinkenberg & Rüping, 2003; Klinkenberg, 2004). The texts are randomly permutated and split into 20 batches of equal size, containing 130 documents each. In all scenarios, a document is considered relevant at a certain point in time if it matches the interest of the simulated user at that time. The user interest changes between two of the topics, while documents of the remaining three topics are never relevant. Figure 6.3 shows the probability of

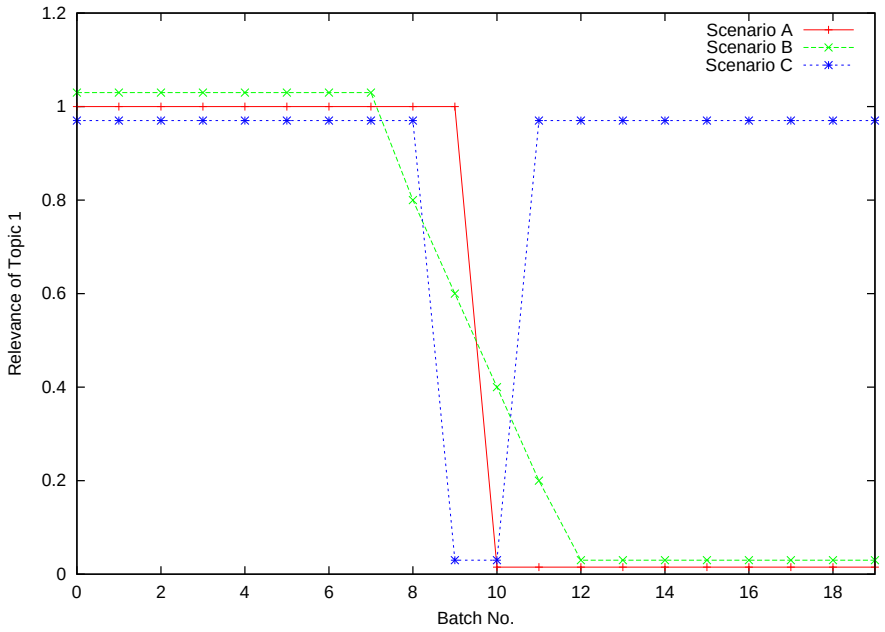


Figure 6.3.: Relevance of the first topic/concept over time in the concept change scenarios A, B, and C, respectively. The relevance of the second relevant topic/concept is *1.0 - relevance of topic 1*.

	Full Memory	No Memory	Fixed Size	Adaptive Size	Batch Selection	KBS stream	KBS hold_out
Scen. A	21.11%	11.16%	9.03%	6.65%	6.15%	6.89%	**5.88%**
Scen. B	21.30%	12.64%	9.76%	9.06%	9.33%	**8.64%**	9.50%
Scen. C	8.60%	12.73%	11.19%	8.56%	**7.55%**	10.11%	8.27%

Table 6.1.: Error of all time window and example selection methods vs. KBS.

being relevant for a document of the first category at each batch for each of the three scenarios; this also implies the probability of the second (sometimes) relevant topic. *Scenario A* is an abrupt concept shift from the first to the second topic in batch 10. In *Scenario B*, the user interest changes slowly from batch 8 to batch 12. *Scenario C* simulates an abrupt concept shift in the user interest from the first to the second topic in batch 9 and back to the first in batch 11.

Table 6.1 compares the results of all static and adaptive time window and batch selection approaches on all scenarios in terms of prediction error (Klinkenberg & Joachims, 2000; Klinkenberg & Rüping, 2003; Klinkenberg, 2003) to the two variants of KBS. The results are averaged over four runs with different random orderings of the examples. In all cases, the learning algorithm was a linear support vector machine.

The KBS algorithm for data streams manages well to adapt to all three kinds of concept drift. Tracking the learners revealed that during stationary distributions the current model was continuously refined. After a concept shift (*scenario A*), a new model was trained and the old model received a significantly lower weight. It was not discarded, however, as it still helped to identify the three topics that are always irrelevant. The hold out set helped to identify the better of the two ensembles reliably at classification time. In *scenario B*, five or more models were

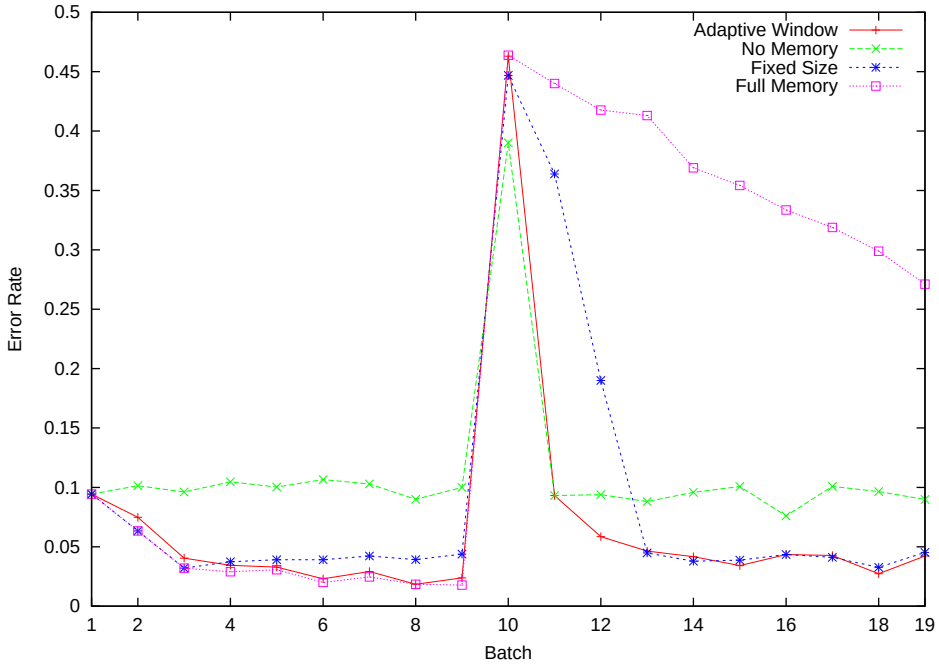


Figure 6.4.: TREC data, scenario A: Error rate over time for the non-adaptive methods versus the adaptive time window approach.

trained. The ensemble accurately adopted to the drift, but at classification time the systematic one-batch-delay of the hold out estimate was sometimes misleading. In *scenario C*, the full memory approach is already competitive to all but the batch selection scenario. Without the hold out set KBS applies the deprecated model one iteration too long for each concept shift. This delay increases the error rate by about 2%. This problem is circumvented by using a hold out set. In essence the KBS algorithm performed very well on this domain, and it even outperformed computationally more expensive approaches. Only in *scenario C* the batch selection method is clearly superior, probably because it is the only method that is able to assemble the data before the first and after the second concept shift into a single training set.

While table 6.1 lists the error rates of the different learning strategies averaged over time, i.e. over all batches, and over all four repeated runs of the experiments, figures 6.4 to 6.7 show the error rates of the different learning strategies over time, i.e. at each batch, also averaged over all runs.

Figure 6.4 compares the non-adaptive methods to the adaptive time window approach in concept drift scenario A. Always learning on all available labeled data and ignoring any possible concept drift that may have happened (*Full Memory*) leads to good generalization performances as long as no concept drift occurs. But as soon as a concept drift occurs, the error rate goes up and only very slowly decreases again, because all the old data no longer representative of the current target concept still is part of the training set and hampers effective learning.

The opposite approach of not storing any old data except for the last labeled batch (*No Memory*) allows a maximally fast adaptation to concept drift, and a correspondingly quick recovery in terms of the error rate. However, the baseline error of this second simple strategy in phases without concept drift is comparatively high, i.e. more than twice as high as that of the other strategies, and hence the overall averaged error rate listed in table 6.1 is also not competitive to other approaches.

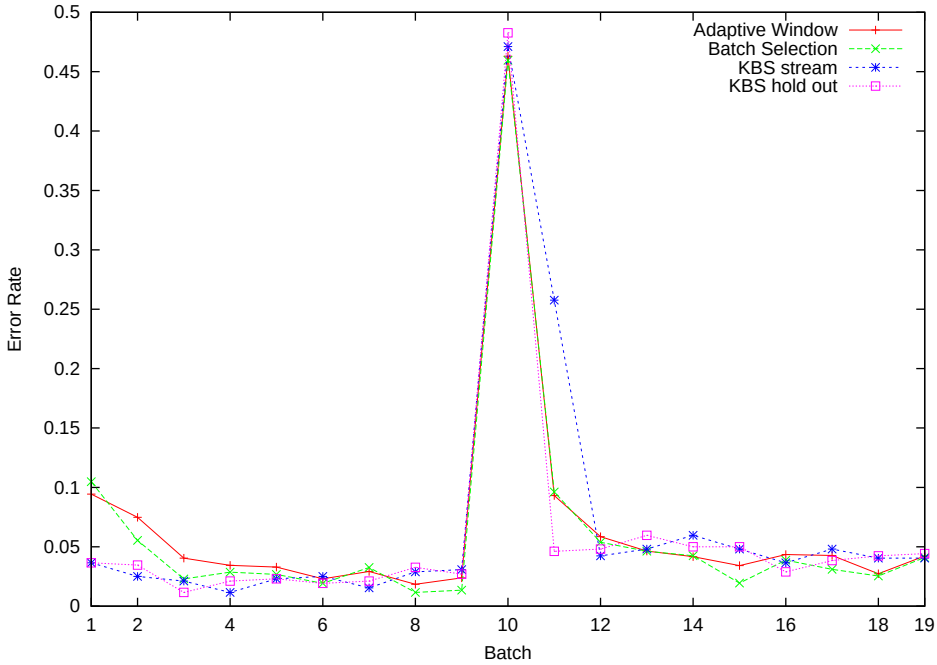


Figure 6.5.: TREC data, scenario A: Error rate over time for the adaptive time window and batch selection techniques versus the two KBS variants.

A sliding time window of *Fixed Size* is a compromise between these two extremes. It has an acceptable baseline error and a better recovery speed than the full memory method. The comparison with the two extremes shows that the performance of such a static window approach is only a compromise and trade-off between adaptivity in phases with concept drift and low error rate in stable phases and hence still leaves a lot of potential for improvements for more adaptive strategies. The described behavior of the non-adaptive methods also explains their high error rates in table 6.1 and motivates the use of adaptive approaches to handling concept drift.

The *Adaptive Window* approach is able to combine the good generalization performance of the full memory method in stable phases without concept drift; it maintains as much (still) representative data as possible, but achieves the fast adaptability of the no memory method by dropping all misleading (old) data immediately as soon as the drift occurs. Hence, the adaptive time window manages to combine the advantages of the two static extremes by adapting to the current extent of drift.

Figure 6.5 compares the adaptive time window and batch selection strategies to the two variants of KBS for data streams in the same concept drift scenario A. Like adaptive time window and batch selection, both KBS variants achieve low baseline error rates and adapt quickly to concept drift. Using a hold-out set allows KBS to adapt to the drift more quickly, and consequently to faster reestablish a predictive model.

Comparing to the adaptive time window and batch selection strategies in concept drift scenario B, a similar behavior of both KBS variants concerning the low base line error and the adaptability to the drift can be observed (Fig. 6.6). The same applies in concept drift scenario C (depicted in Fig. 6.7) if KBS uses a hold-out set, but KBS does not adapt as quickly without the hold-out set. In this scenario, that involves the detection of a re-occurring target concept, the batch selection strategy has the advantage of being able to re-use old data from before previous concept drifts, and to do so quickly. It thereby slightly outperforms the other approaches.

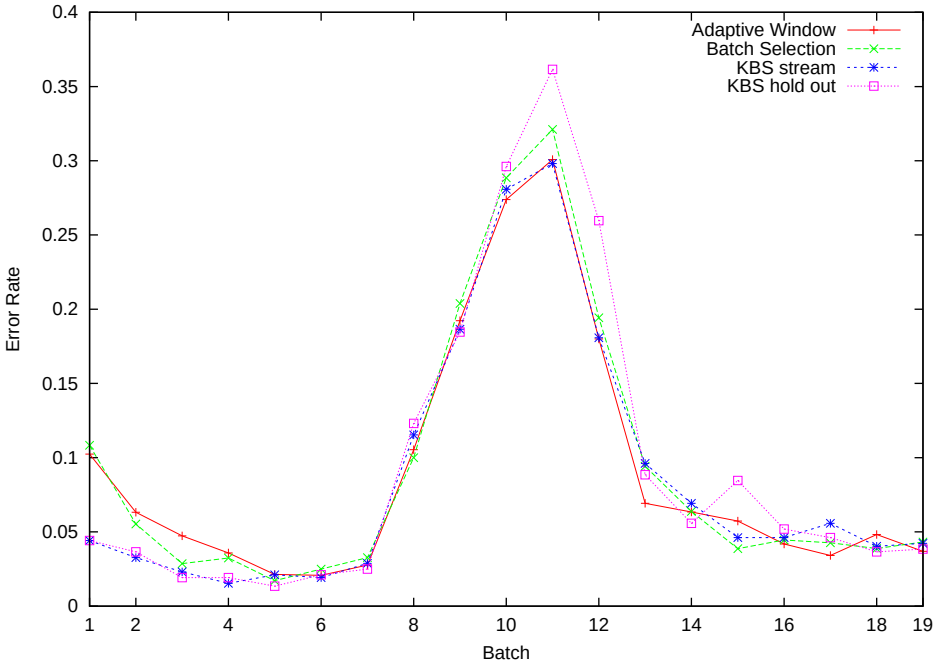


Figure 6.6.: TREC data, scenario B: Error rate over time for the adaptive time window and batch selection techniques versus the two KBS variants.

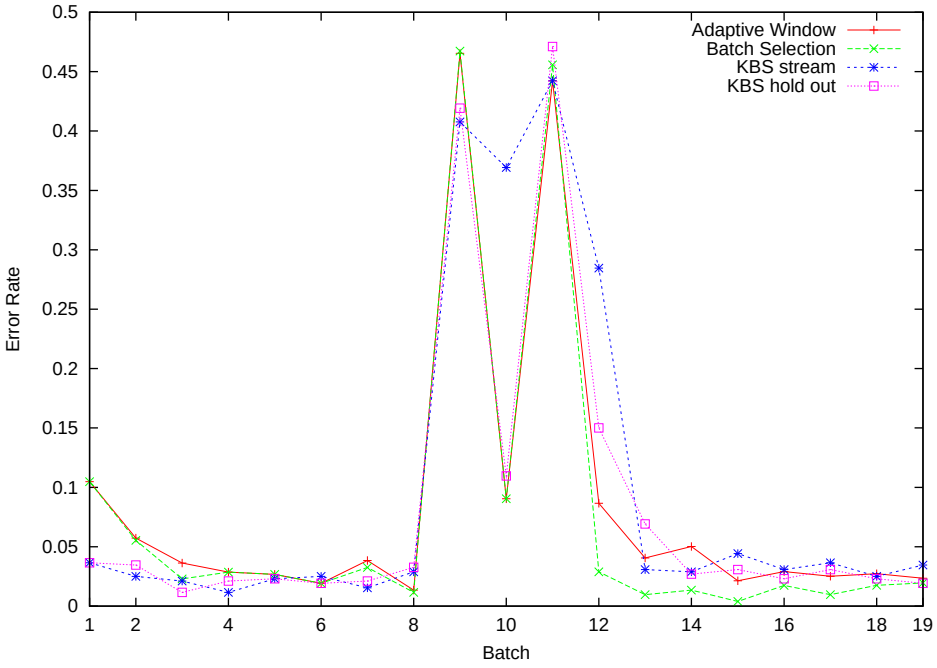


Figure 6.7.: TREC data, scenario C: Error rate over time for the adaptive time window and batch selection techniques versus the two KBS variants.

	J48 Fixed Memory	J48 Full Memory	ADABOOST+J48 Full Memory	KBS stream	KBS hold_out
Scen. A	11.06%	21.65%	20.51%	9.50%	**9.43%**
Scen. B	11.25%	21.22%	19.93%	11.60%	**10.92%**
Scen. C	12.70%	10.83%	**9.50%**	11.55%	10.45%

Table 6.2.: Averaged prediction errors for satellite image data set.

However, even in this scenario, and even more so in the other scenarios, KBS performs competitively well or even better, as shown by the behavior over time in the plots and the overall average error rate in table 6.1.

6.4.3. Evaluation on simulated drifts with satellite image data

The second set of experiments uses the satellite image data set from the UCI library (Blake & Merz, 1998), a real-world data set for classification. It contains no (known) drift over time, so the concept drifts were simulated using the same techniques as described in subsection 6.4.2: The data set was randomly permutated and split into 20 batches of equal size (321 examples per batch). Only two (*grey soil* and *very damp grey soil*) of the six classes were selected to be relevant. The same drift scenarios A-C as described in subsection 6.4.2 were simulated with this data set, where the two selected classes corresponded to the two selected topics in the TREC experiments. The reported results are averages over four runs with different random orderings of the examples.

In order to be able to compare the novel KBS algorithm to boosting techniques that ignore concept drift, a more typical setting for boosting was chosen. The decision tree induction algorithm J48 from the WEKA toolbox was selected as a base learner. It was applied in combination with ADABOOST (Freund & Schapire, 1997), but also as a stand-alone learning technique. Besides, KBS was compared to the non-adaptive "fixed size" window strategy with a window size of 3 batches.

The default settings of the learners were used in all runs. The results are listed in table 6.2. Unlike for the experiments on the TREC data, the results of KBS could always be improved by using a hold out set. The fact that *scenario C* can well be tackled by a full memory approach is reflected by the good performance of ADABOOST in this setting. However, it performs much worse for the other two scenarios, similar to the full memory approach applying J48 stand-alone. The fixed memory approach and KBS show a comparatively stable error rate over all scenarios. KBS performs better than the fixed size window learner, and much better than the full memory approach.

6.4.4. Handling real drift in economic real-world data

The third evaluation domain is a task from economics. It is based on real-world data that exhibits a factual concept drift. The quarterly data describes the West German Business Cycles From 1954 to 1994 (Heilemann & Münch, 1999). Each of the 158 examples is described by 13 indicator variables. The task is to predict the current business cycle phase of the West German economy. In accordance to findings from Theis and Weihs (1999), we use two phases instead of four for the description of the business data, just as described by Morik and Rüping (2002).

The following experiments compare the performance of the KBS data stream algorithm to results reported previously by Klinkenberg (2003), so the same number of 5 and 15 batches

	Full Memory	No Memory	Fixed Size	Adaptive Size	Batch Selection	KBS stream	KBS hold_out
5 batches	32.80%	27.20%	24.00%	24.80%	24.80%	24.60%	**17.46%**
15 batches	28.08%	28.77%	**20.55%**	24.80%	23.29%	26.03%	28.77%

Table 6.3.: Prediction error for business cycle data

were used. The timely order of the examples (quarters) was preserved and no artificial concept drift was simulated.

The results of these two evaluations are shown in table 6.3. The column for the fixed time window approach lists the results for the fixed size that performed best. The fact that this approach performs well may be due to the cyclic nature of the domain. However, the size is generally not known in advance, and as shown by Klinkenberg (2003), using other fixed window sizes leads to significant decreases in performance. The results for 15 batches show that KBS does not perform well if each batch consists of less than a dozen examples. The reason is that it is not possible to get reliable probability estimates from such small data sets. The algorithm could cache older data in such cases, but it is more reasonable to choose larger batch sizes. Using just 5 batches (31 examples) already improves the situation, so KBS performs similar to the fixed size, adaptive size, and to the batch selection approach. The hold out set turned out to be surprisingly effective for larger batches. This result provides first evidence that KBS is able to adapt classifier ensembles to different kinds of concept drift found in real-world data sets.

6.4.5. Empirical drift quantification

The final experiments of this chapter investigate whether the claims made in subsection 6.3.4 are realistic in practice. Two examples extracted during real experiments with the TREC data are presented to illustrate, how KBS base model weights allow to characterize kind and intensity of concept drifts in practice. No sophisticated methods for pruning or model evaluation during learning were applied, in order not to change any results. In the experiments, the four LIFT values of models making boolean predictions were reduced to a single weight per model. Applying the same strategy as in the proof of Prop. 11 (p. 114) the model that estimates odds ratios can be transformed into a classifier of the form

$$\hat{y} := \operatorname{sign}\left(w_0 + \sum_{i=1}^{n} w_i h_i(x)\right),$$

with offset weight w_0 and model weights $w_1, \ldots, w_n \in \mathbb{R}$. Weight vectors could be normalized, but are not in the following figures, in order to ease the comparison of model impacts from one iteration to the next.

Figure 6.8 exemplarily shows the weights of the base classifiers during a KBS application over time. As before, the algorithm is applied to the TREC data set with a simulated concept shift (*scenario A*), using a support vector machine with linear kernel as the base learner. The base models are all trained over a period of time, and afterwards only their weights are adjusted. The performance of the initial model is directly estimated from the (unweighted) most recent batch. Model weights are upper-bounded artificially in the figure to ease visualization. Until the concept shift occurs in the middle of the figure, the first model is refined by extending the training set batch by batch. That way, the model reaches high confidence which varies slightly due to estimations based on small batches (130 examples). The first batch sampled from the

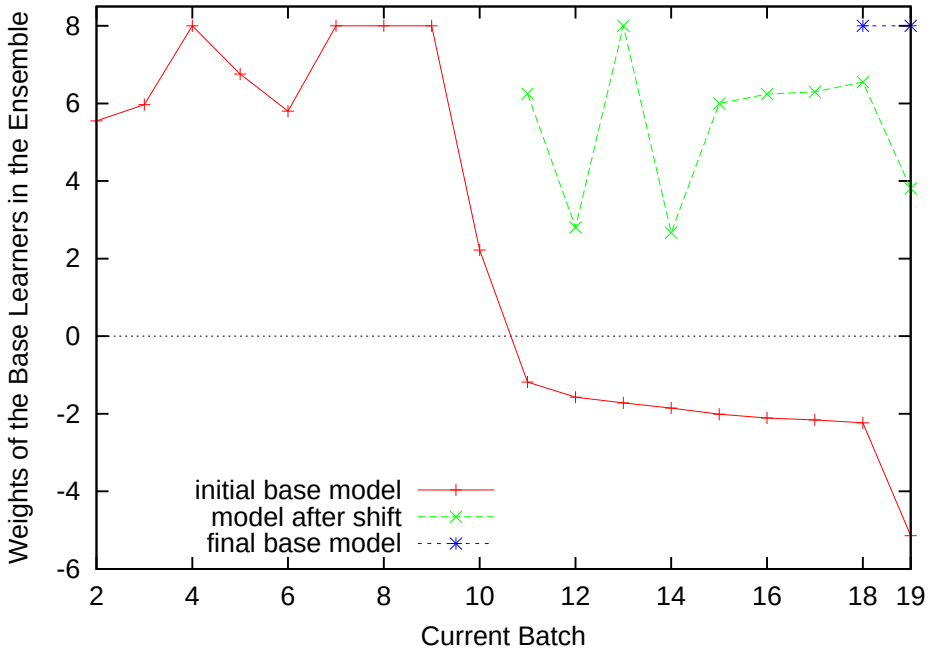


Figure 6.8.: Base model weights of a KBS ensemble for the simulated scenario A on the TREC data. Only the most recent model is refined, the others are frozen and just continuously reweighted with respect to the latest batch. The concept shift occurs where the weight of the initial model drops drastically (batch 10).

new distribution already decreases the weight of the initial classifier rapidly. The classifier is "frozen", and KBS introduces a second classifier, which is now refined for several iterations. The first model still turns out to be useful, but with a negative weight, which indicates that the opposite of the initial target concept is correlated with the new target concept. The precise weights of both models vary slightly, but converge after a while. Refining the second model by further examples does no longer improve the overall accuracy, so at this point the KBS estimator freezes the second model as well, and it introduces a third one. This step allows to increase the expressiveness of the underlying model language wherever this seems promising.

The second experiment provides a realistic counterpart to the motivating example with slow concept drift (*scenario B*), which has been presented in subsection 6.3.1. Figure 6.9 shows how the weights of all involved base models change over time. Just one "outlier model", which is directly removed from the ensemble by the KBS algorithm after induction, has been removed from the figure, in order not to overload it. The initial model reaches a high weight during the first stationary phase, which reflects highly confident predictions. The confidence decreases rapidly during the drift, and after only a few batches sampled from a new stationary target distribution the initial model is even discarded by the learner (batch 16). Two new models are introduced during the drift, which both quickly lose weight as the first target concept diminishes. Please recall, that KBS reweights the batches as if they were sampled from the pure target distribution of the new concept. In this sense, the early batches during the drift can be considered to have a higher variance than later ones, which explains the decreasing weights. The learner still fits each classifier based on a couple of consecutive batches during the drift. Reaching the new stationary distribution, the weights of both intermediate models converge, because they contain a fixed

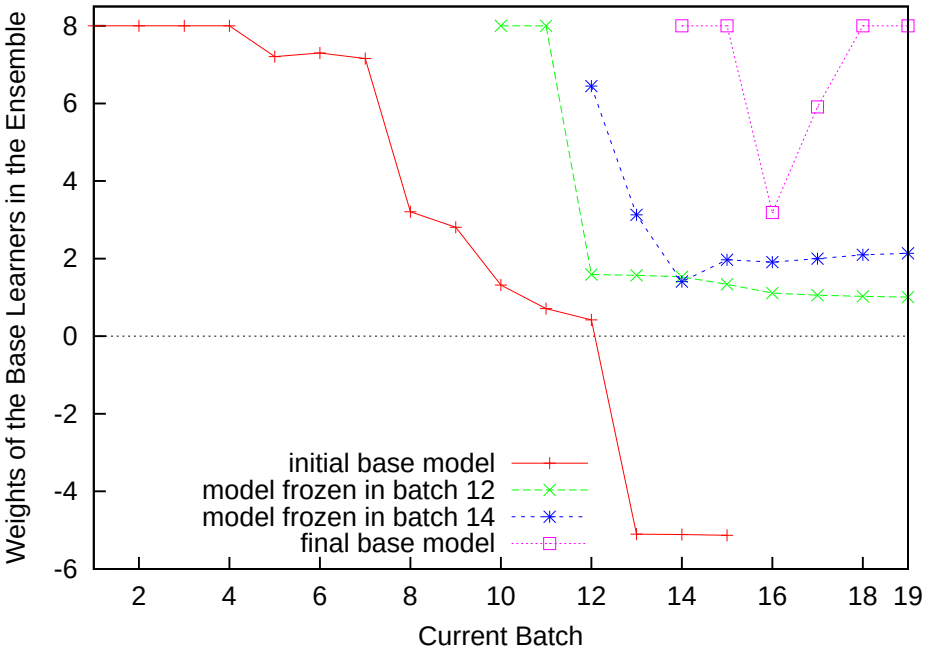


Figure 6.9.: Base model weights for another KBS ensemble trained for the simulated concept drift scenario B on the TREC data.

amount of information on the new target concept. The final model is induced after the drift ends, at a point where the previous model weights have almost converged.

Although the curves are not as simple as in the ideal case sketched earlier, the example illustrates how the weights of base learners can be used to identify the kind and degree of a concept drift underlying a data stream. A higher robustness of the sketched quantification property can be expected when choosing a larger batch size for estimating base model performances.

6.5. Conclusions

In this chapter the KBS-algorithm has been adapted to the task of inducing classifiers from data streams with concept drift. At each iteration base models are induced and reweighted continuously, considering only the latest batch of examples. The proposed strategy adapts very early and quickly to different kinds of concept drift. The algorithm has low computational costs. It has empirically been shown to be competitive to, and often to even outperform more sophisticated adaptive window and batch selection strategies. As a further advantage, it allows to track the kind and degree of concept drift.

Remaining directions for future work include evaluations of more precise and robust strategies for estimating model weights, and the development of models for predicting kind and degree of future concept drifts based on the drift quantification of KBS .

This chapter is an extended version of (Scholz & Klinkenberg, 2005) and a condensed version of (Scholz & Klinkenberg, 2007) with a stronger focus on the novel aspects. For a late draft of the latter publication please refer to (Scholz & Klinkenberg, 2006).

7. Distributed Subgroup Discovery

7.1. Introduction

The amounts of data collected and processed in huge modern companies, the many heterogeneous groups of users accessing it for very different purposes, coupled with a number of technical burdens and legal issues define the daily situation in modern data-warehouses. For KDD applications it is generally assumed, however, that all the data to be analyzed are accessible in the form of a single local flat file. Issues like handling huge amounts of data without loss of useful information have been addressed in previous parts of this thesis, data cleaning and identifying a well-suited representation for learning will be discussed in chapters 8 and 9.

Since many real-world databases are distributed to different nodes, e.g., each capturing the sales of different stores, one practically relevant question is whether the same data mining tasks can be solved from distributed data as well as if collecting all the data physically at a single site. Distributed data mining algorithms are designed to work with geographically distant databases that are connected by a communication network. The major bottleneck for distributed algorithms is communication. Therefore, the aim in the design of such algorithms is to minimize communication costs. Learning tasks can be adopted to distributed scenarios in various ways. This chapter analyzes distributed variants of *rule selection*, a general task that – in previous chapters – has been shown to apply to both descriptive and predictive data analysis. Due to its generality, the task of subgroup discovery fits well into this framework. It is very flexible, because it allows to specify the utility function used for pattern selection as a parameter.

At first sight, distributed data mining seems to be a promising approach, e.g., to decrease computational costs if coupled with parallel model induction. Still, only a few tools support parallel model induction. As its first major contribution, this chapter points out some strong negative results for distributed rule discovery. These results shed light on the question why such distributed approaches are not as popular in practice as the potential benefits might suggest. It is investigated systematically in which situations a local evaluation of rules may help to identify globally best rules, and how corresponding learning tasks are related to each other. In principle, the subsequently derived results apply to a much broader category of supervised learning tasks, because the model class is of minor importance compared to the evaluation functions.

The theoretical findings suggest that approaches applying other than exhaustive search strategies may fail to give reasonable guarantees, or may cause even higher communication costs in distributed settings. For this reason, two exhaustive algorithms for distributed subgroup discovery are presented, analyzed, and empirically evaluated.

This chapter is structured as follows: Section 7.2 provides a more general definition of utility functions for the rule selection problem from non-distributed data sets; a very broad definition of utility functions allows to subsume most of the relevant learning problems under the notion of subgroup discovery. Section 7.3 extends subgroup discovery to distributed data, assuming a homogeneous distribution at all sites. This assumption is weakened in section 7.4 in two ways, which are both shown to increase the computational complexity of finding a set of approximately best rules in the worst case. Additionally, a bound for the maximal deviation of commonly used utility functions is derived. This motivates the task of relative local subgroup discovery, which is

introduced and analyzed in section 7.5. Section 7.6 discusses how the presented tasks are related to distributed boosting and distributed frequent itemset mining. After discussing some practical considerations regarding the design of specific algorithmic solutions, two novel algorithms for exact subgroup discovery from distributed data are presented in section 7.7. The theoretical findings regarding communication costs are empirically evaluated in section 7.8. Section 7.9 summarizes and concludes.

7.2. A generalized class of utility functions for rule selection

Based on the definitions given in chapter 2, this section broadens the formal problem of subgroup discovery from non-distributed supervised rule learning. Given is a set of m classified examples $\mathcal{E} := (x_1, y_1), \ldots, (x_m, y_m)$ from $\mathcal{X} \times \mathcal{Y}$, where $\mathcal{X}$ defines an instance space and $\mathcal{Y}$ a set of labels. Classification rules are used as the representation language $\mathcal{H}$.

In order to be able to provide very general results, the considered notion of *utility functions* should be as broad as possible. Coverage and bias (Def. 17 and 18, p. 22) allow to state an unusually broad class of utility functions that are still covered by the subsequent analysis.

Definition 39 *For the set $\mathcal{D}$ of all probability density functions (pdfs) $D : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^+$ a function $f : \mathcal{H} \times \mathcal{D} \to \mathbb{R}$ is called a* utility function *if and only if it satisfies the following constraints for all $r, r' \in \mathcal{H}$:*

$$(\text{Cov}_D(r) \geq \text{Cov}_D(r')) \wedge (\text{Bias}_D(r) \geq \text{Bias}_D(r') > 0) \;\Rightarrow\; f(r, D) \geq f(r', D)$$

- *If one of the inequalities is strict, then $f(r, D) > f(r', D)$.*
- *All rules r with $\text{Bias}(r) = 0$ are assumed to receive a common utility score, e.g. 0.*

The main objective of a utility function is to trade coverage for bias. In this light, definition 39 can be considered to cover all reasonable utility functions, because its only constraints are (i) monotonicity in these two quantities and (ii) that the same score is assigned to all rules that perform as well as random guessing.

The definition is broad enough to also cover predictive accuracy, which is equivalent to WRAcc for binary prediction tasks with equal default probabilities for both classes, and which is still monotone in Cov and Bias, otherwise.

In association rule mining (Agrawal & Srikant, 1994) rules are filtered (or pruned) by their supports (Cov) and confidences. The latter is monotone in the Bias, although the default probability is usually ignored. If support and confidence are combined (respecting monotonicity) to find a ranking of most interesting rules, this problem can also be subsumed under the task of subgroup discovery with a utility function that fits Def. 39.

For some parts of this chapter it is assumed that all utility functions under consideration are elements of a more restricted class, however, which was previously defined in section 2.4:

Definition 40 *For a given parameter α and pdf D the utility (or quality) $Q_D^{(\alpha)}$ of a rule $r \in \mathcal{H}$ is defined as*

$$Q_D^{(\alpha)}(r) := \text{Cov}_D(r)^\alpha \cdot \text{Bias}_D(r).$$

As discussed, this class covers metrics that are factor-equivalent to the binomial test function ($\alpha = 0.5$) and the weighted relative accuracy (WRAcc, $\alpha = 1$). Hence, it can be considered to contain the most important utility functions.

7.3. Homogeneously distributed data

For any specific choice of a utility function, the goal of subgroup discovery is to identify a set of k best or approximately best rules. A description of existing approaches for this task, containing exhaustive, sampling-based, and heuristic search strategies, was given in subsection 2.2.3 (p. 11).

A first extension towards distributed subgroup discovery is to assume that several sets of data are available, which all follow a common underlying probability distribution. One can think of the sets at the different sites as being generated by bootstrapping from a single, global data set. In such a case, local and global subgroups are basically identical. However, due to statistical fluctuations caused by the bootstrapping procedure and the smaller size of example sets, some of the rules with lower global utility might be found among the k best subgroups evaluated locally at each site.

For the WRAcc metric the probability that the utility function deviates locally from the true (global) value by more than a fixed constant $\epsilon \in \mathbb{R}^+$ can e.g., be bounded by Hoeffding's inequality (theorem 4, p. 45). This probability decreases exponentially fast with a growing number of examples.

Alternatively, the sample bounds for adaptive sampling discussed in subsection 3.3.2 may be used. They also apply for other values of α. In the context of distributed databases it is easy to analyze large local samples at each site. Definitions 39 and 40 define utility functions with respect to an underlying pdf D, which allows to address predictive and descriptive tasks in a single framework. Please recall that the latter kind of tasks is more specific, as it introduces the assumption of a uniform distribution $D_{\mathcal{E}}$ over a given example set $\mathcal{E}$. In the general case, the available example set of size m is considered to be a sample $\mathcal{E} \sim D^m$, so a natural choice of a learning task is the approximately k-best rules problem (Def. 28, p. 51). The results reported for this problem directly apply to homogeneously distributed data sets. For large local data sets the probability of missing a subgroup that is globally much better than the locally best ones is reasonably small.

As discussed in subsection 3.3.2 there are also some practically relevant evaluation metrics that do not allow to tackle the approximately k-best hypotheses problem by adaptive sampling. One example is the Gini index, for which sampling-based utility estimates can be far from the true utilities, regardless of the sample size. For these utility functions, distributed subgroup discovery from local data becomes intractable. The following sections address more complex learning problems; it is reasonable to focus on utility functions for which learning is tractable at least in the case of homogeneously distributed data, i.e. instances of the class described by Def. 40.

7.4. Inhomogeneously distributed data

Subgroup discovery for homogeneously distributed data can be tackled and analyzed using the same techniques as in the non-distributed setting. This section addresses the situation in which data is split to different sites, but no distributional assumption can be made. First of all, the notation for different databases is introduced.

Any example set $\mathcal{E}$ is composed of s subsets $\mathcal{E}_1, \ldots, \mathcal{E}_s$ that were sampled from different probability distributions. Let D_i denote the probability density function at site i, $\mathcal{E}_i \subseteq \mathcal{E}$ be a corresponding example set, and let D define the global densities of $\mathcal{E}$. D is a weighted average of the local density functions.

The *local* Cov and Bias of a rule $A \rightarrow C$ at site i can be expressed in terms of definitions 17

and 18 (p. 23), replacing D by D_i. For example,

$$\text{BIAS}_{D_i}(A \to C) := \Pr_{(x,y)\sim D_i}[y = C \mid x \in A] - \Pr_{(x,y)\sim D_i}[y = C]$$

refers to the local BIAS at site i. Accordingly, a local utility function $q^{(\alpha)}_{D_i}(r)$ evaluates each rule $A \to C$ by computing

$$q^{(\alpha)}_{D_i}(A \to C) := [\text{COV}_{D_i}(A \to C)]^{\alpha} \cdot \text{BIAS}_{D_i}(A \to C),$$

which yields the following definition for the specific case of $\alpha = 1$:

Definition 41 *The local weighted relative accuracy of rule r at node i for a local pdf D_i is defined as*

$$\text{WRACC}_{D_i}(r) := \text{COV}_{D_i}(r) \cdot \text{BIAS}_{D_i}(r).$$

The first task stated in this setting is to find subgroups that globally perform well, given a discovery procedure that evaluates rules locally. If, for instance, the globally best rule appears poor at any site, then it obviously needs to perform even better at some other. For this reason one could expect that the globally best rules are easily found at the local sites, even if the local distributions differ. A similar property eases frequent itemset mining from distributed data, because it allows for safe pruning in the case of skewed data (Cheung & Xiao, 1998).

In the case of homogeneously distributed data, assumed in section 7.3, the marginal distributions over $\mathcal{X}$ and the conditional probabilities of the target given $x \in \mathcal{X}$ were identical at all sites. The following definitions are useful for quantifying by how much each of these assumptions is weakened in more general settings.

Definition 42 *Two density functions $D_1, D_2 : \mathcal{X} \to \mathbb{R}^+$ are called* factor-similar *up to a constant γ for an $A \subset \mathcal{X}$ and $\gamma > 1$, iff*

$$(\forall x \in A) : \gamma^{-1} \leq \frac{D_i(x)}{D(x)} \leq \gamma.$$

Definition 43 *For an $A \subseteq \mathcal{X}$ two joint density functions $D_1, D_2 : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^+$ are called* conditionally similar up to ϵ, $\epsilon > 0$, *iff*

$$(\forall (x,y) \in A \times \mathcal{Y}) : \left| \frac{D_1(x,y)}{D_1(x)} - \frac{D_2(x,y)}{D_2(x)} \right| \leq \epsilon.$$

Please note that definitions 42 and 43 do not necessarily require the same set of examples to be observable at all sites to allow for finite constants γ and ϵ, because the utility functions are defined based on the underlying probability density functions.

The following theorem shows, that if the assumption of homogeneously distributed data made in section 7.3 (a form of i.i.d. sampling) is weakened at all, then it is possible to obtain drastically different sets of best rules when evaluating a quality function globally and locally.

Theorem 10 *Let G_i denote the set of k best rules for each site $i \in \{1, \ldots s\}$ ($s \geq 2$), given an arbitrary utility function. Let G denote the set of k best rules with respect to the global distribution. Then, in the general case, it is possible that every $x \in \mathcal{X}$ is covered by at most one rule set from $\{G, G_1, \ldots, G_s\}$, where a rule set is said to cover x if one of its elements does. This statement even holds in the following two cases:*

1. *The global and local marginal distributions of $\mathcal{X}$ are equivalent, and global and local joint distributions of $\mathcal{X} \times \mathcal{Y}$ are conditionally similar up to an arbitrarily small $b > 0$.*

2. *For all local sites $i \in \{1, \ldots, s\}$ the conditional distributions of $\mathcal{X} \times \mathcal{Y}$ are identical, and each local marginal distribution of $\mathcal{X}$ is factor-similar to the global one up to an arbitrarily small $\gamma > 1$ for any subset of $\mathcal{X}$.*

Proof
It is sufficient to generically construct an example for both specific cases. The following proofs apply to all utility functions covered by Def. 39, but require some assumptions about the set $\mathcal{H}$ of possible hypotheses. These assumptions are met for the logical rules commonly used to characterize subgroups.

First, the theorem is shown to hold in the case of equal marginal distributions ($\gamma = 1$). The idea is to "prepare" for each site $i \in \{1, \ldots, s\}$ a set S_i of k disjoint subsets of $\mathcal{X}$: $S_i = \{R_{i,1}, \ldots, R_{i,n}\}$. For the global view, a separate set $S_0 = \{R_{0,1}, \ldots, R_{0,n}\}$ of k rules is prepared. Let the common marginal density function D assign equal weights to each subset, so that all rules with antecedent $R \in \bigcup_{i=0}^{s} S_i$ have the same coverage COV. All reasonable utility functions increase monotonically with the BIAS, in this case. Let C denote an arbitrarily chosen class and b and ϵ small, strictly positive real values. The joint density function $D_i : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^+$ at site i is constructed so that

$$\text{BIAS}_{D_i}(R_{p,j} \to C) = \begin{cases} b/s + \epsilon, \text{ for } p = 0 \text{ (global)} \\ b \qquad\quad , \text{ for } p = i \text{ (local)} \\ 0 \qquad\quad , \text{ for } p \notin \{0, i\} \end{cases}$$

for all $1 \leq j \leq k$. The joint global density function $D : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^+$ is the average of the joint local densities, because the marginal distributions are assumed to be equivalent. Hence, the BIAS of every "local rule" $R_{i,j} \to C$, $i > 0$ is b/s under D, that of the "global rules" $R_{0,j} \to C$ is $b/s + \epsilon$ at all sites and when evaluating globally. As a consequence, under D_i the k rules constructed from R_i are ranked highest by all reasonable utility functions, but globally the rules corresponding to R_0 have a higher utility.

It remains to be shown that a pdf as described above exists. An additional constraint is, that no other rule in $\mathcal{H}$ may reach a higher utility score, neither at any local site nor globally. The following construction is possible if $\mathcal{H}$ contains only single rules $A \to C$ with each A being a conjunction of literals. For s sites and k rules to be selected let

$$z := \lceil \log_2(s+1) \rceil \cdot \lceil \log_2(2k) \rceil.$$

For at least one set of z atomic formulas $\{a_1, \ldots, a_k\}$ it is assumed that

$$\{l_1 \wedge \ldots \wedge l_z \to C \mid l_i \in \{\neg a_i, a_i\} \text{ for } 1 \leq i \leq z\} \subseteq \mathcal{H}.$$

For all considered rules literal l_i refers to the same atomic formula, but it may be positive or negative. Each of the rules may be represented by a boolean vector of length j, where the i^{th} bit refers to the sign of literal i. In turn, each vector $\vec{v}$ of length j represents a rule $(A_{\vec{v}} \to C) \in H$, and for two such vectors $\vec{v}_i \neq \vec{v}_j$ we have $\text{Ext}(A_{\vec{v}_i}) \cap \text{Ext}(A_{\vec{v}_j}) = \emptyset$.

Now the bit representation can be used to define the sets $R_{i,j}$ for $0 \leq i \leq s$ and $1 \leq j \leq k$ from above: We set the first $\lceil \log_2(s+1) \rceil$ to the binary encoding of the corresponding site number i and let the subsequent $\lceil \log_2(k) \rceil$ bits encode rule number j. Each combination of i and j covers two subsets now, since there is one more bit/literal. The subset defined by an even

number of positive literals is defined as positive ($R^+_{i,j}$), the other one as negative ($R^-_{i,j}$). The following equalities imply a common marginal distribution:

$$\int_{x \in R^+_{i,j}} D(x)dx = \int_{x \in R^-_{i,j}} D(x)dx = \frac{1}{(s+1)2k}$$

$$D(x) = D(x') \text{ if } x, x' \in R^+_{i,j} \text{ or } x \in R^+_{i,j} \wedge x' \in R^-_{i,j}.$$
$$D(x) = 0 \text{ if } x \notin \bigcup_{i=0}^{s} \bigcup_{j=1}^{k} \left(R^+_{i,j} \cup R^-_{i,j} \right)$$

For two classes and a prior class probability of p_0 we define the joint density function at site $i \in \{1, \ldots, s\}$ to be

$$D_i(x, C) = D(x) \cdot \begin{cases} p_0 + b/s + \epsilon & \text{, for } x \in R^+_{0,j} \\ p_0 - b/s - \epsilon & \text{, for } x \in R^-_{0,j} \\ p_0 + b & \text{, for } x \in R^+_{i,j} \\ p_0 - b & \text{, for } x \in R^-_{i,j} \\ p_0 & \text{, otherwise} \end{cases}$$

for $1 \leq j \leq k$. The positive subsets refer to the original rules, which thus have the desired properties stated earlier[1]. Any rule that covers more than one positive subset will inevitably also cover the negative counterparts. This is a consequence of the syntactical structure of $\mathcal{H}$, and of the fact that the bit vectors for positive subsets all have a Hamming-distance of at least two. The BIAS will be zero in this case. Specializing rules reduces the coverage without any increase in the BIAS.

The second part of the theorem can be proved similarly. Let the same subsets of $\mathcal{X}$ be associated to $R^+_{0,1} \ldots, R^-_{s,k}$ as before. The idea is to construct a pdf for which all rules have an identical BIAS, and to locally adjust the marginal distributions in order to achieve a similar situation as in the proof of the first case. To this end, let the *local* marginal densities $D'_i(x)$ for $1 \leq i \leq s$ be defined using the functions $D : \mathcal{X} \to \mathbb{R}^+$ above, which assign equal weight to all subsets, and which are uniformly distributed within each subset:

$$D'_i(x) = D(x) \cdot \begin{cases} 1 - \epsilon_m/3 & \text{, for } x \in R^{+/-}_{0,j} & \text{(global rule)} \\ 1 & \text{, for } x \in R^{+/-}_{i,j} & \text{(local rule for site } i\text{)} \\ 1 - \epsilon_m & \text{, for } x \in R^{+/-}_{p,j}, p \notin \{0, i\} & \text{(local rule, other site)} \\ 0 & \text{, otherwise} & \text{(unused subset)} \end{cases}$$

with $R^{+/-}_{(\cdot),j} := R^+_{(\cdot),j} \cup R^-_{(\cdot),j}$. The *local joint* density functions $D'_i : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^+$ can now be constructed for all sites $1 \leq i \leq s$ using *site-independent* factors:

$$D'_i(x, C) = D'_i(x) \cdot \begin{cases} p_0 + b, \text{if } x \in R^+_{i,j}, 1 \leq j \leq k \\ p_0 - b, \text{if } x \in R^-_{i,j}, 1 \leq j \leq k \\ p_0 \quad \text{, otherwise (BIAS} = 0) \end{cases}$$

[1] If $\log(s+1)$ or $\log(k)$ are no integers, then some subsets of $\mathcal{X}$ are not related to any rule. This has no effect on the validity, since these subsets receive no weight under any of the distributions.

All rules have the same BIAS b at all sites, and thus globally. The global COV values are

$$\begin{aligned} \mathrm{COV}_{D'}(R_{0,j} \to C) &= \frac{s(1-\epsilon_k/3)}{s} = 1 - \frac{\epsilon_k}{3} && \text{(global rules)} \\ \mathrm{COV}_{D'}(R_{i,j} \to C) &= \frac{1+(s-1)(1-\epsilon_k)}{s} \leq 1 - \frac{\epsilon_k}{2} && \text{(local rules)} \end{aligned}$$

As required, the "global rules" are ranked highest regarding the global pdf D'. At each local site i the corresponding "local rules" $R^+_{i,(\cdot)}$ have the highest COV regarding D'_i and are thus ranked highest. More general rules, subsuming several of the positive subsets of $\mathcal{X}$, will also cover the negative subsets, as discussed in the proof of the first part. Analogously, a specialization of rules leads to a reduced COV without increasing the BIAS. Choosing ϵ_k so that $\gamma = (1-\epsilon_k)^{-1}$ completes the proof. □

Theorem 10 implies that rules globally performing best are not necessarily among the k locally best rules at *any* site. Even for arbitrarily unskewed data, formalized in terms of definitions 42 and 43, the best rules collected from all sites, including the globally best rules, may be completely disjoint, in the sense that no example is covered twice. Please note that – unlike for homogeneously distributed data – this is not a problem of poor estimates. Theorem 10 applies to arbitrarily large sample sizes, and it covers the case of uniform distributions assumed in descriptive settings. The two parts of the proof illustrate that adjusting the marginal *or* the conditional densities "maliciously", sometimes to a very small degree, suffices to make the globally best rules look poor at all local sites. This implies that any distributed subgroup discovery procedure will have to estimate global densities *and* conditional probabilities of the target at the same time.

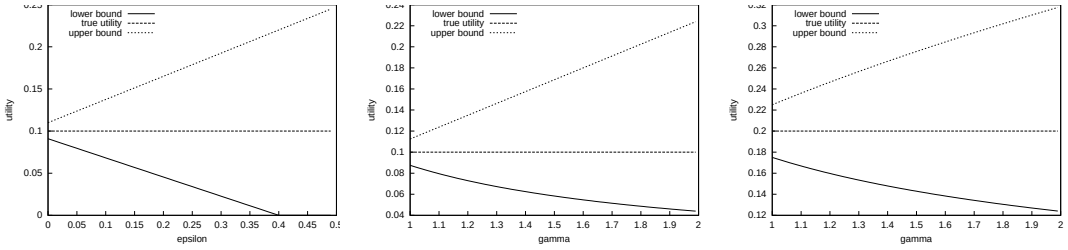
Although finding the globally best rules from local data is not possible in the worst case, finding approximately best rules might still be tractable. The following theorem gives a tight bound on the difference between locally and globally evaluated utilities, for simplicity assuming positive utilities and common class priors.

Theorem 11 *Let $D : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^+$ denote a global density function which is a weighted average of s local pdfs D_i, all sharing the same class priors. Considering a rule $A \to C \in \mathcal{H}$, let the marginal densities of D and a local density function D_i ($i \in \{1,\ldots,s\}$) be factor-similar up to γ for A, and let the joint density functions D and D_i be conditionally similar up to ϵ for the rule. Then the difference between global and local utilities of $Q^{(\alpha)}$ is bounded by*

$$\begin{aligned} &\max\left(0, \frac{Q^{(\alpha)}_{D_i}(A \to C)}{\gamma^\alpha} - \frac{\epsilon}{\gamma^\alpha}\mathrm{COV}_{D_i}(A \to C)^\alpha\right) \\ \leq &\max\left(0, Q^{(\alpha)}_{D}(A \to C)\right) \\ \leq &\max\left(0, \gamma^\alpha Q^{(\alpha)}_{D_i}(A \to C) + \epsilon\left[\gamma \mathrm{COV}_{D_i}(A \to C)\right]^\alpha\right) \end{aligned}$$

For valid choices of ϵ these bounds are tight in the general case.

Proof
A local marginal probability of an antecedent differs by at most a factor of $\gamma^{\pm 1}$ from the corresponding global probability. Similarly, the conditional probability differs by at most an additive



(a) $Q^{(1)}$ vs. ϵ for $\gamma \leq 1.1$ (b) $Q^{(1)}$ vs. γ for $\epsilon \leq 0.05$ (c) $Q^{(1/2)}$ vs. γ for $\epsilon \leq 0.05$

Figure 7.1.: Estimated global utilities with bounded uncertainty based on local utilities. The global COV is 0.25 and the global BIAS is 0.4. Marginal deviations are bounded by γ and deviations on class distributions are bounded by ϵ.

constant of $\pm\epsilon$. This implies

$$\begin{aligned}
&Q_D^{(\alpha)}(A \to C) = \text{COV}_D(A \to C)^\alpha \text{BIAS}_D(A \to C) \\
&\leq \gamma^\alpha \text{COV}_{D_i}(A \to C)^\alpha \cdot (\text{BIAS}_{D_i}(A \to C) + \epsilon) \\
&= \gamma^\alpha Q_{D_i}^{(\alpha)}(A \to C) + \epsilon\gamma^\alpha \text{COV}_{D_i}(A \to C)^\alpha
\end{aligned}$$

if all terms are positive. The lower bound is shown analogously.

Given that ϵ is chosen as a valid BIAS with respect to the prior of the target class, it is trivial to construct cases for which the bounds are tight. □

For distributed data approximately sharing a common underlying pdf, e.g., if $\gamma \leq 1.1$ and $\epsilon \leq 0.05$, the bounds are tight enough to allow for estimates with bounded uncertainty. This is illustrated in figure 7.1, showing the bounds for a rule with a global COV of 0.25 and a global BIAS of 0.4. For $\gamma \leq 1.1$ figure 7.1(a) shows upper and lower bounds for $Q^{(1)}$ with ϵ at the x-axis. Figure 7.1(b) and 7.1(c) depict bounds for different values of γ, assuming distributions that are conditionally similar up to an $\epsilon \leq 0.05$. Qualitatively, the curves for utility function $Q^{(1)}$ (WRACC, Fig. 7.1(b)) and $Q^{(1/2)}$ (binomial test function, Fig. 7.1(c)) are similar, but the latter is less sensitive to deviating marginal distributions.

Please note that theorem 11 allows to exploit different estimates for each antecedent $A \subseteq \mathcal{X}$ under consideration. Hence, the theorem is not restricted to learning tasks in which conditional or marginal distributions are known to be very similar. It also allows to collect rule-specific bounds from various sites. Possible sources of rule-dependent bounds on γ and ϵ range from background knowledge over density estimates to previously cached queries.

The following experiment illustrates how utilities can be estimated with bounded uncertainty. A synthetic data set was used, because this allows to control the different kinds of skew. A decision tree for a domain of 10 boolean attributes was constructed randomly. For each inner node the probability of the tested attribute being 1 was fixed at a value randomly sampled from $N(0.5, 0.25)$. The same was done for the distribution of the boolean target label at the leaves. For all examples, unspecified attributes were simply completed by drawing truth values uniformly. The examples were distributed to 5 sites by explicitly assigning a separate γ- and ϵ-skew to each leaf for each site. The skew-parameters were selected uniformly within the previously used intervals: $\gamma \in [0, 1.1]$, $\epsilon \in [0, 0.05]$. Based on this randomly constructed tree 10.000 examples were generated.

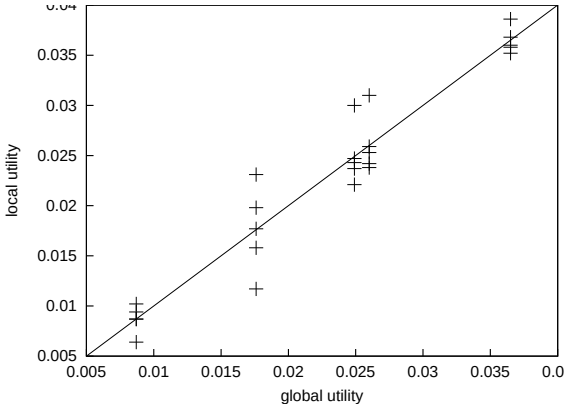


Figure 7.2.: Results for a synthetic data set containing 10.000 examples distributed to 5 local sites with controlled marginal and conditional skews. For the 5 globally best subgroups the global and local utilities are compared.

Subgroup	global $Q^{(1)}$	Lower bound	Upper bound
a5=1, a6=0, a8=0	0.0249	0.0231	0.0292
a5=1, a8=0	0.0260	0.0235	0.0316
a1=1, a5=1	0.0087	0.0083	0.0105
a5=1	0.0365	0.0191	0.0573
a1=1, a4=0, a5=0	0.0176	0.0164	0.0180

Table 7.1.: Corresponding utility bounds according to theorem 11.

The MIDOS algorithm, part of the KEPLER toolbox, was applied to the data, in order to select 5 best subgroups with respect to WRACC. Each dot in figure 7.2 compares the global utility of a rule (x-axis) to the corresponding local utilities at all sites (y-axis). Dots close to the diagonal represent similar utilities, which are useful for estimating the global utilities from local ones with bounded uncertainty. Table 7.1 lists the bounds that could be derived based on the local estimates, exploiting $\gamma \leq 1.1$ and $\epsilon \leq 0.05$. It is interesting to note, that only for the largest subgroup ($a5 = 1$, $\text{COV} \approx 0.35$) the bounds are not tight; for such large subgroups the utility can easily be estimated from samples, instead.

The question which rules do *not* allow to compute their utilities sufficiently well by techniques related to theorem 11 motivates a new extension of the learning task, discussed in the next section. It explicitly takes the locality of data into account.

7.5. Relative local subgroup mining

As motivated in the last section, inhomogeneously distributed data allows to define subgroups as subsets of local example sets[2] $\mathcal{E}_i$ that show distributions of the target attribute other than $\mathcal{E}$. This definition of subgroups has a natural interpretation that might be of practical interest in several domains. The corresponding rules could help to point out the characteristics of a single supermarket in contrast to the average supermarket, for example. For the specific case of distributed

[2]More precisely, these definitions refer to the weight of subsets with respect to D and D_i. These weights are of course estimated based on the example sets.

frequent itemset mining, an algorithm for mining *exceptional patterns* taking the locality of data into account has recently been presented by Zhang et al. (2004). Another unsupervised approach with a related aim, mining *high contrast frequent itemsets*, has been suggested by Otey et al. (2004). Based on entropy, it identifies itemsets with counts that are inhomogeneously distributed to the different sites. A corresponding extension to the task of subgroup discovery is lacking. The following function captures the idea of locally deviating rules.

Definition 44 *For* $r \in \mathcal{H}$ *the utility function* $\mathrm{RQ}_{D_i}^{(\alpha)}$ *at site* i *is defined as*

$$\mathrm{RQ}_{D_i}^{(\alpha)}(r) := \mathrm{Cov}_{D_i}(r)^{\alpha} \cdot (\mathrm{Bias}_{D_i}(r) - \mathrm{Bias}_{D}(r))$$

The rules maximizing this function are referred to as relative local subgroups.

Please note, that only the global *conditional* distribution is required in this context, since Cov is evaluated locally. Exploiting that Cov differs by at most a factor of γ, it is possible to restate theorem 11, again assuming common class priors.

Corollary 4 *For a given target class* C *let*

$$\begin{aligned} \mathrm{RQ}_{\max}^{(\alpha)} &:= \max\{\mathrm{RQ}_{D_i}^{(\alpha)}(r) \mid r \in \mathcal{H},\ r \textit{ predicts } C\} \textit{ and} \\ \mathrm{RQ}_{\min}^{(\alpha)} &:= \min\{\mathrm{RQ}_{D_i}^{(\alpha)}(r) \mid r \in \mathcal{H},\ r \textit{ predicts } C\} \end{aligned}$$

denote the maximal and minimal utilities of relative local subgroups. Then for all rules $r' \in \mathcal{H}$ *the difference between local and global utility is bounded by*

$$\gamma^{-\alpha} \cdot \left(\mathrm{Q}_{D_i}^{(\alpha)}(r') - \mathrm{RQ}_{\max}\right) \leq \mathrm{Q}_{D}^{(\alpha)}(r') \leq \gamma^{\alpha} \cdot \left(\mathrm{Q}_{D_i}^{(\alpha)}(r') - \mathrm{RQ}_{\min}\right)$$

if all terms are positive.

Corollary 4 allows to translate the utility of local subgroups into global scores with bounded uncertainty for any rule-dependent γ. The special case of a common marginal distribution is obtained by setting $\gamma = 1$.

Corollary 5 *For* $\gamma = 1$ *the three utility functions for local, relative local, and global subgroup discovery complete each other:*

$$\mathrm{Q}_{D}^{(\alpha)}(A \to C) = \mathrm{Q}_{D_i}^{(\alpha)}(A \to C) - \mathrm{RQ}_{D_i}^{(\alpha)}(A \to C)$$

Obviously, the tasks of discovering relative local subgroups and that of approximating the global conditional distribution are of similar complexity in this case. Corollary 5 shows how to detect global subgroups searching locally, given precise estimates of $\mathrm{RQ}_{D_i}^{(\alpha)}$, and how to compute $\mathrm{RQ}_{D_i}^{(\alpha)}$ from $\mathrm{Q}_{D}^{(\alpha)}$ for $\gamma = 1$.

7.6. Practical considerations

The theoretical results derived in the last sections mainly suggest that distributed subgroup discovery is a hard problem, because it is necessary to estimate both the marginal distribution of $\mathcal{X}$ and the conditional distribution of the label given $x \in \mathcal{X}$. The task of finding relative local subgroups still requires precise estimates of the conditional distribution.

This section relates the sub-tasks to known learning strategies. One can distinguish between three kinds of strategies, applying trained models, sampling with respect to the global distribution, and searching exhaustively. After discussing these issues, two algorithms following the last of these three approaches will be presented in the next section.

site	Y	A_1	A_2
1	-	0	0
1	+	0	1
2	+	1	0
2	-	1	1

Table 7.2.: An example for which distributed learning fails.

7.6.1. Model-based search

The idea of a model-based search is to first train a model that approximates the global conditional distribution of the target attribute. If the model yields precise estimates, then $\mathrm{RQ}_{D_i}^{(\alpha)}$ can directly be computed from the local data, which allows to discover the relative local subgroups in the next step. For a common marginal distribution of $\mathcal{X}$ ($\gamma = 1$) this also allows to discover the global subgroups by applying corollary 5. Without this assumption, bounded estimates for global rule utilities can be given (Cor. 4).

A simple learner that allows to approximate the conditional distribution is NAÏVEBAYES. It can easily be applied to distributed data, because the global model can be obtained by collecting the counts from all sites. A more complex technique that usually comes with higher accuracy is distributed boosting. Lazarevic and Obradovic (2002) proposed an algorithm similar to confidence-rated versions of ADABOOST (Schapire & Singer, 1999). In each iteration all learners train models based only on their local data. No examples are exchanged, but trained models together with their corresponding performance values. All learners have access to the same global model after each iteration. Weights are maintained locally, similarly to the case of a single example set. This allows to locally observe the importance of examples in the global learning context. Models are combined by choosing the prediction of highest confidence. For real-world data the authors report good results of their approach. It can be shown, however, that there are situations in which learning is not possible without exchanging examples. Consider the distributed learning problem shown in table 7.2. The target function is A_1 XOR A_2, but at site 1 A_1 is always 0 and at site 2 it is always 1. Hence, learner 1 will output $(A_2 \rightarrow +)$ and learner 2 $(A_2 \rightarrow -)$, without a chance to recover from this choice by means of reweighting examples.

Another problem with the model-based strategy is that even precise models usually do not obtain 100% accuracy. This means that some of the relative local subgroups may not be found, because it is unknown for which subsets the predictions of the model are poor. Hence, such strategies do not allow to give desirable guarantees.

7.6.2. Sampling from the global distribution

As discussed in section 7.1, it is often not possible to collect all the data at a single site. If the reasons are communication costs rather than privacy, then it may still be cheaper to learn directly with respect to the global distribution than to address a hard learning task using distributed approaches that do not come with any guarantees.

Applying the adaptive sampling techniques discussed in chapter 3, one can hope to find approximately best models with probabilistic guarantees after transferring just a small fraction of the data to a central node for the data mining step. By transferring data in appropriate proportions from all the sites, the model induction step can be performed on a sample drawn from the global distribution; the confidence bounds for utility functions discussed in subsection 3.3.2 apply, and the approximately k-best rules task can be solved as discussed before.

In the presence of prior knowledge or after identifying an approximately best model, this information can be broadcasted to all the local sites. Hence, it becomes possible to perform knowledge-based sampling, which means to sample from an altered global distribution in the sense of theorem 6 (p. 74). Since knowledge-based sampling constitutes a specific kind of rejection sampling it can be combined with cost-sensitive rejection sampling in a straight-forward manner. Due to the similarity between KBS and boosting, this procedure promises to yield models with high predictive performance. However, the number of examples that have to be transmitted when taking this approach is not clear in advance; it highly depends on the data set at hand and may become unreasonably large if many rules happen to perform about equally well. A thorough investigation of how to recognize situations in which this technique is tractable is still lacking and may be the subject of future work.

7.6.3. Searching exhaustively

The issues discussed in the last subsections, especially the fact that an approximation of the conditional distribution does not help to find the global subgroups reliably in the general case, are good reasons to tackle relative local and global subgroup discovery by means of exhaustively searching the hypothesis space.

Efficient distributed strategies exist for frequent itemset mining (Zaki, 1999); they basically exchange itemsets and counts. A straightforward extension of the Apriori algorithm is COUNT DISTRIBUTION (CD) (Agrawal & Shafer, 1996). At each round, every database generates all $i+1$ candidates from the globally large i-itemsets and broadcasts all counts to all other nodes. This procedure causes communication costs of $\Omega(|C|s^2)$, where $|C|$ is the number of candidates and s denotes the number of nodes (sites). One way to improve the CD algorithm is to use a designated node for each candidate that is responsible for polling and redistributing all counts of the candidate itemset. This method is part of the FDM algorithm (Cheung et al., 1996). It reduces the communication complexity of the algorithm to $\Theta(|C|s)$. Two additional pruning techniques are applied by FDM. Local pruning is based on the observation that, for an item to be frequent, it must be frequent at least at one node. Counts need to be exchanged only for such items. Second, nodes use an optimistic estimate for the support of an itemset based on partial counts (received from other nodes). Whenever this estimate is smaller than the minimal support, the candidate can be pruned. Schuster and Wolff (2001) present an improved algorithm for distributed association rule mining that is based on comparing local to global estimates of the support of an itemset. Counts are exchanged only in cases of conflicting local and global estimates of whether itemsets are frequent or not.

These ideas cannot directly be applied to the novel tasks studied in this chapter. Association rule mining differs significantly from subgroup mining in that only positive literals are supported, and in that all rules meeting a minimal support and confidence constraint are returned, not only the k best ones. Moreover, there will usually be many more frequent itemsets than subgroups, because the pruning performed during itemset mining does not exploit the specific characteristics of a chosen utility function. This will generally increase the number of candidates that have to be evaluated, and will hence increase the runtime complexity as well as communication costs. The idea of a polling site, as introduced by FDM, is very useful to avoid costly broadcasts, however, even in a subgroup discovery context.

The real power of the above approaches to distributed frequent itemset mining lies in their local pruning strategies. A straightforward idea would be to adapt this approach to distributed global subgroup mining. As shown before, this is not possible; globally optimal rules can simultaneously be inferior at each individual node, while pruning strategies applied to distributed

frequent itemset mining rely on the fact that globally frequent itemsets *must* be frequent at least at one node. This reflects that subgroup utility functions are lacking the monotonicity of rule support, a prerequisite for efficient itemset mining. This substantial difference between the tasks also hinders the application of more sophisticated pruning strategies for frequent itemset mining, e.g., the ones proposed by Otey et al. (2004) and Schuster and Wolff (2001).

Concerning relative subgroup mining, approaches based on association rule mining cannot be applied either; the score of a relative rule does not only depend on the local support, but also on two additional independent quantities, namely the local and global confidences of rules.

Applying the pruning strategy of MIDOS (Wrobel, 1997) allows to safely discard specializations of a rule with small COV, if these cannot contain improvements on the k best subgroups found so far. Additionally, since global counts generally need to be collected from all sites, more specific pruning techniques sometimes allow to stop the evaluation of a rule after receiving the counts of only a subset of all the sites. In the next section, two novel MIDOS-like algorithms for exhaustive distributed subgroup discovery are proposed, one for the global and one for the relative local task.

7.7. Distributed Algorithms

7.7.1. Distributed global subgroup discovery

This section presents an algorithm for distributed global subgroup mining, after introducing further definitions that ease notation. The absolute number of true positives of a rule r is denoted as $p(r)$, and the number of its false positives as $n(r)$. The argument is omitted if clear from the context. P and N denote the number of positives and negatives in the complete data set; the number of positives and negatives at site i are denoted as P_i and N_i, respectively.

Definition 45 *For any rule* $r : A \rightarrow C$ *the absolute number of covered positives and covered negatives* at node i *is denoted as*

$$p_i(r) := |\{A(x) \wedge C(y) \mid (x,y) \in E_i\}| \text{ and } n_i(r) := |\{A(x) \wedge \overline{C(y)} \mid (x,y) \in E_i\}|.$$

For simplicity, this section confines itself to subgroup discovery with the weighted relative accuracy metric. The proposed algorithm is based on count polling and distributed rule pruning. As shown in the last section, local pruning as used by distributed association rule mining is not sufficient in this setting. Therefore, another strategy based on optimistic estimates is analyzed.

A basic principle of the algorithm is that for each rule r, all refinements of this rule r' are created and counted at exactly one node. Hence, a refinement operator as defined in (Wrobel, 1997) can be applied. The following definition assumes a fixed total order on the set of attributes.

Definition 46 *A refinement operator* ρ *is a function that maps each rule to the set of its direct successors. A rule* $r' : A' \rightarrow C'$ *is a direct extension of* $r : A \rightarrow C$*, if and only if* $C = C'$ *and* $A' = A \cup \{X_i = v\}$ *for a variable* X_i *with the property that all attributes* X_j *in* A *have an index* j *which is strictly lower than* i*. The transitive relation* $r' < r$ *denotes, that* r' *is a refinement of* r*.*

In MIDOS, this operator is used in combination with the following pruning rule: The coverage of each rule r decreases monotonically with each refinement, so the upper-bound

$$\text{WRAcc}(r) \leq \text{Cov}(r) \cdot \left(1 - \frac{P}{P+N}\right) \tag{7.1}$$

allows to prune all refinements r' of a rule r if the coverage $\text{Cov}(r)$ is so low that r' cannot improve over the WRAcc of the k-best rule found so far.

This pruning method will now be adapted to support imbalanced information regarding counts requested from different sites. If for each node the counts for a rule r or a predecessor of r, denoted as r' are known, we can calculate a tight upper-bound on $\text{WRAcc}(r)$. If this maximal score is worse than the currently k-best rule, then the algorithm can safely prune rule r.

Lemma 6 *The (global) utility of a rule* r *is bounded by the following term*

$$\text{WRAcc}(r) \leq \frac{N}{(P+N)^2} \sum_{i=1}^{s} p_i(r'_i) \tag{7.2}$$

where $r'_i = r$ *or* $r < r'_i$. *This bound is tight for the most specific rules* $p_i(r'_i)$ *is known for.*

Proof
The correctness of this lemma follows from the fact that WRAcc orders rules according to the function $p - \frac{P}{N} \cdot n$. This can be seen when multiplying WRAcc with the constant term $(P+N)^2/N$:

$$\begin{aligned}
\frac{(P+N)^2}{N}\text{WRAcc}(r) &= \frac{(P+N)^2}{N} \cdot \frac{p(r)+n(r)}{P+N}\left(\frac{p(r)}{p(r)+n(r)} - \frac{P}{P+N}\right) \\
&= \frac{(P+N)}{N} \cdot \left(p(r) - p(r)\frac{P}{P+N} - n(r)\frac{P}{P+N}\right) \\
&= p(r) - \left(\frac{P}{N}n(r)\right) \leq p(r) = \sum_{i=1}^{s} p_i(r) \leq \sum_{i=1}^{s} p_i(r'_i) \qquad (7.3)
\end{aligned}$$

Optimal refinements "discard" no positives. If refinements are optimal at all sites i then all considered rules r'_i cover the same number of positives at site i as r. Furthermore, optimal refinements discard all negatives. This means that optimal refinements r are characterized by

$$p(r) = \sum_{i=1}^{s} p_i(r) = \sum_{i=1}^{s} p_i(r'_i) \quad \text{and} \quad n(r) = 0.$$

In this case, both inequalities in eqn. (7.3) are tight. □

The difference to eqn. (7.1) is that the coverage is replaced by the fraction of true positives $p(r)/|\mathcal{E}|$, a quantity which is strictly smaller than $\text{Cov}(r) = (p(r)+n(r))/|\mathcal{E}|$ unless r cannot further be improved by refinements, anyway. The pruning strategy exploits the fact that WRAcc increases monotonically if refinements discard only negatives. It is maximized by refinements that discard all negatives and no positives. For this reason straightforward adaptations of eqn. (7.1) apply to the broad class of utility functions sharing this property of monotonicity, e.g., to the binomial test function. It is sufficient to substitute the tightest known counts during optimistic score computation in lemma 6 for each rule, and to optimistically assume that a subsequent refinement is able to discard only the covered negatives.

The lemma can be used to prune rules, for which exact counts are available only from a subset of all nodes. If the upper bound for $\text{WRAcc}(r)$ is worse than the k-best rule, then r can be pruned directly without polling further counts. Lemma 6 allows to exploit a weak kind

of monotonicity: If a rule r' is pruned, then all refinements r of this rule can be pruned as well, as their optimistic scores are known to be no better than the optimistic score of r'. A rule $r : A \rightarrow C$ can hence be pruned (i) based on its optimistic score, or (ii) because it is *subsumed* by a previously pruned rule $r' : A' \rightarrow C'$, that is $C' = C$ and $A' \subset A$ (for the sets of literals), so $\{A(x) \mid x \in \mathcal{X}\} \subset \{A'(x) \mid x \in \mathcal{X}\}$. In the latter case we have $p_i(r) < p_i(r')$ at all sites, as the extension of r' is a superset of the extension of r, also allowing to apply lemma 6.

The novel algorithms for distributed subgroup mining scale linearly with the number of nodes. They make use of the discussed pruning strategies together with count polling. Each node i maintains three data structures. First, a list B_i containing the k (currently) best hypotheses. Second, a list of pruned hypotheses Z_i. These are rules for which it is known that no descendant can reach a score better than

$$b_i := \min_{r \in B_i} \text{WRAcc}(r),$$

the k-best score at node i. To this end, an optimistic upper-bound is computed using lemma 6. Finally, each node keeps a list of all rules it is polling counts for. This list is denoted as Q_i.

The distributed subgroup mining algorithm is initialized by assigning all rules with an empty body to an arbitrary node. The computation then follows the scheme shown in algorithm 7.

A node that receives an assignment for a rule r generates all canonical refinements (direct successors) $\rho(r)$ and serves as their polling node. For each refined rule r' the algorithm first obtains the local counts from the database and checks whether the rule can be pruned. If a rule is pruned based on its optimistic score, the node additionally informs all other nodes about this step of pruning. In contrast, subsumption-based pruning of a rule r' does not require to broadcast r', because each node is known to also have a rule subsuming r' in its list of pruned rules Z_i. If a rule is not pruned, the node broadcasts a query for counts on r and adds r to the list of open hypotheses Q_i. The individual nodes then reply their local counts for r'. As more and more local counts arrive, the bound on the global count gets tighter.

If all local counts for a rule r are available and r cannot be pruned, it is first evaluated if r is better than b_i. If this is the case, r is inserted into B_i as described above and broadcasted to all other nodes. Then, the rule is assigned to a node that is responsible for generating and counting the canonical refinements of this rule. Besides the rule itself, the local counts from all nodes for rule r are transmitted. This information is necessary to allow for pruning that is based on partially available counts, as described above. Each rule is assigned to the node with the highest local coverage. The rationale behind this choice is that this node is the most likely one to be able to prune the rule without querying other nodes for counts.

Algorithm 7 shows distributed global subgroup mining at node j. M_j denotes the input message queue of node j. best_{ij}, prune_{ij}, count_{ij}, query_{ij} and assign_{ij} are messages, where i denotes the sender and j the receiver. The procedures that are part of algorithm 7 are executed as long as messages arrive.

The algorithm has communication cost that are bounded by $O(|C|s)$, and thus scales linearly with the number of nodes s. This can be seen easily, considering all messages that are exchanged *per candidate*: a query for counts, its replies, and possibly a broadcast for a new best hypothesis or for pruning. These messages contain only rules and individual counts. Additionally, at most one delegation message for each rule is produced, containing a set of local counts. This message is of size $O(s)$. The former kind of messages are assumed to have a constant length and are sent to all s nodes; the latter kind has a length of $O(s)$, but is sent only once. So, in both cases the costs for a specific kind of message per candidate is in $O(s)$. Since the number of messages per candidate is bounded by a constant, the asymptotic bound of $O(s)$ still holds after aggregating all costs for a candidate.

Algorithm 7 Distributed Global Subgroup Mining (at node j)

```
// Update best rules
for best_ij(r, WRAcc(r)) ∈ M_j do
  if WRAcc(r) > b_j then
    insert(B_j,r);
  end if
end for

// Update pruned rules
for prune_ij(r) ∈ M_j do
  Z_j = Z_j ∪ {r};
end for

// Obtain message counts
for count_ij(r, n_i(r), p_i(r)) ∈ M_j do
  recalculate optscore(r);
  if prunable(r) then
    Z_j = Z_j ∪ {r};
  else
    if counts-complete(r) then
      if WRAcc(r) > b_j then
        best.insert(B_j, r);
        bcast(best(r, WRAcc(r)));
      end if
      Q_j = Q_j \ {r};
      a = argmax_i(n_i(r) + p_i(r));
      send(assign_ja(r, {(p_1(r), ...)}));
    end if
  end if
end for

// Handle assignment to refine a rule
for assign_ij(r, {(p_1(r), ...)}) ∈ M_j do
  for r' ∈ ρ(r) do
    recalculate optscore(r');
    if not(prunable(r')) then
      bcast(query(r'));
      Q_j = Q_j ∪ {r'};
    end if
  end for
end for

// Answer requests for local counts
for query_ij(r) ∈ M_j do
  send(count_ji(r, n_j(r), p_j(r)));
end for

prunable(r):
  if (∃r' ∈ Z_j) : r' subsumes r then
    return true;
  end if
  if optscore(r) < b_j then
    bcast(prune(r));
    return true;
  end if
  return false;
```

It is interesting to note that if communication costs may be ignored for this task, then the algorithm performs the same search as MIDOS, but distributed evaluation reduces the time required at each local site for each rule. Aggregating counts is cheap, so the total runtime (ignoring communication delays) even *benefits* from distributed data, in this case. The required time for computing rule counts for a data set of size n can be assumed to be in $\Theta(n \log n)$. This means, that even the aggregated local computation times of this distributed algorithm will usually be lower than those of a global discovery algorithm operating on a single database. In practice, communication costs highly depend on the specific database and network architecture. They may range from microseconds to seconds, and usually cannot be ignored. A serious evaluation of the dependencies between communication costs and runtime complexity needs to consider different distributed architectures, which is out of the scope of this thesis.

There is a straightforward way to combine distributed global subgroup discovery with the knowledge-based sampling algorithm for sequential subgroup discovery (KBS-SD, algorithm 2, p. 89); all that is required is (i) a globally consistent reweighting strategy, and (ii) to compute and transmit *weighted* rule counts. We may apply the same global reweighting strategy

$$D_{t+1}(e) := D_t(e) \cdot (\text{LIFT}_{D_t}(r_t, e))^{-1}$$

for each example $e \in \mathcal{E}$ in each iteration t as KBS-SD, simply by using the global model (e.g., set of rules $\{r_1, \ldots, r_t\}$), which is known at all sites, in combination with the *global* LIFTs when reweighting *locally* at each site. Algorithm 7 selects the same rules from distributed data as if the data was not distributed, and using the global performances of rules, KBS-SD also reweights the data exactly as if working on a single global data set. This implies that if subgroups are discovered sequentially, if the data is locally reweighted, and if weighted counts are transmitted, then the result is a distributed KBS-SD algorithm that yields exactly the same results as when processing all the data after transferring it to a central site.

A further aspect worth noting is that the proposed algorithm applies to multi-relational data as well, because it applies the same search techniques as the multi-relational MIDOS algorithm. It suffices to use the same multi-relational refinement operator as proposed by Wrobel (1997).

7.7.2. Distributed relative local subgroup discovery

Compared to definition 44 (p. 158), the algorithm proposed in this section addresses a slightly simplified relative local subgroup discovery task. It aims at the identification of rules maximizing the following evaluation metric:

Definition 47 *The* relative local utility *of a rule* r *at node* i *is defined as*

$$\mathrm{RLU}_i(r) := \mathrm{COV}_i(r) \cdot (\mathrm{BIAS}_i(r) - \mathrm{BIAS}(r) + c_i), \textit{ with } c_i := \frac{P_i}{P_i + N_i} - \frac{P}{P + N}.$$

This is, because different class skews P_i/N_i are of minor interest for this task. The additional term c_i is used to focus on deviations of globally and locally differing conditional class distributions for subsets covered by considered rules. This turns the term in brackets into deviations of local from global confidences, as motivated above. A more convenient version of the RLU metric is

$$\begin{aligned}\mathrm{RLU}_i(r) &= \mathrm{COV}_i(r)\left(\frac{p_i(r)}{p_i(r)+n_i(r)} - \frac{p(r)}{p(r)+n(r)}\right)\\ &= |E_i|^{-1} \cdot \left(p_i(r) - \underbrace{p(r)\cdot\frac{p_i(r)+n_i(r)}{p(r)+n(r)}}_{=:\hat{p}_i(r)}\right) = \frac{p_i(r)-\hat{p}_i(r)}{|E_i|}.\end{aligned}$$

The term $\hat{p}_i(r)$ can be interpreted as the estimated number of positives within the subset covered by rule r at site i. This estimate is based on the fraction of positives in the subset of the *global* data that are covered by the rule, i.e. on the global confidence. A factor-equivalent metric to RLU is

$$\mathrm{RLU}^*_i(r) := p_i(r) - \hat{p}_i(r). \tag{7.4}$$

Finding relative subgroups differs from finding global subgroups in that each node finds an own, individual set of rules. The score of a rule is defined with respect to its local coverage and its relative bias. While the coverage of a rule r can easily be computed locally for each database, computation of the bias requires to obtain global counts for r. The global counts of a rule can be calculated as described in the last section. There is one important difference, however. Rules can only be pruned, if they are pruned at *every* node. In the next paragraphs, algorithm 7 is adapted to the task of relative local subgroup mining. The variant is also based on count polling and optimistic pruning. The following tight optimistic pruning rule holds for the task of relative local subgroup mining when using the RLU metric.

Lemma 7 *For relative local subgroup discovery, rules* r *with* $p_i(r)$ *positives,* $n_i(r)$ *negatives, and* $\hat{p}_i(r)$ *estimated positives covered by rule* r *at site i,*

$$RLU_i(r') \leq \frac{p_i(r) - \max(0, \hat{p}_i(r) - n_i(r))}{|E_i|},$$

is a tight upper-bound for the local utilities of all rules $r' < r$.

Proof
Considering the factor-equivalent metric RLU^* (see eqn. (7.4)) it is easily seen that an optimal refinement of rule r reduces $\hat{p}_i(r)$ by covering less examples that are "predicted" positive, while not reducing $p_i(r)$. If the $n_i(r)$ negative examples covered by r are predicted positive by $\hat{p}_i(r)$, and if a refinement r' of r exists, that covers only the $p_i(r)$ positive examples then we reach at a utility of

$$RLU_i^*(r') = p_i(r) - \max(0, \hat{p}_i(r) - n_i(r)).$$

This cannot be improved any further by refinements, since r' covers only positives, and any further refinement reduces $p_i(r)$ at least as much as $\hat{p}_i(r) - n_i(r)$. Since $RLU^* = RLU \cdot |E_i|$ this proves the lemma. □

The algorithm for relative subgroup mining works as follows: Again, each node has a list of best rules, pruned rules, and open rules. In addition, nodes maintain a rule cache used to store the global counts of rules for which a node serves as the polling node. The mapping of rules to responsible nodes is realized in terms of a hash function.

Each node starts with an empty set of rule candidates. It then generates first-level rules that are evaluated locally. If a rule r can be pruned based on lemma 7 it is discarded. Otherwise, the node requests global counts $p(r)$ and $n(r)$ for r from a polling node that is determined by calculating a hash value for the rule. The node that receives this request checks whether it finds the rule in its cache. If so, it directly returns the corresponding global counts. Otherwise, the node first queries all other nodes for their corresponding local counts. After aggregating all local counts $p_i(r)$ and $n_i(r)$ the polling node stores and returns the global counts. Given the global counts and the local counts for a rule r, the exact utility score of r can be computed. If r is better than the k^{th} best rule, then it is inserted into B_i, as described in the last section. If r – and thus all of its refinements – receive an optimistic score that is worse than the lowest score in B_i, then r is pruned. Neither best rules nor pruned rules are broadcasted, as they are not relevant to other nodes.

The approach scales linearly with the number of nodes, although the pruning strategies for relative local subgroup mining are weaker than those proposed for distributed global subgroup mining. Thus, communication costs for relative subgroup mining are bounded by $O(|C|s)$, where $|C|$ are the candidates considered by at least one node. Relative subgroup mining for all nodes is usually more expensive than global subgroup mining, because each rule may be relevant, unless it is pruned at all nodes simultaneously.

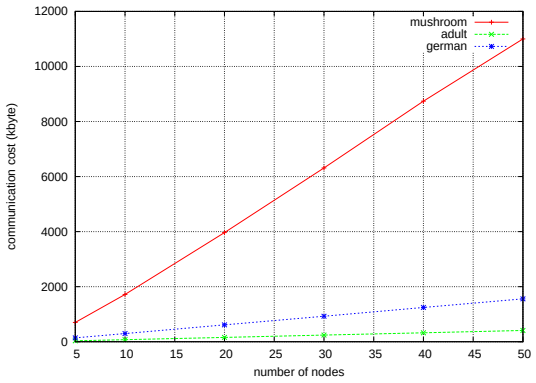


Figure 7.3.: Communication costs for distributed global subgroup mining

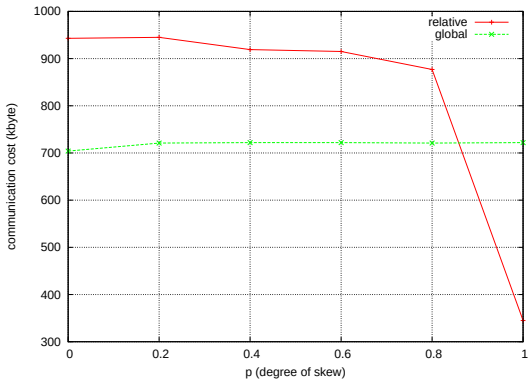


Figure 7.4.: Data skew / communication costs for global and local algorithm

7.8. Experiments

This section empirically evaluates the properties of the algorithms presented above. As both algorithms are guaranteed to find the best rules, evaluation is only concerned with communication costs, the focus of section 7.7.

Costs are evaluated on three data sets taken from the UCI library, Mushrooms, Adult, and German. For Adult and German, numerical attributes were discretized using minimal entropy partitioning (Fayyad & Irani, 1993).

First of all, the substantial difference between the tasks of subgroup and association rule mining is illustrated exemplarily. Association rule and frequent itemset mining rely on a user-provided support threshold, and are usually applied to find huge amounts of rules. Subgroup discovery finds only the k best rules with respect to a user-specified utility function, not requiring a threshold. Even if the best rule utility was known to a frequent itemset mining algorithm in advance, it would be more costly to generate all itemsets based on a corresponding support threshold in a distributed setting than to run distributed subgroup discovery; state of the art algorithms for distributed frequent itemset mining evaluate at least all frequent itemsets at all nodes. For example, the German data set contains more than 50.000 frequent itemsets, when using the support-based pruning threshold of the MIDOS algorithm (see eqn. (7.1)) in combination with the (usually unknown) utility of the best subgroup. In contrast, the global subgroup discovery algorithm evaluates less than 3.000 *candidates*.

Still, the communication costs of the algorithm grow no more than linearly with the number of nodes. This property was validated in a first experiment, measuring costs by accounting 4 bytes for each rule transmitted over the network and 2 bytes for each count. To be able to measure the impact of data skews in the distribution of data to individual nodes, the following procedure was used. First, the data was clustered using an EM algorithm. The number of clusters was chosen equal to the number of nodes. A parameter p_{skew} denotes the probability that the clustering determines the node the example is assigned to. Otherwise examples are assigned randomly to a node with uniform probabilities. For $p_{skew} = 1$, each node receives all data points in its corresponding cluster. For $p_{skew} = 0$, all examples are distributed randomly. This allows to adjust the data skew between both extremes. The results for the data sets using $p_{skew} = 0$ and finding one global rule ($k = 1$) for rules of constrained length as in MIDOS (searching for best rules containing up to 3 literals) are shown in Fig. 7.3. For all three data sets the curves confirm

the theoretical findings concerning the scalability of the proposed methods. Please note, that in this experiment each database contains about the same amount of data, the worst case for this method.

The second experiment compares the communication costs for distributed global and relative local subgroup mining for varying degrees of skew. The results of mining the most interesting rule of length up to 3 literals for the Mushrooms data set is shown in Fig. 7.4 for a network of $s = 5$ nodes. It can be seen that the data skew has a low impact when mining distributed global subgroups. For relative subgroup mining the situation is different. Given a low skew, the cost for finding relative subgroups increases. The reason is that relative subgroups can only be found if the data distribution among nodes deviates. For low skews only rules with very low scores can be identified, which however forces all nodes to search a very large search space as pruning cannot be applied. Reaching at a certain level of skew the distributions deviate sufficiently to identify corresponding rules, leading to a sharp decrease of costs in Fig. 7.4 for relative local subgroup mining.

7.9. Summary

Major companies usually do not store all of their data in a single database, but generally have a huge number of smaller databases that may be geographically distributed. Privacy issues and data volumes hinder the collection of all the data at a single site. Hence, there is a growing interest in distributed approaches to data mining that minimize communication costs while giving guarantees at the same time that the same or similar results are reported as when data mining from all the data that is available globally. Since the data are generally not distributed at random, but tend to reflect semantically meaningful partitionings, e.g., sales grouped by store or state, approaches that take the locality of data into account allow to define novel data mining tasks that benefit from data being stored in a distributed fashion.

In this chapter a broad class of rule evaluation metrics has been adapted to distributed learning. All theoretical findings can easily be adapted to more general forms of supervised learning. First it was shown that the utility measures common in the literature on subgroup discovery apply to homogeneously distributed data in the same way as for non-distributed data. If the different sites do not share a single underlying distribution that generated the data, however, then even precise estimates may yield completely disjoint rule sets at all sites, none of which contains a single one of the best k rules. For the general case, a tight bound for the differences between global and local rule utilities was derived, which allows to translate local rule utilities into global ones with bounded uncertainty. For the task of discovering rules that have a higher local than global utility it was shown that it is at least as hard as approximating the global conditional distribution of the target attribute. For a common marginal distribution each problem can be solved locally, given a solution for the other.

The results indicate that distributed subgroup discovery is a hard problem, since it requires precise estimates of both, the global marginal (e.g., $\Pr(x)$) and the global conditional distribution (e.g., $\Pr(y \mid x)$). The former may for example be obtained by distributed variants of frequent itemset mining, the latter by means of distributed boosting. In practice, the bounds required to translate local into global rule utilities will usually be unknown, so the negative results presented in the first part of this chapter have a higher practical relevance than the positive results. As discussed, there are good reasons to tackle the problem by exhaustively searching the hypothesis space, applying specific pruning strategies whenever possible. The MIDOS pruning strategy based on optimistic scores has been improved and adapted to the distributed setting.

Moreover, a tight pruning threshold for relative local subgroup discovery was derived, and count polling as known from frequent itemset mining was adapted to the new tasks. Based on these ideas, two algorithms for distributed subgroup discovery were proposed that both guarantee to deliver optimal results at communication costs linear in the number of nodes and rule candidates. This is an essential property for scalable distributed algorithms. The global distributed subgroup discovery algorithm evaluates the same candidates as a corresponding exhaustive non-distributed algorithm. As a result, a combination with knowledge-based sampling yields the same weights or selection probabilities for each examples as a non-distributed algorithm after each iteration. With query costs that scale super-linearly with the number of examples at each node – as common when using index structures – the total computational costs of distributed subgroup discovery are even lower than an exhaustive search from non-distributed data.

Possible directions for future work contain evaluations of probabilistic strategies to distributed subgroup mining. If e.g., transferring data is legal but expensive, adaptive sampling can be combined with knowledge-based sampling. Transferring patterns that play the role of compressed example sets is also possible in this setting. The goal would be to reduce the communication costs further, while allowing for probabilistic guarantees regarding the quality of discovered rule sets.

A condensed version of sections 7.1 to 7.6.2 of this chapter has been published as (Scholz, 2005d). Sections 7.6.3 to 7.8 are joint work with Michael Wurst (Wurst & Scholz, 2006).

8. Support for Data Preprocessing

8.1. The KDD process

In the last years, a consensus among researchers was achieved that knowledge discovery in databases is not just a linear process of selecting data, applying a data mining algorithm, and reporting or deploying its results, but a complex and *iterative* process that, if done right, comprises a large number of phases. Similar refinements of process models could be observed in software engineering; early process models like the waterfall model (Royce, 1970) were organized as a single linear top-down approach from informal to formal specifications, followed by phases of implementation, testing, and maintenance. This view has changed drastically, giving rise to iterative models like the unified process (Jacobson et al., 1999). In each phase new insights into the structure of the problem at hand can be gained. This may reveal unjustified crucial assumptions made at earlier stages, which might require to step back and adapt specifications. In turn, some insights that can only be gained at a later phase will affect design decisions of earlier phases substantially, so it is often a wise choice not to rely on early specifications or project plans, but to keep the process flexible, stepping forth and back between the phases. For instance crucial parts of a system can first be implemented in a rapid prototyping manner, leaving room for iterative refinements of the overall project plan.

KDD projects have a number of specific properties, that are mostly rooted in the data-driven, analytical nature of the discipline. A well-known definition that tries to capture the general goals of KDD has been proposed by Fayyad et al. (1996):

Definition 48 *Knowledge discovery from databases (KDD) is the nontrivial process of identifying valid, previously unknown, potentially useful patterns in data.*

Process models for KDD aim to structure KDD projects in terms of typical subtasks and their interdependencies. The next paragraphs discuss the best known KDD process model, the CRoss-Industry Standard Process for Data Mining (CRISP-DM) by Chapman et al. (2000). First it should be noted that CRISP-DM uses the term *data mining* as a synonym for knowledge discovery in databases. In contrast, as in most of the literature on KDD, this thesis uses the term data mining only for the step in the process that addresses model induction or pattern extraction, the setting of required parameters for the applied machine learning algorithms, and maybe post-processing of models or of resulting sets of patterns.

The CRISP-DM model has a strong industrial background. It aims to provide a step-by-step guide to practical knowledge discovery applications, including several remarks on when to consider budget and other resource constraints, and it focuses also on steps like understanding the underlying business goals.

The model distinguishes between six different phases, depicted in figure 8.1, business understanding, data understanding, data preparation, modeling, evaluation, and deployment. Each of these phases depends on the results of preceding phases, but the CRISP-DM model considers the phases to provide a flexible framework for iterative refinements and cyclic iterations, as motivated above. The phases can roughly be characterized as follows.

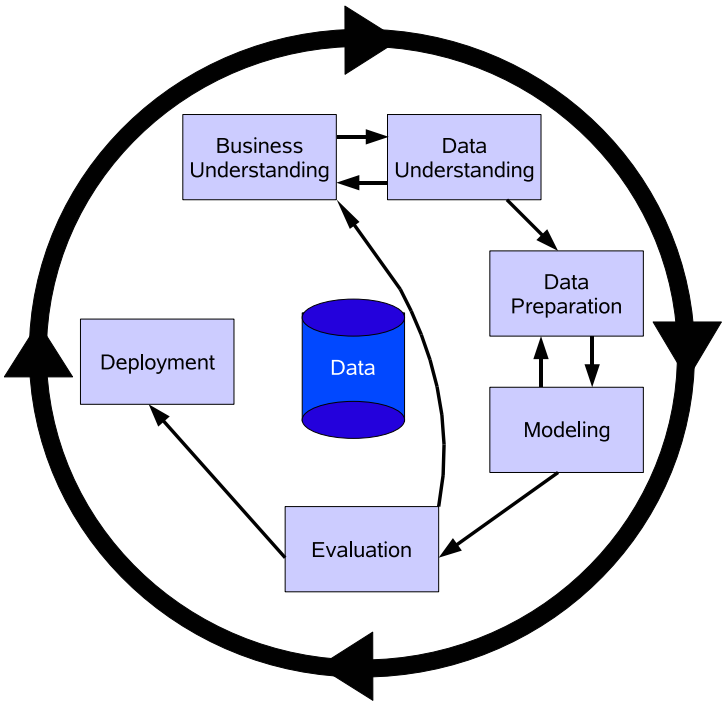


Figure 8.1.: An overview of the six different phases part of the iterative CRoss-Industry Standard Process for Data Mining (CRISP-DM). Arcs connecting phases highlight directions into which KDD processes typical continue.

Business understanding The first phase of the model aims to provide a solid understanding of the problems to be tackled and the constraints to be met from a decision maker's point of view. One aspect of this phase is to discuss and clarify all constraints of the data mining project, e.g. the budget, the availability of domain experts, of technical equipment, and of the data to be analyzed. Another aspect is to agree on or clarify the relevant concepts in the business domain. Sometimes it will be useful to set up a dictionary for the domain terminology. Next, the business goals should be clarified precisely and it should be agreed on criteria for measuring success. The business goals directly address business problems, e.g. to reduce the number of churning customers of a telecommunication company. It should be clarified to which target group the KDD project results are presented, or who is supposed to use the results, and in which way. Risks to the success of a project should be identified in advance, and alternative project plans should be prepared for cases in which these risks occur. Such risks range from delays in the schedules over costs to compromised results of the KDD project. A detailed evaluation of costs and potential benefits helps to gain a clearer picture of whether it is reasonable to carry out the project at all, and if so, what might be appropriate stopping criteria.

The next step is to translate the business goals of the project into formal data mining problems that can directly be addressed by data mining experts. If the business goal is to reduce churn, then a data mining goal might be to identify a maximum likelihood model that allows to predict whether any specific customer will churn within the next 6 months. Analogously, as for the business goals, success criteria should be identified for the data mining goals.

The final steps of the business understanding phase are to set up a detailed project plan containing dependencies, risks, milestones, etc., to discuss it with the involved personnel, and to assess the equipment and tools to be used during the project.

Data understanding After the goals have been formulated, the next step is to inspect the available data. Often the data will come from different sources, which requires to convert and join them before being able to process them with appropriate analysis tools. The first crucial question is which tables, attributes, and tuples to include. Some of the relevant information may not be available from the beginning, which motivates to spend some efforts on completing the data sets, while many of the tuples and attributes may not be relevant at all, and should hence be removed from consideration. The data formats may vary between different tables or flat files and may sometimes not be well-suited to be analyzed directly. Free text entries, for example, should be transformed into a structured format, if possible, and formats should be standardized throughout all tables.

The main objective of the data understanding phase is to collect various kinds of information about the data. This covers aspects like the availability of useful information, the number of tables and attributes to be mined, and e.g., the number of tuples per table. Another important aspect is the (initial) format of the data, e.g., whether attributes are nominal or continuous, can be structured in terms of a taxonomy, and whether attributes are deemed relevant from a business perspective. This high-level description should be augmented by a first statistical analysis that describes attribute ranges, the number of duplicates, missing values, and correlations between different attributes and tables. Initial hypotheses can be formulated or refined based on first, very simple data mining approaches in this phase. Typically, only visualization and querying techniques will be applied at this point.

A final crucial part of this phase is the assessment of the data quality. Among the most common problems to be identified and reported are incomplete tuples, incorrect values, and violations of referential integrity constraints. Missing values may be represented in different formats, as constants, like “9999”, as empty fields (“NULL” entries in relational databases), or both. The semantics of missing values may differ substantially between data sets, and should hence be clarified. To be able to figure out the impact of missing values for the analysis task, their frequency should be estimated for each relevant data source. Free-text entries may contain variations and typos, which may easily lead to different representations of the same attribute value. More generally, several nominal values may have similar semantics or may even encode synonyms, which should be captured by choosing a more appropriate representation. In turn, the same attribute value might have been interpreted and used differently by different users, leading to another kind of inconsistency. Plausibility checks might reveal different kinds of incorrectly entered data. Especially if tables from different sources are joined it is likely that some entries will be redundant or inconsistent. Flat files are a data format most prone to quality problems. Missing delimiters require to reconstruct the original mapping from values to attributes, for example. Sometimes the values may not fit the required format of the corresponding attribute.

The result of this phase is a detailed report, describing the basic characteristics of the collected data, encountered data quality issues, and results of a first statistical evaluation. Plots are useful to visualize individual distributions, correlations, and inconsistencies found in the data.

Data preparation The data collected and documented in the preceding phase is usually still not in an appropriate form for the analysis step. The data preparation phase hence aims to address all of the problems reported in the last phase. Furthermore, it covers the transformation into a representation that allows for successful data mining.

The first step is to reconsider the selection criteria for tables, attributes and tuples in the light of the previous results. For example, some attributes may be removed from consideration, because the statistical evaluation revealed that they are irrelevant for the task at hand. Technical constraints may suggest to confine to subsamples of the data if the available data volume may not

be processed in reasonable time with the selected tools. Sometimes the integration of additional data sources, e.g. publicly available demographic data, may be considered at this point.

The identified data quality problems are addressed by data cleansing techniques. Missing values are avoided by selecting only clean subsets, or by substituting the blank fields, using default values, or values predicted by machine learning techniques. Recognized kinds of noise should be removed, if possible, or at least be documented.

The representation of the data has a large impact on the quality of data mining results. The notion of *feature construction* subsumes various techniques that construct new attributes (features) based on existing attributes. This allows to organize the relevant information in a way that better meets the demands of data mining techniques. For example, ages could be computed as the difference between the current date and date of birth. If attributes are known to be of different relevance, specific attribute weights may be introduced. Further ways to transform representations contain the normalization of attributes, turning continuous into discrete values by discretization techniques, or capturing implicit orders over nominal attributes by mapping those attributes to a numerical range.

If the preprocessed data is split to multiple tables and the subsequently applied technique is not able to handle this representation, a step of merging and aggregating the tables is required. Merging tables is also reasonable, if several tables contain similar kinds of information for the same objects. A final step addresses tool-specific demands regarding the data format. For example, a tool could assume the first attribute to be the target attribute in a supervised learning scenario, or require a specific flat file format.

Modeling The model building (or data mining) step takes as input the data prepared in the previous phase. The choice of data mining techniques is narrowed by the data mining goals and the available tools as specified in the business understanding phase. It should be made sure that any underlying assumptions of data mining techniques, like “no missing values”, “only numerical values”, or “normally distributed attributes”, are met by the data produced during the preparation phase.

The model building step itself is discussed in subsequent chapters of this thesis. Clearly, the quality of models should be evaluated using a sound test design, for example cross-validation for predictive tasks. Initial parameter settings should be justified and documented. This aspect is of diminishing importance, however, since parameters are nowadays often optimized in larger learning loops as e.g. supported by the YALE learning environment (Mierswa et al., 2006).

The resulting models may be improved by post-processing steps, and the assessment of their utilities may be eased by annotating performances. Large sets of association rules may for example be reduced to a most relevant subset of rules, each of which is typically annotated by support and confidence scores. Depending on the kind of models or extracted patterns different kinds of additional documentation may help to interpret results and point out potential weaknesses at later points. A subsequent assessment phase compares different models in terms of the chosen evaluation criteria, and tries to figure out their potential impact, their novelty and usefulness. As far as possible the results should also be evaluated with respect to the business problems, and be commented on by domain experts. The modeling step may have to be repeated several times due to additional insights gained during the procedure.

Evaluation The CRISP-DM model contains a separate evaluation phase for data mining results, going beyond the evaluation foreseen as part of the modeling phase. *All* the results are evaluated, which include the models produced in the previous phase, but also any additional findings. The rationale is that, apart from the models, during steps of e.g. data inspection and

exploration unexpected results might have been observed, that are not related to the initial business goals, but might nevertheless provide useful insights into the domain. The evaluation of models has a stronger focus on the initial business goals in this phase. In particular, experts aim to interpret the results in terms of the application at hand. The results are improved models or short-comings to be addressed in further iterations, but also additional business questions to be addressed in further data mining projects, e.g. in response to additional findings. A critical review of all steps performed during the KDD project, including failures and possible improvements, is also foreseen. Finally, it has to be decided whether the business goals have been met sufficiently well, how to refine the project plan with the remaining resources, and whether to move on to the deployment phase or to step back to one of the previous phases.

Deployment The goal of the final phase is to transfer all the results gained during the KDD project into the daily business procedures. This starts with an analysis of the deployable models and other findings. A plan is set up, how the results may be used and by whom, how to monitor the benefits gained by this step, and which problems may occur. Similar to software development, the maintenance of the deployed result deserves attention if it is supposed to become part of the daily business. Monitoring helps to identify situations in which the results are no longer appropriate. Setting up strategies for updating the applied knowledge at low costs in dynamic environments, or at least for identifying criteria that determine when to stop using the results are hence reasonable. One strategy that helps to derive such criteria is to document and reconsider the business problems initially addressed by the project.

Finally, CRISP-DM foresees a final report and presentation, as common for most IT projects. The report should document the full process and the experiences made, in order to provide a useful guideline for subsequently planned similar projects, the costs incurred at each step, deviations from the initial plan, and recommendations for future work. A separate project review might document more of the details, e.g., pitfalls or experiences made by individual members of the team.

The notion of a *data mining context* as defined by the CRISP-DM model is a description of KDD projects in four dimensions, namely the application domain, the data mining problem type, the technical aspects, and the tools and techniques. Each KDD project moves along all of these four dimensions, and may be similar to other projects with respect to any of these aspects. As pointed out for the deployment phase, one of the goals of CRISP-DM is to preserve the experiences made during a project, in order to ease subsequent applications that are similar with respect to some of these aspects. This objective is shared by the MININGMART system, which will be illustrated in the next sections.

8.2. The MiningMart approach

The MININGMART system has been developed as part of a European research project[1]. The author of this thesis has worked on this project in the period from September 2001 to March 2003, the official end of the project. He subsequently provided system maintenance and integrated novel aspects until the end of 2003. The remainder of this chapter provides a high-level introduction to the concepts realized by the MININGMART system; in chapter 9 the main contributions of the author of this thesis are discussed in detail.

The MININGMART system aims to ease preprocessing and algorithm selection in order to turn KDD into a *high-level query language* for accessing *real-world databases*. As a first benefit

[1]The MININGMART project was supported by the European Union under contract IST-1999-11993.

for practical applications, it provides a pre-defined library of the most import preprocessing operators. These operators perform data transformations such as, e.g., discretization, replacing null values, aggregation of attributes, and transforming time-stamped data into sequences of events. As an important aspect for KDD practitioners, all these preprocessing operators directly access relational databases and are capable of handling large volumes of data.

MININGMART relies on meta-data descriptions of KDD applications, which can easily be set up via a graphical user interface. The system contains a specific meta-data compiler that translates these meta-data specifications into executable SQL code. This section starts with a description of the meta-data model (subsection 8.2.1). Subsequently it is illustrated, how the MININGMART system supports users in setting up preprocessing cases (subsection 8.2.2 and 8.2.3). The role of the compiler is briefly discussed in subsection 8.2.4, and more elaborately in the next chapter. The different levels of abstractions integrated into the system, and the resulting support for case adaptation motivate the collection of successful solutions from the past in a public Internet case-base. A brief discussion of these issues is provided in subsection 8.2.5.

8.2.1. The Meta-Model of Meta-Data M4

Meta-data or ontologies have been a key to success in several areas. In the scope of MINING-MART, the advantages of meta-data driven software generation are:

Abstraction: Meta-data are given at different levels of abstraction, a conceptual (abstract) and a relational (executable) level. This makes an abstract case understandable and re-usable.

Data documentation: All attributes together with the database tables and views, which are input to a preprocessing chain are explicitly listed at both, the conceptual and relational part of the meta-data level. An ontology allows to organize all data, e.g. by distinguishing between concepts of the domain and relationships between these concepts. For all entities involved, there is a text field for documentation. This makes the data much more understandable, e.g. by human domain experts, than if just referring to the names of specific database objects. Furthermore, statistics and important features for data mining (e.g., presence of null values) are accessible as well. This augments the meta-data usually found in relational databases and gives a good overview of the data sets at hand.

Case documentation: The chain of preprocessing operators is documented, as well. First of all, the declarative definition of an executable case in the M4 model can already be considered to provide a documentation. Furthermore, apart from the opportunity to use "speaking names" for steps and data objects, there are text fields to document all steps of a case together with their parameter settings. This helps to quickly figure out the relevance of each step and makes cases reproducible.

Ease of case adaptation: In order to run a given sequence of operators on a new database, only the relational meta-data and their mapping to the conceptual meta-data has to be defined. A sales prediction case can, for instance, be applied to different kinds of shops, and a standard sequence of steps for preparing time series for a specific learner might even serve as a template that applies to very different mining contexts. The same effect eases the maintenance of cases, when the database schema changes over time. The user just needs to update the corresponding links from the conceptual to the relational level. This is especially easy when all abstract M4 entities are documented.

The MININGMART project has developed a model for meta-data together with its compiler, and has implemented human-computer interfaces that allow database managers and case designers

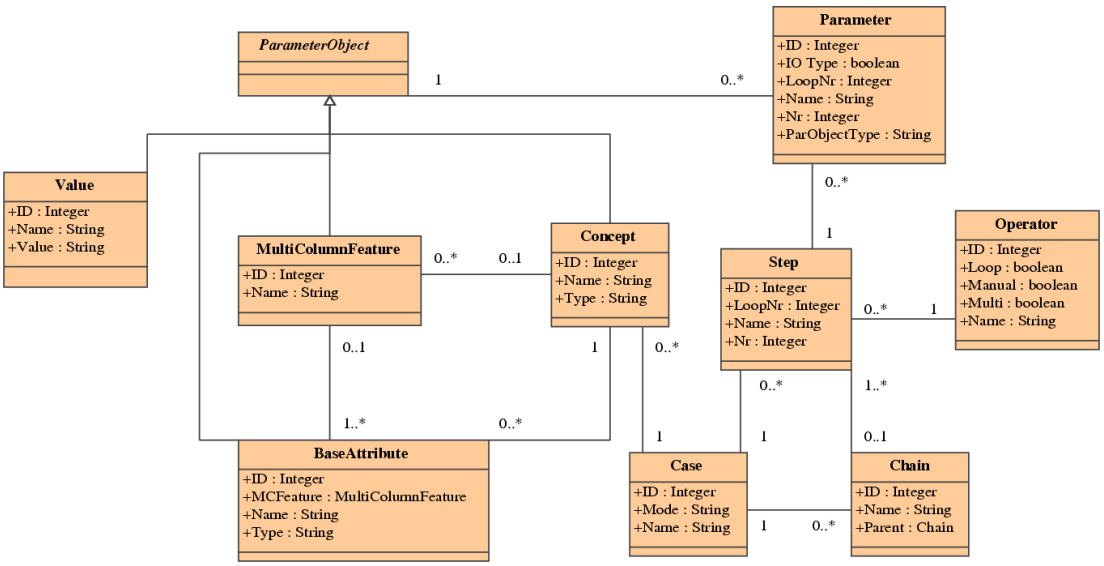


Figure 8.2.: Simplified UML diagram of the MININGMART Meta Model (M4)

to fill in their application-specific meta-data. The system supports preprocessing and can be used stand-alone or in combination with a toolbox for the data mining step.

This section gives an overview of how a case is represented at the meta-level, how it is practically applied to a database, and which steps need to be performed when developing a new case, or when adapting a given one.

The form in which meta-data are to be written is specified in the meta-model of meta-data, M4. It is structured along two dimensions, topic and abstraction. The *topic* is either the data or the case. The data consist of observations to be analyzed. The Case[2] is a sequence of (preprocessing) Steps. The *abstraction* is either conceptual or relational. The conceptual level is expected to be the same for various applications, while the relational level actually refers to the particular database at hand. The conceptual data model describes concepts like Customer and Product and relationships between them like Buys. The relational data model describes the business data that are analyzed. Most often it already exists in the database system in the form of the database schema. The meta-data, written in the form as specified by M4, are stored in a relational database themselves.

Figure 8.2 shows a simplified UML diagram of the conceptual level of the M4 model. Each Case contains Steps, each of which embeds an Operator and is linked to a set of corresponding parameters. Apart from Values, parameters may be Concepts, BaseAttributes, or MultiColumnFeatures (features aggregating multiple BaseAttributes). This part is a subset of the conceptual part of M4. The relational part contains Columnsets and Columns. Columnsets refer to database tables, to database views, or virtual (meta-data only) views. Each Columnset consists of a set of Columns, each of which refers to a database attribute. Likewise, Columns are the relational counterpart to BaseAttributes. For Columns and BaseAttributes there is a predefined set of data types, also omitted in figure 8.2. Section 9.2 discusses the M4 model in more detail.

[2]For clarity, terms starting with a capital letter are used in this and the following chapter when referring to concepts of the MININGMART meta model (M4), e.g., “Case”, “Step”, and “Parameter”.

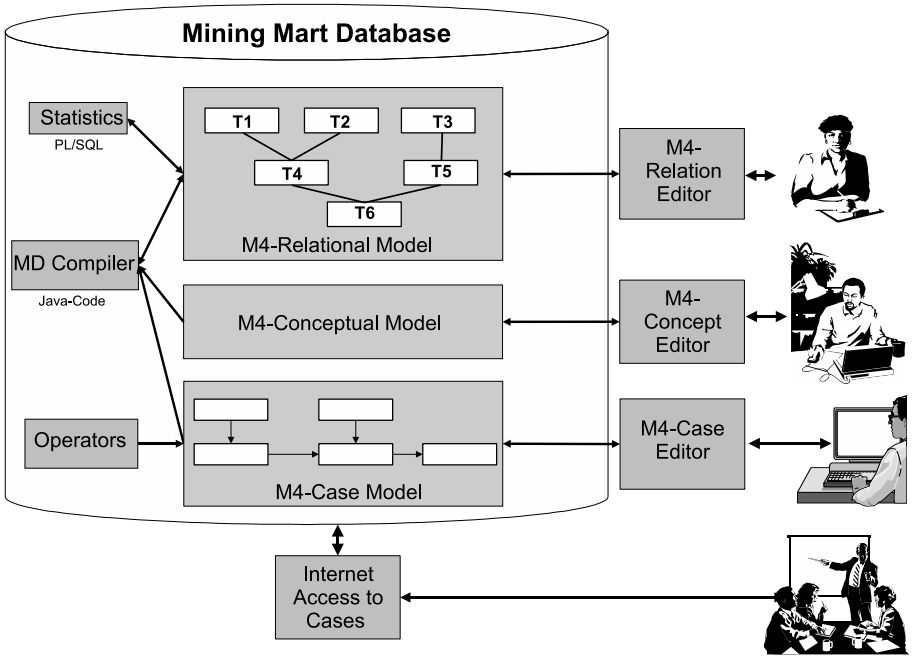


Figure 8.3.: Overview of the MININGMART system and its user groups

8.2.2. Editing the conceptual data model

As depicted in figure 8.3, there are different kinds of experts working at different ends of a knowledge discovery process. First of all, a domain expert defines an ontology that names and relates all the entities relevant to the data mining application at hand. Ontologies are also referred to as conceptual data models, because they are commonly used to model conceptual domain knowledge. The main building blocks supported by M4 to set up an ontology are *Concepts* having *Features*, and *Relationships* between these Concepts. Typical example Concepts for many business domains are Customer and Product. Each Concept consists of a set of Features, each of which, in turn, is either a single BaseAttribute or a MultiColumnFeature. A BaseAttribute simply corresponds to a database attribute, e.g., the name of a customer. A MultiColumnFeature, in contrast, aggregates a fixed set of BaseAttributes. This kind of Feature should be used, when semantically related information is split over multiple BaseAttributes. If the amounts and currencies of bank transfers are represented by two separate BaseAttributes, for example, then it is possible to define a single MultiColumnFeature that contains both BaseAttributes, and hence all of the related information. Another example are the separate date and time-of-day attributes common in some DBMSs. These can be aggregated to a single point-in-time information.

Relationships are connections between Concepts. There could be a Relationship named Buys between the Concepts Customer and Product, for example. At the database level one-to-many Relationships are represented by foreign key references, many-to-many Relationships make use of cross tables. However, these details are hidden from the user at the abstract conceptual level.

To organize Concepts and Relationships, the M4 model offers the opportunity to use inheritance. Modeling the domain in this fashion, the concept Customer could have *Subconcepts* like Private Customer and Business Customer. A Subconcept inherits all Features of its Superconcept. As an example of an inheritance between Relationships we may derive a Subrelationship Installment purchase from the general Relationship Buys. Please note, that in

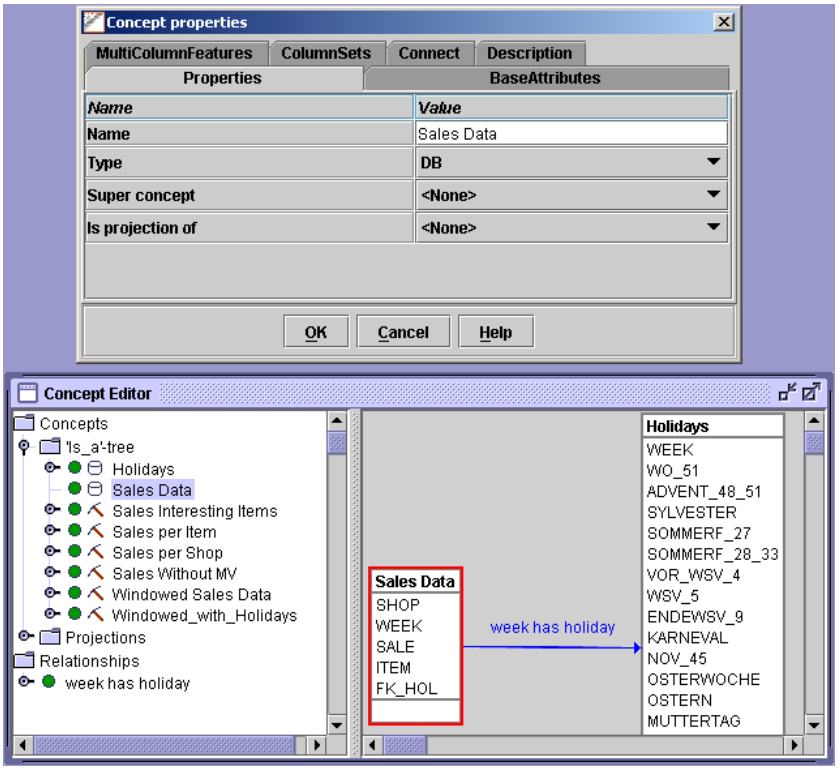


Figure 8.4.: Screenshot of the Concept Editor while editing the Concept “Sales Data”

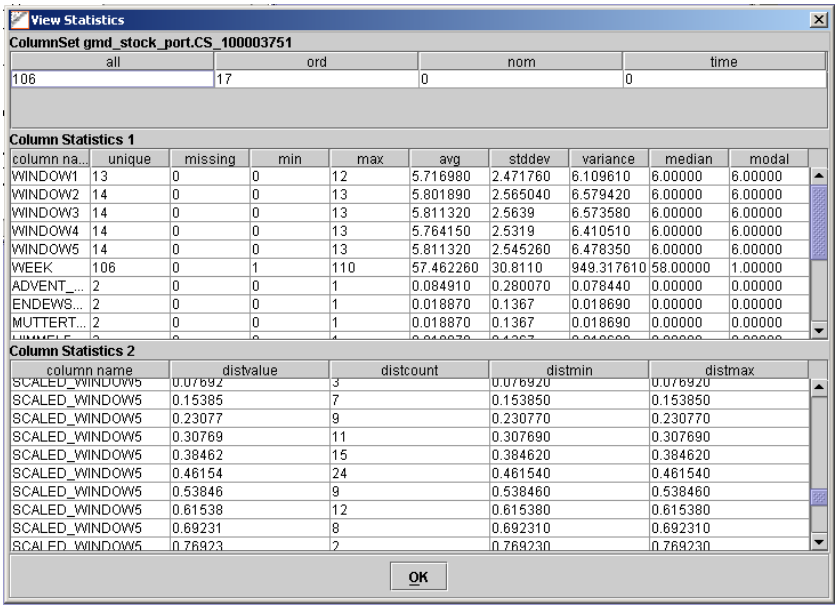


Figure 8.5.: Statistics of a database view, including the frequency of attribute types and several aggregates and distributional information for each individual attribute.

subsequent releases of MININGMART the set of supported relations between Concepts has been re-designed (Euler, To appear).

Figure 8.4 shows a screenshot[3] of the *concept editor*, a module of the MININGMART system that supports the user in setting up and maintaining conceptual models. In the figure it is used to list and edit BaseAttributes. The right part of the lower window indicates that the selected Concept Sales Data is connected to another Concept Holidays by a Relationship week has holiday.

8.2.3. Editing the relational model

Given a conceptual data model, a database administrator maps the involved entities to the corresponding database objects. The relational data model of M4 is capable of representing all the relevant properties of a relational database. The most simple mapping from the conceptual to the relational level is given, if concepts directly correspond to database tables or views. A Concept Customer, containing the features Customer ID, Name, Address is mapped to a database table CUST_T, for example, which contains a matching attribute for each BaseAttribute of the Concept, e.g., CUST_ID, CUST_NAME and CUST_ADDR. In this case, it suffices to specify the table name and the attribute mapping.

A more complex kind of mapping is required if the information on names and addresses is distributed over different tables. There may for example be a common key attribute CUSTOM_ID that allows to relate tuples across the different tables, so the mapping requires a join operation. Please note, that the much simpler situation above can always be reached manually by inspecting the database and creating a view that corresponds to the Concept under consideration. However, there should be an adequate support by the MININGMART system for mapping the conceptual to the relational meta-data, because this is a crucial prerequisite for adapting best-practice cases to new domains.

There is a line of research that aims at finding corresponding entities in similar database schemas. Rahm and Bernstein (2001) provide a survey on so-called *schema matching* approaches. Wagner (2005) describes a first module that integrates schema-matching into the MININGMART concept editor. It adapts ideas from Do and Rahm (2002) to the conceptual vs. relational meta-data framework. Moreover, this module supports users in creating views for Concepts in situations sketched in the example above, namely that the matching database attributes of a Concept's BaseAttribute are spread over different tables or views.

As a further feature supporting the exploration of database entities, there is a data viewer embedded into the concept editor. This tool displays database tables, but is also capable of displaying simple statistics of connected tables and views. This also allows to inspect intermediate and final results produced by the M4 compiler (see subsection 8.2.4). Figure 8.5 shows an example of the statistics displayed. For each view or table, the number of tuples and the numbers of nominal, ordinal, and time attributes are computed. For numerical attributes, the number of different and missing values are displayed, and the minimum, maximum, average, median, and modal values are computed, augmented by standard deviations and variances. Only the applicable ones of these functions are applied to ordinal and time attributes. Finally, information on the distribution of values is shown for all kinds of attributes.

[3] All screenshots in this chapter were taken from the official release at the end of the project in 2003.

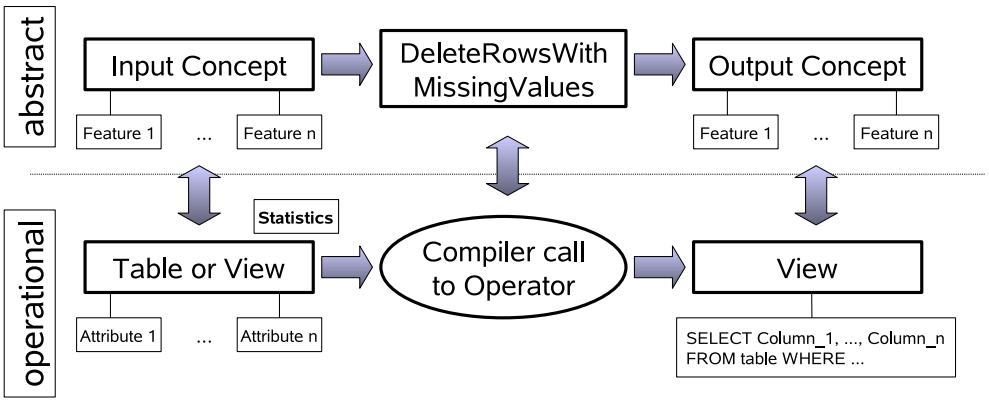


Figure 8.6.: Illustration of the coupling between abstract conceptual and operational level.

8.2.4. The Case and its compiler

All the information on conceptual descriptions and the corresponding database objects involved are represented within the M4 model and stored within relational tables. M4 *Cases* denote a collection of Steps, basically performed sequentially, each of which creates or extends one Concept. Each Step is related to exactly one M4 *operator*, and holds all of its input arguments. The M4 compiler reads the specifications of Steps and executes the corresponding operator, passing all the necessary inputs to it. This process requires the compiler to translate the conceptual entities, like input Concepts of a Step, to the corresponding relational entities, like database table names, the name of a view, or the SQL definition of a virtual view, which is only defined as relational meta-data in the M4 model.

One can distinguish between two kinds of operators, manual and machine learning operators. Manual operators just read the M4 meta-data of their input and add an SQL-definition to the meta-data, which establishes a virtual table. Currently, the MININGMART system offers 41 manual operators for selecting rows, selecting columns, handling time data, and generating new columns for the purposes of e.g., handling null values, discretization, moving windows over time series, or gathering information concerning an individual (e.g., customer, patient, shop).

External machine learning operators, on the other hand, are employed following a wrapper approach. Currently, the MININGMART system offers learning of decision trees, k-means, support vector machines, association rule mining, and subgroup discovery as learning operators. These learners are either used as preprocessing operations, or they are applied in the classical way, as data mining operators. The necessary business data are read from the relational database tables, converted into the required format, and passed to the algorithm. After execution the result is read by the wrapper, parsed, and either stored as an SQL-function, or materialized as additional business data.

In any case, the M4 meta-data will have to be updated by the compiler. A complex machine learning tool to replace missing values is an example for operators that alter the business data. In contrast, for operators like a join it is sufficient to *virtually* add the resulting view together with its corresponding SQL-statement to the meta-data.

Figure 8.6 illustrates how the abstract and the executable (or relational) level interact. Initially, just the upper sequence is given. It consists of an input Concept, a Step, and an output Concept. The definition of the Concept contains a set of embedded Features; the Step contains an operator together with its parameter settings. Apart from operator-specific parameters, the

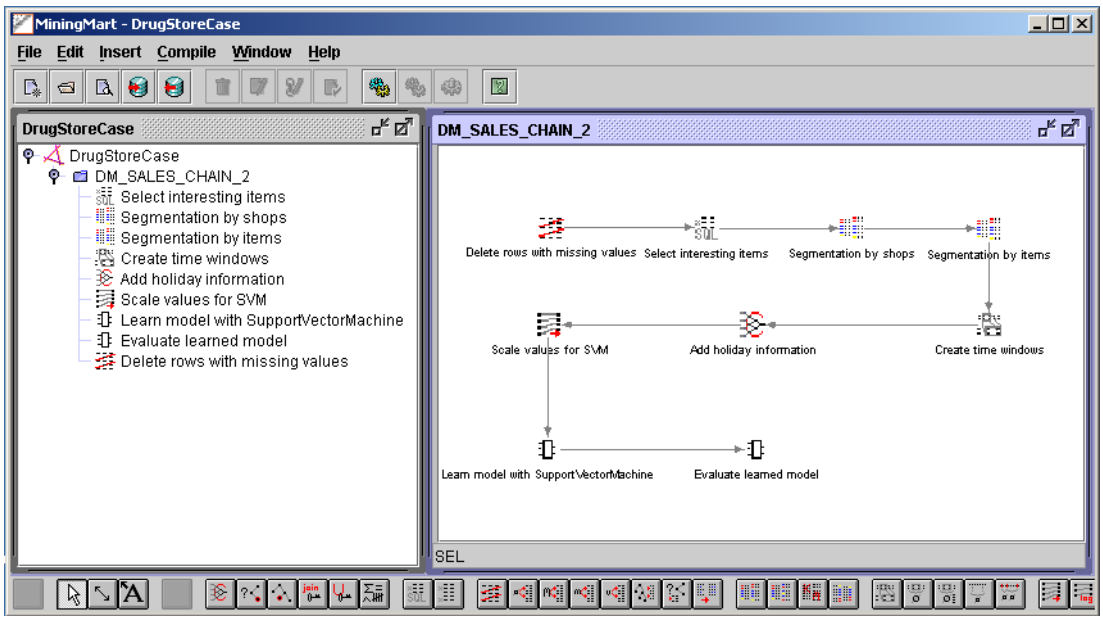


Figure 8.7.: A small example Case in the case editor. The Concepts are listed on the left, the Steps and their interdependencies are shown on the right.

input and output Concept are parameters of the Step, as well. The compiler needs the inputs, e.g., the input Concept and its Features, to be mapped to relational objects before execution. The mapping may either be defined manually, using the concept editor, or it may be a result of executing a preceding Step. If there is a corresponding relational database object for each input, then the compiler executes the embedded operator. In the example this is a simple operator named DELETEROWSWITHMISSINGVALUES. The corresponding executable part of this operator generates a view definition in the database and in the relational meta-data of M4. The latter is connected to the conceptual level, so that after the execution there is a mapping from the output Concept to a view definition. The generated views may be used as inputs to subsequent Steps in MININGMART, but they may as well be accessed by other tools, for example by main memory data mining toolboxes like YALE. The meta-data compiler, its library of operators, and dependencies to other MININGMART modules will be discussed in detail in chapter 9.

The task of a case designer, ideally a data mining expert, is to find sequences of Steps resulting in a representation well suited for the given data mining task. This work is supported by the *case editor* tool which is a module of the MININGMART system. Figure 8.7 shows a screenshot of a rather small example Case while edited by this tool. Typically a preprocessing chain consists of many different Steps, usually organized as a directed acyclic graph, as opposed to the linear sequence structure depicted in figure 8.7. To support the case designer, a list of available operators and their categories, e.g., feature construction, clustering, or sampling is part of the conceptual case model M4. The idea is to support a fixed set of powerful preprocessing operators, in order to offer a comfortable way of setting up Cases on the one hand, and in order to ensure re-usability of Cases on the other. Several useful operators that could be identified over time have been implemented and added to the repository.

For each Step, the case designer chooses an applicable operator from the collection, and sets all of its parameters (i.e. assigns the input Concepts, input BaseAttributes and/or input Relationships and specifies the output). To ease the process of editing Cases, applicability constraints on

the basis of meta-data are provided as formalized knowledge, and are automatically checked by the human computer interface (see section 8.2.2). That way, only valid sequences of Steps can be produced by a case designer. Furthermore, the case editor supports the user by automatically creating output Concepts of Steps according to certain meta-data constraints, and by offering property windows tailored to the properties of chosen operators.

A sequence of many Steps, a *Chain* in M4 terminology, transforms the original database into another representation. Each Step, as well as the dependencies between different Steps, are formalized in M4, so the system automatically keeps track of the performed activities. This enables the user to interactively edit and replay a Chain or parts of it. A full application is referred to as a *Case* in MININGMART.

As soon as an efficient preprocessing Case has been found, it can easily be exported and added to an Internet repository of MININGMART *best-practice cases*. Only the conceptual meta-data is submitted, so even if a Case involves sensitive information, as given for most medical or business applications, it is still possible to distribute the valuable meta-data for re-use, while hiding all the sensitive data and even the local database schema.

8.2.5. The case-base

One of the major advantages of the MININGMART system is the opportunity to publish successful applications on the Internet, and to re-use best-practice cases published by other users. The shared knowledge allows all users to benefit from new Cases. Submitting a new Case of best practice is a safe advertisement for KDD specialists or service providers, since the relational data model is not disclosed. The structured information of each Case may be utilized to support users in finding the Case that is most similar to their own application. To this end, the project set up a web interface that visualizes the conceptual meta-data. It is possible to navigate through the case-base and to inspect single Steps, see which operators were used and with which kinds of Concepts. The web interface directly accesses the Case data from M4 meta-data tables in the database. It thereby avoids redundancies as well as any additional technical efforts. Figure 8.8 shows a screenshot of a Case's business level description. In addition to the data that is explicitly represented in M4, a business level has been added. This level aims to relate the Case to business goals and to give several kinds of additional descriptions, e.g., which success criteria were chosen. For example, the sales prediction answers the question "How many sales of a particular item do I have to expect?". The business goal is to avoid situations in which requested items are sold out, while minimizing the stock at the same time. A particular constraint of this application that reflects expected supply times is that sales forecasts are only useful if made at least four weeks ahead. Especially the more informal descriptions should help decision makers to find a case tailored towards their specific domain and problem. The additional information is stored in an XML-representation that is directly connected to M4 entities. On the Internet these connections are reflected by hyperlinks. Figure 8.9 shows the ontology of the business layer.

It is possible to start the search for a Case at each category of the business level or of the conceptual level. In this sense, Cases are indexed by all the categories that are part of the conceptual M4 model and the business model. If a user considers a Case useful, then its conceptual data can be downloaded from the server. The downloadable Case itself is a category in an XML framework. The locally installed MININGMART system offers an import facility for installing the meta-data into the user's M4 tables. If problems arise or further help is required, the business level holds a category with contact information of the case designer or a company that provides support.

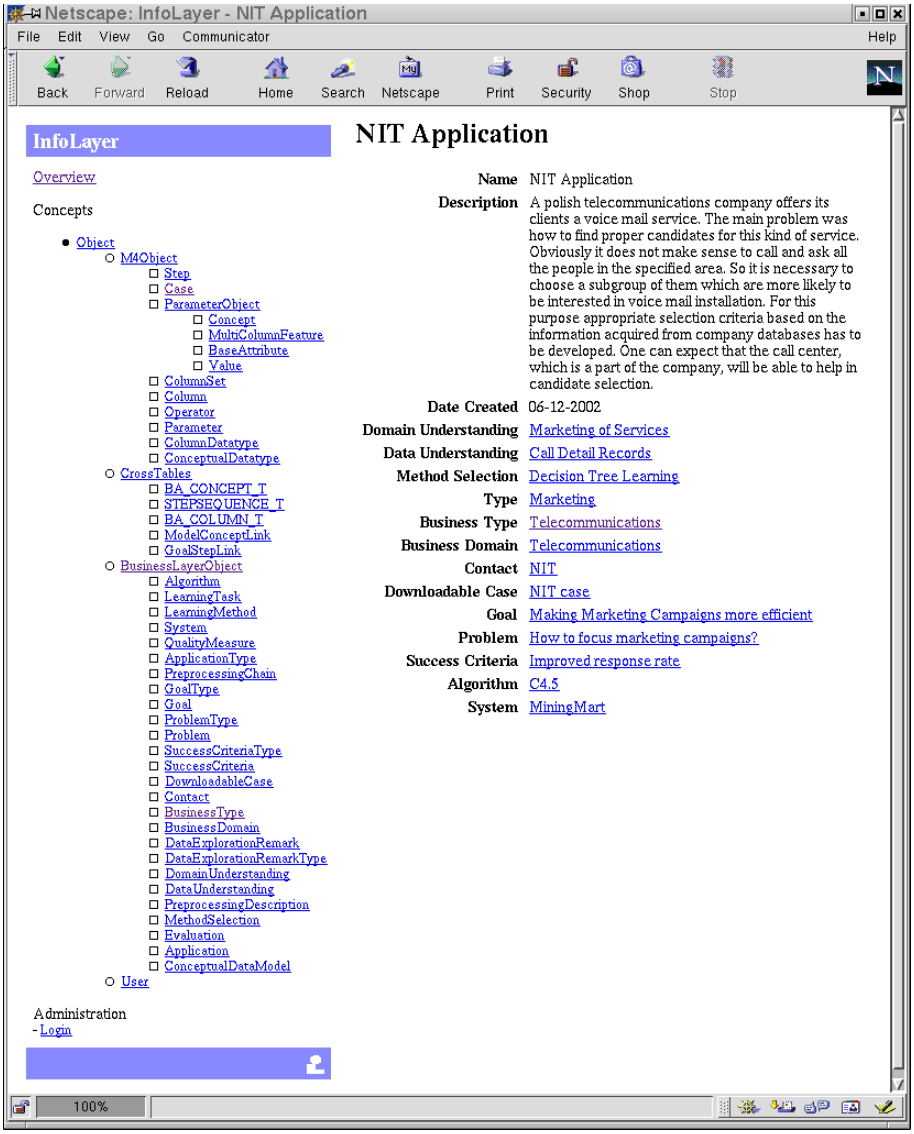


Figure 8.8.: The web interface to the case base visualizes all Cases, their Steps, embedded operators, and parameters in HTML format. Entities related in the M4 schema are connected by hyperlinks. Additionally, a business level is part of the interface. It describes the available Cases in terms of e.g., the addressed business goals of the data analysis task. After choosing a case based on conceptual M4 and business layer descriptions, the user can simply download it. The Case adaption facilities of the MININGMART system helps to quickly adjust the case to the user's environment.

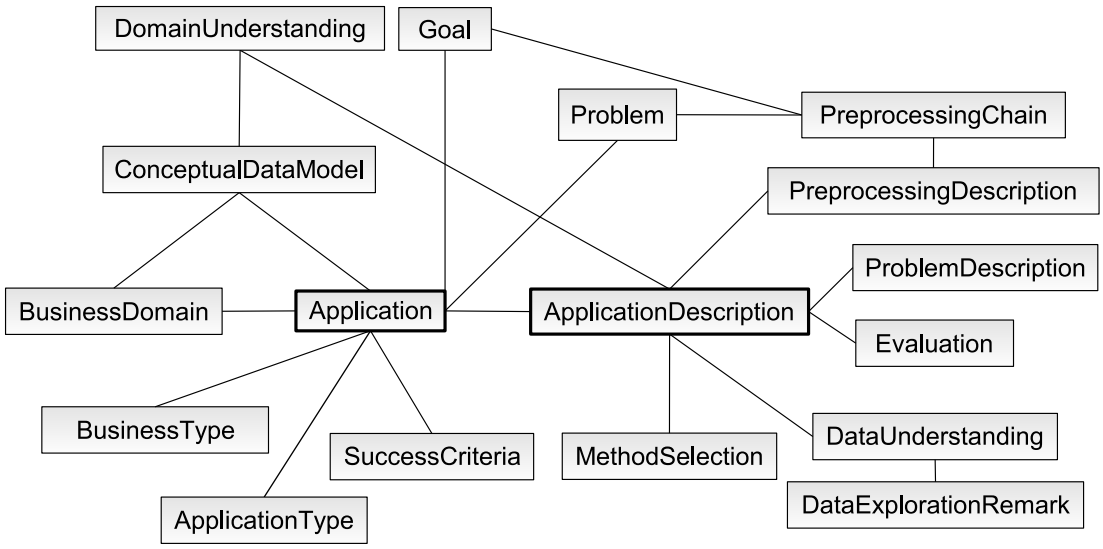


Figure 8.9.: The ontology of the business layer, used to describe M4 Cases in business terms. Lines represent 1:n or m:n references.

As of this writing, the case base has a focus on Customer-Relationship Management. It contains the following 5 cases:

Call Center Case: an analysis of call center and contract data for marketing in the telecommunication domain (Chudzian et al., 2003)

Life Insurance Churn Prediction: a case describing the analysis of insurance data for direct mailing (Kietz et al., 2000b; Kietz et al., 2000a)

Churn Prediction Case: churn prediction application of a telecommunication service provider (Richeldi & Perrucci, 2002)

Model Case Telecom: an instructive template for churn prediction cases (Euler, To appear)

DrugStoreCase: a full analysis of sales data for sales prediction, including the data mining step and its validation (Rüping, 1999)

The fact that MININGMART was neither open source nor free of charge for commercial applications turned out to be problematic for collecting best-practice cases in the past. A first open source version of MININGMART has been released on 12.4.2006. However, a remarkable number of alternatives to the presented system and to the meta-data formalism M4 have been proposed in the meantime. An overview is given in the next section. Hopefully, standards are going to emerge in the near future. This could help to foster the idea of maintaining best-practice cases which can be submitted and re-used by the whole KDD community without presupposing specific commercial software solutions.

8.3. Related work

When comparing MININGMART to similar approaches, please keep in mind its basic ideas of how to support the preprocessing phase of KDD:

Integrated platform for preprocessing Users of the MININGMART system can address the whole preprocessing phase of KDD with a single tool that has graphical support and only allows for valid sequences of Steps. Supported parts of this phase contain the modeling of an ontology of domain concepts and relationships, the selection of corresponding relevant database tables, discarding irrelevant attributes manually or automatically, the construction of additional attributes, data cleansing, the aggregation of attributes, computation of statistics, and sub-sampling. However, unlike for most other systems, each intermediate result might as well be accessed from any other application, because the database acts as a natural interface without any need to define standardized languages for exchanging data. In turn, the results of other tools, if written back to the database, may be read and further processed with MININGMART.

High-level specification of data and its transformations The MININGMART system is based on the M4 meta-data formalism, which offers an abstract layer for both, the business data at hand and the transformation steps. The domain ontology establishes an abstract layer on top of the former. This layer helps to better understand the data, because it is formalized in domain-specific terminology, but it also eases the re-use of successful applications in different settings, because only the mapping of the abstract level to the business data needs to be rewritten. Transformation steps are based on abstract operator descriptions, and they address the abstract data level, that is, they refer exclusively to the domain ontology rather than to the business data. Abstract operators are specified in terms of their functionality, encapsulating any details regarding their implementation. Users hence only deal with higher-level operator specifications.

Case-based problem solving Finding a representation of the data which best fits an analysis task and learning algorithm at hand is non-trivial a task. None of the approaches addressing this task automatically currently yields convincingly good results, so the MINING-MART approach relies on a semi-automatic strategy, which supports KDD experts in exchanging and re-using excellent solutions. The M4 meta-data language acts as a convenient medium to communicate successful solutions, as it allows to directly publish Cases at the MININGMART Internet case-base.

This section describes related approaches. Although the goals are overlapping, none of the related approaches realizes the same set of basic ideas. The most relevant work in this field can be categorized into planning-based approaches and KDD specification languages.

Planning-based approaches (subsection 8.3.1) are coupled with integrated KDD systems that aim to support users in setting up valid and effective preprocessing chains. These systems make use of data representations and operator descriptions that allow to apply standard planning approaches, as known from the field of artificial intelligence.

A growing number of *standardized languages* for KDD have been proposed recently, most of which are based on XML. They serve different purposes, starting from specifying and exchanging data mining models up to specifying full KDD applications. These languages are interpreted by KDD systems similar to MININGMART and are reviewed in subsection 8.3.2. A brief overview of further data mining and KDD *systems*, including modern DBMSs that contain data mining algorithms is given in subsection 8.3.3.

8.3.1. Planning-based approaches

The main objective of MININGMART is to support end-users and KDD experts in setting up preprocessing chains that allow for a successful application of a data mining algorithm. Bernstein et al. (2005) address a similar task by *intelligent discovery assistants*. The phases in the KDD process supported by these tools range from later stages of preprocessing to post-processing of data mining models, which includes the selection and parameterization of a well-suited data mining algorithm.

Unlike with MININGMART, where a target view may in principle act as an input to an OLAP application, to a model induction step, and as a basis for further preprocessing at the same time, a user is supposed to specify a single analysis goal in the system IDEA (Bernstein et al., 2005; Bernstein et al., 2002). More precisely, only classification tasks based on a main-memory sized single flat-file of input data in WEKA (Witten and Frank (2000), cf. subsection 8.3.3) format are (currently) supported; but the user may enter preferences like a trade-off between processing time and model accuracy, or assign a weight to the interpretability of the resulting model. IDEA exploits meta-data of the training set, and of a set of preprocessing, data mining, and post-processing operators to suggest operator sequences that induce a model and meet the specified goals. Similar to MININGMART, the operator meta-data contain pre- and post-conditions, which allows to compose executable and meaningful chains, only. The main difference is, that IDEA applies a planning approach to find a sequence of operators that starts with an example set and ends with a well suited model. To this end, heuristic techniques are applied that estimate the time required to execute a preprocessing chain. Further, IDEA uses *auto-experimentation*. This means that half of the available data are used to train models, which are evaluated on the other half, yielding a ranking of alternative model-induction techniques with respect to their estimated accuracies. This approach is very similar to meta-learning (see e.g., Brazdil et al. (2003)).

Bernstein et al. (2005) emphasize that they enumerate all alternative operator chains in the search space, but this is possible, only because they offer a very limited set of operations. For instance, most of the preprocessing and data mining operators are unparameterized. The number of resulting intervals after discretizing, or the amount of data to be sub-sampled, are parameters that need to be fixed ahead of time, because any form of complex parameter-tuning would no longer be fully enumerable. For a similar reason, the system is restricted to well-understood phases of the KDD process in which there are just a few alternative choices at each step. This excludes most of the manual operators part of the MININGMART system from the reasoning process. Please note, that supporting some of them would still be useful in the single table representation of WEKA, e.g., feature construction, but it is not reasonable to just enumerate them. Some phases of the KDD process require user interaction instead, and can probably be supported best in a semi-automatic framework. Real-world applications of knowledge discovery often involve hundreds of steps.

Unintendedly, for most kinds of input data the rankings of the IDEA system are identical and rather simple. Hence, the authors try to assimilate the idea of re-using best-practice preprocessing cases by integrating templates of operator sequences into their system. Templates address higher-level business goals. They contain a few gaps that are filled by the planner at execution time. This seems to be reasonable only, if the user is involved in template selection. Adapting excellent prior solutions is an efficient way of combining human expertise with computational power.

Zhong et al. (2001; 1997) proposed another planning approach. They apply an agent system called GLS, which aims at supporting the overall KDD process, i.e. preprocessing, knowledge elicitation, and refinement of results. Despite the different terminology, mostly a result of the

different (agent) paradigm that work is rooted in, GLS shares many aspects of MININGMART. Agents are similar to the MININGMART operators, and the controller of the GLS system corresponds to the M4 meta-data compiler. Both systems make use of meta-data descriptions for the business data and for the library of operators. Similar as for IDEA, in GLS the operator descriptions specify the pre- and post-conditions that are required for planning. A major difference to IDEA is that the quality of alternative operator chains are not evaluated systematically in GLS, but the system relies on user feedback. In particular, GLS performs no auto-experimentation. The operator libraries of MININGMART and GLS differ in several aspects. For instance the feature generation and selection – a focus of MININGMART – is less developed within GLS. The role of the database is also different. The interaction between GLS and the database is not the primary focus of the research of Zhong et al. (2001; 1997), whereas MININGMART accesses and processes all the data directly in relational DBMSs, compiles meta-data into SQL code, and even has some of its operators integrated into the database. Another difference is the degree of automation and the use of human expertise. In GLS, some user interaction is required in order to optimize the automatically generated sequences. However, the notion of a complete case at the meta-level is not part of the meta-model. This implies that failed attempts along the way to an optimal sequence of agent activities are not documented, and are effectively lost, so similar kinds of mistakes are likely to reoccur. There is no mechanism to apply a successful chain to similar but different databases. MININGMART, in contrast, compiles a successful case together with a meta-model of new data into a running new KDD case.

Both planning-based approaches, IDEA and GLS, have in common that the planning necessarily focuses on "easier" parts of the KDD process which are well-understood and allow to be automated to a certain extent. The systems are basically tailored towards operating on a *single* database table. In contrast, the MININGMART system aims at providing support for steps like selecting just the relevant of maybe hundreds of tables from a huge relational database, joining them to constitute a few highly relevant views, and constructing some semantically even more meaningful features from a data miner's point of view. These steps depend on domain-knowledge and data understanding, so MININGMART relies on the experience of KDD experts and domain experts. This experience can best be communicated by offering access to a large collection of best-practice cases, indexed by their domains and business goals, documented by the developers, and with each part of the conceptual model easily inspectable via a web interface.

8.3.2. KDD languages – proposed standards

Several languages have recently been proposed to support different aspects of KDD. The objective of the Predictive Model Markup Language (PMML) (The Data Mining Group, 2004) is to provide a standard that allows to exchange models between different applications and platforms. A model that has been trained on one system may be exported to PMML format, imported to any other system, and hence be applied or visualized independently of the original source of the model. PMML accounts for the fact that the deployment of a model generally happens in a different environment than its training. With version 3.0 the language has reached a certain maturity. It is supported by several tools, but it solely supports the exchange of *models* augmented by a few very basic transformations of the original input data. To be precise, each kind of model requires its own DTD, so currently only the most common kinds of models are supported. New features of PMML version 3.0 contain the combination and sequencing of models (Raspl, 2004). Combinations are required to formalize bagging and boosting models, which were not available before. Sequencing means, that the prediction of one model acts as an input to another one. In contrast, using a system like MININGMART simply allows to apply one model after an-

other, specifying in each step which of the available attributes the classifier is supposed to use. Changing the PMML language for each supported model is a time consuming and cumbersome process, but it allows to validate each input based on syntactical restrictions.

The background of the XML-based language XDM by Meo and Psaila (2002) are *inductive databases*, an architecture that integrates pattern mining and the storage of the results into a single DBMS framework. Hence, the main goal of XDM is to serve as a unified language for maintaining data tuples on the one hand, and for storing discovered patterns together with the statements that triggered the discovery, on the other. The same formalism allows to formulate (create) statements for inducing patterns, to query for data tuples, and to query for previously created statements and their corresponding results. This functionality requires references as part of the XML-based formalism, which are realized by XPATH expressions. Operators are canonically specified in terms of XML SCHEMA expressions, which allows to efficiently verify the syntactic validity of operator chains.

A disadvantage of the proposed language is a lack of support for abstractions. Regarding the business data, the formalism does not support a conceptual level, so specifications directly address the original schema. Describing operators by XML SCHEMA expressions does not allow to introduce more intuitive abstractions, either. As previously discussed, the domain ontology of MININGMART does not only increase the interpretability of KDD applications, it also eases the adaptation of best-practice cases to other database schemas. Applications formalized in XDM are more tied to a specific environment. Finally, the XDM language lacks support for feature construction and other kinds of preprocessing, which can be explained by its different purpose, to act as a convenient formalism in the scope of inductive databases.

The KDD markup language (KDDML) has recently been proposed by Romei et al. (2006). It is referred to as a *middleware language for KDD* by the authors, and it shares many objectives and properties of the meta-data representation language M4. In fact, in several aspects KDDML goes beyond the functionality provided by M4. This is not surprising, since (i) M4 puts much more emphasis on efficiently supporting preprocessing *directly in relational DBMSs*, whereas KDDML focuses on an integration of various kinds of data, and (ii) KDDML has been released four years later than M4 (Morik et al., 2001; Kietz et al., 2001), so its developers were able to reflect on experiences reported by various researchers in the meantime.

An advantage of KDDML and the corresponding reference implementation are the supported abstractions. Romei et al. (2006) argue that the data representations provided for KDD applications should consist of a physical and a logical level. This is similar to the conceptual and relational level of M4, but the atomic KDDML data fields do not have to correspond to database attributes. Consequently, the variety of data formats is larger than in MININGMART, also covering taxonomies and semi-structured data. The *system* operationalizing KDDML is required to convert all the supported types automatically, hiding technical details like the physical data level from the user. In the database literature such automatic conversations between data formats are referred to as *mediation*, going back to Wiederhold (1992).

Similarly as for XDM, all the patterns and models themselves are stored in the same formalism. Moreover, just as for different data formats, models may as well be specified using different commonly used formats, and are converted automatically on demand. KDDML itself is based on a variety of XML-based formalisms, like PMML and XQUERY. It is supposed to establish higher levels of abstractions that are independent of the lower levels realizing them.

A bibliography of operators is provided, each of which can be referenced by name when specifying a data transformation. This aspect is similar to MININGMART. Less developed features of the reported KDDML reference system are the database access and the GUI. Improving the former to the same extent that is supported by MININGMART seems to require serious additional

efforts, while a GUI is probably already under construction.

Just as in MININGMART, each operator has its own signature, specifying its inputs and outputs. Each instantiation of an operator has an input vector with components of different types, which have to match the signature of the operator's specification. The system supplies all objects referenced by the input vector to the operator that actually processes them and returns another object that has a predefined type. Consequently, operators can be considered to have functional semantics in KDDML (and MININGMART). An advantage of the XML-based formalism is that appropriately defined DTDs allow to check validity with generic XML tools. The validity checking in MININGMART is more complex, but in turn, it allows to formulate a richer set of constraints, which will be discussed in subsection 9.3.3.

Grid and parallel computing is another field where XML is used intensively to specify KDD processes. Cannataro et al. (2004) give an overview of several frameworks, and present their own approach, the so-called *knowledge grid architecture*, which is an additional layer on top of common grid toolkits. The XML-based meta-data language is used to describe (i) computational resources of all hosts in the grid, (ii) the data, which is not necessarily stored in a DBMS, but may as well take the form of semi-structured flat files, (iii) tools for different kinds of processing, including the data mining step, and (iv) models in PMML format. The focus of the approach is on handling the heterogeneity of grid architectures, for instance by providing the means to identify appropriate data and tools in the grid for a problem at hand. A graphical user interface supports users in setting up cases. Validity is verified automatically, the abstract data level is connected to appropriate resources, and the system executes the case. There is no large variety of operators for preprocessing. The focus is rather on operations like copying data from one site to another. Another difference to MININGMART is that data modeling is required (and hence supported) only for technical reasons. There is no ontology to improve the data and domain understanding of human analysts. Consequently, there is also no business layer, that documents the case and allows to communicate operational best-practice cases. Re-usability is not addressed by the knowledge grid approach, but still, it is probably easy to map abstract data to the signatures of other data sources, because this is part of the core functionality of a grid.

8.3.3. Further KDD systems

There is a variety of other KDD systems that share some of the aspects of MININGMART. Some of these tools have an open source license, others are commercial products.

Open source software

The best known open source learning toolbox is probably WEKA by Witten and Frank (2000). It provides a huge variety of state of the art learning algorithms and a few basic preprocessing and validation features. Its integrated nature makes it a very comfortable choice for evaluating and comparing different candidate data mining algorithms in KDD projects. Its main disadvantages are a lack of a convenient database interface, a limitation to single relations (tables) in attribute-value representation, and a lack of support for sophisticated feature selection, feature construction, and other phases of data preprocessing.

YALE (Fischer et al., 2002) is yet another learning environment which is typically applied to data in attribute-value representation. It offers access to all WEKA learners via a wrapper approach, but it overcomes the lack of support for a number of preprocessing operations. Among the strong aspects of YALE are an intuitive graphical user interface that allows to set up complex experiments with only a few mouse clicks, operators for validating the quality of models with

respect to many different metrics, a long list of machine learning operators, a flexible framework for automatic feature selection and feature construction, a simple plug-in mechanism for adding own operators, support for several learning scenarios, tools for visualizing datasets and models, and a mechanism to store trained models, which also allows to apply models to previously unseen data. Several experiments reported in this thesis exploit the flexibility of YALE, sometimes augmented by own operators, but often using WEKA implementations of learning algorithms. Database-related preprocessing is not supported by YALE, however, as it also depends on samples up to main memory size, and does not operate on multi-relational data. The MININGMART system complements the functionality of YALE by focusing on preprocessing very large (multi-)relational databases.

SUMATRATT (Aubrecht et al., 2002) is a JAVA framework for preprocessing. It is tailored towards flat-file data representations, but it supports JDBC, which allows to process volumes of data up to main memory size. Just as in MININGMART, users can set up chains of preprocessing operators. There are three different ways to use the system, setting up chains using a GUI, using a tool-specific scripting language, and implementing own operators in JAVA. In contrast to MININGMART, the system is not tailored towards very large databases, but relies on main memory access. The main focus seems to lie on visualization techniques.

Java Data Mining (JSR-73) (JSR-73 Expert Group, 2004) defines a collection of standard JAVA interfaces, which have become part of the official API. The goal of this effort is to provide a standardized framework to the data mining community that allows each vendor to provide an own data mining implementation. Programming against such a standard interface would, for example, allow to exchange models between JAVA applications, to augment existing applications by own operators, and to even mix implementations of different vendors. Java Data Mining did not yet succeed in becoming a widely accepted standard, which may be due to the lack of a reference implementation. Still, the existence of this collection of interfaces underlines the growing public interest in data mining.

Commercial systems and DBMS with data mining support

There is also a long list of commercial KDD systems that support preprocessing. The SAS ENTERPRISE MINER 5.1[4] offers data mining modules / operators in combination with a programming language for data analysis. The system SPSS CLEMENTINE 9.0[5] offers a graphical user interface that allows to set up operator chains in a similar fashion as possible with MININGMART. One of the major differences is that the system also addresses the data mining step. The system is closed, however, and intermediate results are stored in legacy format flat files[6].

An interesting phenomenon is that all major RDBMS vendors have integrated data mining solutions into their products. Examples for closely coupled external up to fully integrated tools are: the IBM DB2 INTELLIGENT MINER 8.2[7], ORACLE 10G DATA MINING[8], and MICROSOFT SQL SERVER 2005[9]. The latter provides its own XML-based meta-data languages, DMX and XML/A. DMX is a data mining query language that integrates e.g., training and applications of predictive models, and the conversion of PMML models into a SQL-like language. XML/A is a language that aims at providing a standard for accessing analytical data and functionality. A

[4] http://www.sas.com/technologies/analytics/datamining/miner
[5] http://www.spss.com/clementine
[6] There is also a client-server version, which might support database-internal preprocessing. Since CLEMENTINE is a commercial tool and this version was not made available, it was not evaluated.
[7] http://www-306.ibm.com/software/data/iminer/
[8] http://www.oracle.com/technology/products/bi/odm/
[9] http://www.microsoft.com/sql/

proposed general data mining extension to SQL is part of SQL/MM (International Organization for Standardization (ISO), 2003b).

The list of presented commercial KDD solutions is not meant to be complete. It shall rather help to understand the position of MININGMART and illustrate the growing interest in KDD support tools. Please refer to (Euler, 2005a; Euler, To appear) for a study that compares MINING-MART to some of the above-mentioned commercial products in more detail.

8.4. Summary

The relevance of supporting not only single steps of data analysis but sequences of steps has long been underestimated. A large variety of excellent tools exist that offer data mining algorithms, but only very few approaches tackle the tasks of making clever choices during preprocessing and combining these choices to define effective and efficient KDD sequences. Systems like CLEMENTINE offer processing chains to users, but focus on the data mining step, not on preprocessing. Data is extracted from databases and cached in legacy format flat files, which prevents user from switching to other analytical tools. The common data format in tool boxes such as e.g. SPSS or WEKA provides users with the prerequisites to formulate their own sequences (Witten & Frank, 2000). However, even for similar tasks the user is required to set up sequences from scratch each time.

Recently several tools were advertised with a support for the CRISP-DM process model; this illustrates an increasing awareness that the preprocessing phase consumes most of the time in real-world applications, and should be addressed appropriately. Several important goals that are related to KDD processes have the potential to reduce the efforts required for successfully applying KDD techniques drastically in the future, but have not yet been met sufficiently well by existing solutions. Examples include convenient mechanisms that allow to exchange models between systems, a standardized meta-data representation for describing business data, and a declarative formulation of KDD applications. The MININGMART system discussed in this chapter (version 2, released on 12.4.2006) is a non-commercial open source product that already meets a subset of these goals reasonably well. It focuses on preprocessing, so it does not implement the PMML standard for exchanging data mining models, but the M4 meta-data language allows to declaratively describe chains of preprocessing operations and the corresponding business data. MININGMART offers a variety of further advantages:

Very large databases: It is a database oriented approach that easily interacts with all relational databases supporting the SQL standard. It scales up to real-world databases without any problems. Some operators have been re-implemented in order to make them suitable for very large data sets.

Sophisticated operators for preprocessing: Preprocessing can make good use of learning operators. For instance, a learning result can be used to replace missing values by the learned (predicted) values. Feature generation and selection in the course of preprocessing enhances the quality of data that are the input to the data mining step.

Meta-data driven code generation: The MININGMART approach relies on a meta-data driven code execution, which includes the creation of database functions and the execution of SQL statements. Meta-data on operators and business data are used by the compiler in order to generate an executable KDD application.

Case documentation: Meta-data on Cases document the overall KDD process with all operator selections and their parameter settings. In addition, a business layer describes Cases in less technical terms, and thereby establishes a better user interface.

Case adaptation: The notion of a complete Case in the meta-model allows to apply a given expert solution to a new database. The user only has to provide a new data model to the system and the compiler generates a new Case. For fine-tuning the new application, the human-computer interface offers easy access to the meta-model with all operators.

Euler (2005a) compares MININGMART to several commercial KDD systems with respect to a variety of diverse aspects. Further details on how to model KDD cases conceptually, which Case information is stored at the business layer, and how to use it to retrieve Cases from the case-base can be found in (Euler, To appear). Parts of this chapter are based on the publication (Morik & Scholz, 2004).

9. A KDD Meta-Data Compiler

After an overview of the MININGMART system was given in chapter 8, this chapter describes the meta-data driven software and view generation in detail. The objectives of the corresponding compiler module are sketched in section 9.1. To allow for a better understanding of the technical details, the different levels of abstraction, all part of a single meta-model named M4, are discussed in section 9.2. The operational framework described in section 9.3 is based on this meta-model. It consists of an operator taxonomy, parts of which are embedded into the M4 formalism, while others are part of the M4 compiler code. In section 9.4, the data- and control-flows during a compiler run are illustrated exemplarily. Technical details on how to run code at various locations (section 9.5), for instance inside a relational database, or how to use secondary toolboxes for the data mining step after preprocessing with MININGMART (section 9.6) conclude this chapter.

The conceptional ideas underlying the version of the meta-data compiler released at the end of the MININGMART project and large parts of their implementation are the work of the author of this thesis. This excludes the graphical user interface, and the parts that are related to re-usability of best-practice Cases, in particular the Internet case-base.

9.1. Objectives of the compiler

The M4 formalism allows to specify best-practice KDD applications which are not only easy to understand, due to their modular structure and documentation, but which are at the same time *operational*. The module of the system that operationalizes Cases represented in M4 is referred to as the M4 *compiler*. The term *compiler* was chosen in order to point out the analogy to common programming languages. The Human-Computer interface (HCI) supports the user in setting up preprocessing cases and enforces their validity. Hence, MININGMART can be thought of as a graphical programming language for knowledge discovery in databases, in the spirit of e.g., LABVIEW[1] (by National Instruments), a graphical programming language for applications in engineering.

The foundation of relational database managements systems (RDBMSs) was laid by the relational algebra by Codd (1970). SQL is a continuously extended standard query language based on this algebra and supported by all major RDBMSs. The current specification is SQL:2003 (International Organization for Standardization (ISO), 2003a), but many of the more recent extensions are not yet fully supported by all vendors, so any of the older releases can be considered as the de facto standard. Referring to a widely accepted standard allows to abstract from most of the technical details regarding any specific underlying RDBMS. The interface to the logical data layer considered in this thesis is an abstract SQL *virtual machine* which allows to query the database, to create views, and to directly execute specific kinds of code in the database. The role of the compiler is to bridge the gap between higher-level specifications of preprocessing cases that were set up with a graphical HCI on the one hand, and the low-level queries and commands directly interpretable by the SQL virtual machine, on the other.

[1] `http://www.ni.com/labview/whatis/intuitive_graphical.htm`

Before any details of the compiler are presented, the analogy to compilers as known from most higher-level programming languages shall be illustrated. In general, compilers *translate* the code of convenient programming languages into executable native code. By construction of the programming languages' syntaxes this process is computationally cheap for all popular programming languages. At application time the resulting native code is executed, which takes as much time as required by the application itself, but usually much longer than it took to compile the code. Similarly, the compiler of the MININGMART system reads higher-level M4 descriptions of a preprocessing case and its data, and it generates SQL code, primarily in the form of view definitions. Creating a view, or even a longer sequence of layered views is computationally cheap, even if the definition contains expensive joins between database tables. This is considered to be the *compilation step*. The result is a final set of views, which constitute the input to the data mining step. The SQL code is *executed* at the moment of any secondary data mining tool or user reading data from any of the target views. Any complex operation like a join or data aggregation embedded into a view definition may then result in costly operations like full table scans.

Although this scenario captures the underlying idea of the M4 compiler well, in practice there are several operators that require to be executed at *compilation* time. Examples contain learning operators that fill the missing values of an attribute, and automatic feature selection operators. For reasons of computational costs it is not reasonable to run such complex operators based on SQL statements, which conflicts with the overall compiler framework, however. Even worse, in the case of automatic feature selection, the schema of resulting database views may depend on the business data at hand, which makes it impossible to delay the execution of the operator in the case of subsequently applied preprocessing steps. This is discussed in subsection 9.4.2 in more detail. For these reasons several operators are executed at compilation time and the results are written back to the database in appropriate form. In such cases, the term M4 *interpreter* might be more intuitive than M4 *compiler*, but in order to be consistent with prior publications (Morik & Scholz, 2002; Euler et al., 2003; Morik & Scholz, 2004), the MININGMART user guide[2], and several project internal documents, the former term is used throughout this chapter.

9.2. M4 – a unified way to represent KDD meta-data

In section 8.2.1, the meta-model M4 has been presented as a unifying framework for different kinds of KDD-related meta-data. Before introducing the meta-data compiler, it is instructive to distinguish between several kinds of abstractions, which are all realized as components of a single meta-data model. Early specifications of M4 can be found in (Morik et al., 2001; Kietz et al., 2001). The author of this thesis has maintained the M4 model during the implementation of the M4 meta-data compiler[3], which required several extensions and refinements. For a documentation of the final version of the MININGMART meta-model please refer to (Scholz & Euler, 2002).

[2] `http://mmart.cs.uni-dortmund.de/downloads/SYSTEM/UserGuide.pdf`

[3] A simple single-step M4 compiler prototype and a corresponding, yet incomplete relational M4 model were handed over by the former MININGMART partner SwissLife in April 2002. The responsibility for both M4 model and compiler were transferred to the University of Dortmund at this time, and a publicly available report has been sent to the European Commission. The author of this thesis started to extend, and later on to rewrite the M4 compiler according to refined specifications required by the overall system, which exceeded the MINING-MART project period. Some adjustments of the M4 model could hardly be avoided, but will not be discussed in detail in this thesis.

9.2.1. Abstract and operational meta-model for data and transformations

The main purpose of M4 is to store meta-data on the data transformation steps of preprocessing cases, as well as the schemas of the processed data during all phases of a preprocessing case. These two aspects are represented by different parts of the meta-data model, the *data model* shown on the left side of figure 9.1 (conceptual and relational data model), and the *transformation model* on the right side (case model and its executional counterpart). The transformation model specifies the kind of operations that are applied to the data and any corresponding parameter settings. The data transformation usually starts from *raw data* and ends with a format well suited to apply data mining algorithms that induce models. For both transformation model and data model there are two levels of abstraction supported by M4.

The lower *relational* level of the *data model* holds a copy of the business data schema, complemented by some statistics that go beyond those of common database management systems. The relational meta-data contains an M4 object for each view and table of the business data accessed by MININGMART, and further objects that represent Columns, Values, integrity constraints, and statistics. The *conceptual data model*, also referred to as the (M4) *ontology*, established an abstract view on the relational data model, and is mapped to the relational entities. It consists of Concepts taken from the domain terminology, Features of these Concepts, which may be composed of several primitive BaseAttributes, and of Relationships, each between two Concepts. Relationships are meant to reflect domain knowledge, and to abstract from e.g., foreign key references as known from relational databases, which are at the technical, hence relational level. The use of ontologies in modeling the data for KDD processes is still in its early phase, but it promises significant improvements of understandability and re-usability in the future (Euler & Scholz, 2004).

The *transformational part* of M4 contains an abstract *Case model* and a *technical description* of each operator. The abstract *Case model* is based on a predefined library of preprocessing operators. Each operator has a specific functionality specified in KDD terminology, while the implementational details are irrelevant at the conceptual level. Steps are instantiations of operators, connecting them to inputs and outputs specified in terms of the conceptual data model. More complex chains of operations are defined by specifying connections between separate Steps, each embedding its own abstract operator. The mapping from the operators to be executed to the code blocks that realize them is up to the M4 compiler. Details are discussed in subsequent sections. Each technical operator description contains either a reference to a JAVA class part of the compiler code, to executable code running in the database, or to native code, that is connected to the system by a wrapper. Executing code blocks that reside in different locations will be discussed in detail in section 9.5.

The business layer on top of the conceptual layer allows to abstract even further from technical details of the case. It is relevant for different kinds of tasks, for example to retrieve best-practice cases that might be adapted to a new KDD task at hand, or to learn more about prototypical solutions. To illustrate the work of the compiler, however, it is not useful to consider the business layer, so it is not further discussed in this chapter.

9.2.2. Static and dynamic parts of the M4 model

The different kinds of M4 meta-data can also be characterized in terms of the modules that are allowed to change them. This motivates the notions of static, and of different kinds of dynamic data.

Static data describes properties of the MININGMART system that are not supposed to change during a user session or compiler run, but may only be updated by a system developer. Examples

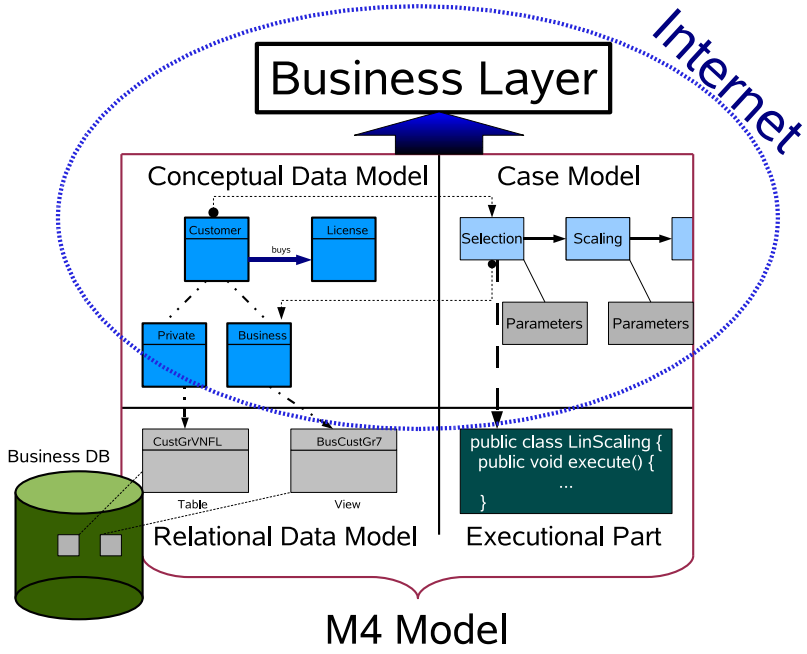


Figure 9.1.: Overview of the four parts of M4, their interdependencies, links to the business data, and the part published in the Internet case-base. Abstraction increases from bottom to top.

contain the list of available operators, formal constraints of their applicability, and assertions regarding their output. The database schema that operationalizes the M4 meta-model is also static, of course.

Dynamic data can be categorized in terms of which kinds of actions may affect it. The *conceptual part* of a Case consists of its Chains, Steps, parameters etc. on the one hand, on the other hand of the corresponding conceptual data, namely Concepts, Features, and Relationships between Concepts. This kind of information is (i) explicitly specified by the user, (ii) set up with system-support, or (iii) fully derived by the system. An example for (i) fully user-specified meta-data, are Concepts connected directly to the database, because the conceptual names and data types required in this case cannot be derived automatically. System support during case specification (ii) is possible when there is a pre-defined list of valid parameters, for instance when selecting a kernel for a support vector machine operator. In this case, the system can offer a specific menu, containing exactly the valid kernel types. An example for (iii) conceptual meta-data completely derived by the system are operator outputs, like the data types and names of output features after a row selection step. The modules of the MININGMART system supporting the user in these settings are the M4 interface[4], the graphical user interface (see also Euler (To appear)), and the operator constraint module presented in subsection 9.3.3. The compiler never changes conceptual data.

The *relational data-model* is the most dynamic part of M4. It contains volatile mappings between Concepts/Relationships and database views, but also between Features and corresponding attributes in the database. Statistics are also a part of the relational meta-data, and are linked to their corresponding views and attributes. Except for the schema of the original business data,

[4]In its original form the interface did not provide this kind of functionality. As part of a system re-design it was re-implemented with several additional features at the University of Dortmund after the end of the project period.

all the relational meta-data is created and deleted on demand at runtime by the M4 compiler. Details about these processes are the subject of section 9.3.

The *case-model's executional level* is not fully reflected by M4. Its static part consists of operator *classes* (hence static) that may be referenced by Steps, while at runtime the compiler maintains individual *instances* (dynamic) of these operators, i.e. executable JAVA objects that realize the specified functionality. Only the static part is visible at the M4 level, while the dynamic counterpart is directly maintained by the compiler in main memory at runtime. Operator classes are instantiated according to the specifications found in the executional model of the preprocessing case at hand. All of the provided parameters are automatically linked to operator instances. To this end, the compiler reads and interprets the static operator constraints with respect to required parameters, and it prepares an appropriate library of parameter instances for each operator instance. The meta-data-driven control flows of the compiler and the corresponding maintenance of a memory-based M4 mirror are discussed in section 9.4.

9.2.3. Hierarchies within M4

In addition to the abstract level on top of the relational data model and the executional part of the Case model (see figure 9.1), the M4 model offers further kinds of abstraction worth noting. The most interesting ones are the taxonomies for Concepts and Operators.

Concept taxonomies

Concepts are organized in two separate *taxonomies*. The first one relates them by the relation super-concept-of, the second one by the relation projection-of. A super-concept contains the same Features as its sub-concepts, but it is semantically less restricted, so its extension is a superset of the sub-concepts' extensions. The Concept of database_users may be modeled as a super-concept of MiningMart_users, for instance. In this case the Concept MiningMart_users inherits all Features of Concept database_users, like name, database vendor and version, but may also have additional Features, like MININGMART version in use, and the number of Cases actually present in the M4 schema. In any case, the extension of the concept MiningMart_users is a subset of the concept database_users, since every user of MININGMART also uses a database. Knowing about this relation between the Concepts allows to visualize the domain Concepts by specific graphical views, which is realized by the concept editor. In contrast to super-concepts, projections of a Concept have the *same* extensions as the original Concept, but the Feature sets are subsets of the original Feature set. A typical example of a projection is feature selection, e.g. to remove features that are irrelevant for the data mining step. Relationships between M4 Concepts may also be specified as specializations of Super-Relationships.

Organizing Concepts with respect to the two taxonomies is a feature supported by the concept editor, see figure 9.2. Users may actively relate Concepts by the super-concept-of, the projection-of, or both of these relations in the concept editor. Some of the operators available in MININGMART are known to produce output Concepts that are related to the input Concept by one of these relations. This property is exploited by the Human-Computer interface to automatically derive parts of the taxonomies. More details are provided in (Euler, To appear).

Operator taxonomies

As most of M4, the *abstract transformational model* is also organized hierarchically. Operators are grouped by their functionality, which naturally motivates groups that are also found in the CRISP-DM model (see section 8.1 or Chapman et al. (2000)), like *data cleansing* and *feature*

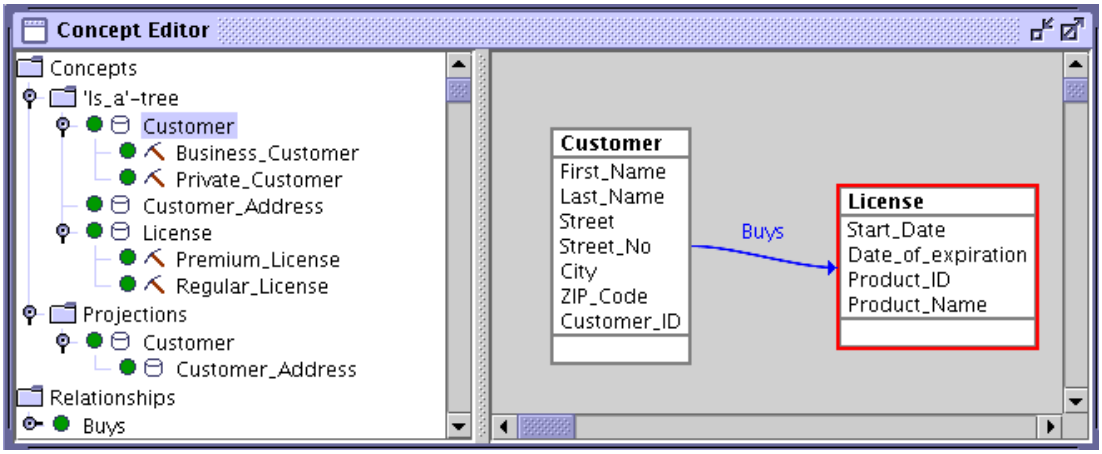


Figure 9.2.: The concept editor allows users to organize their concepts hierarchically in terms of the relations super-concept-of (*Is a* tree) and projection-of. For some Concepts these relations are automatically derived by the system.

construction. These functional groups are mainly used by the Human-Computer interface for menu construction, to help case designers in finding their required operators, and in getting an overview of alternatives that are also supported by the system. The M4 compiler does not access this kind of information.

There is a second, more implementationally oriented operator taxonomy, which is embedded into the compiler and more or less hidden from end-users. It derives specific operators from more abstract prototypes, which is relevant for developers of new operators, and helps to illustrate how the meta-data driven execution of preprocessing cases is realized in a generic way in MINING-MART. This technical operator taxonomy is discussed in more detail in subsection 9.3.4.

9.3. The MININGMART compiler framework

The compiler framework of MININGMART is based on the meta-data representation language M4 described in the last section. The architecture of the meta-data compiler is sketched in subsection 9.3.1. In subsection 9.3.2 it is discussed how to decompose the Case compilation problem into a sequence of single-step compilations. The signatures of MININGMART Operators and semantically oriented dependencies between Steps can formally be stated in terms of the constraints, conditions, and assertions described in subsection 9.3.3. Finally, the structure of the operator library and implementational details are sketched in subsection 9.3.4.

9.3.1. The architecture of the meta-data compiler

The M4 meta-data compiler is a separate module of the MININGMART system, just as the Human-Computer interface (HCI) and the M4 interface. It is to a large extent written in JAVA, complemented by a few platform dependent binaries and some operators directly run inside the business database. Its main objective is to execute the Cases represented in an abstract form.

Figure 9.3 summarizes the interdependencies between the modules that are active during a Case compilation. The compiler accesses the meta-data repository exclusively via the M4 interface, a module that offers a convenient access at the JAVA level. All the meta-data loaded into the compiler are automatically cached for later usage. The overall meta-data compiler can further be

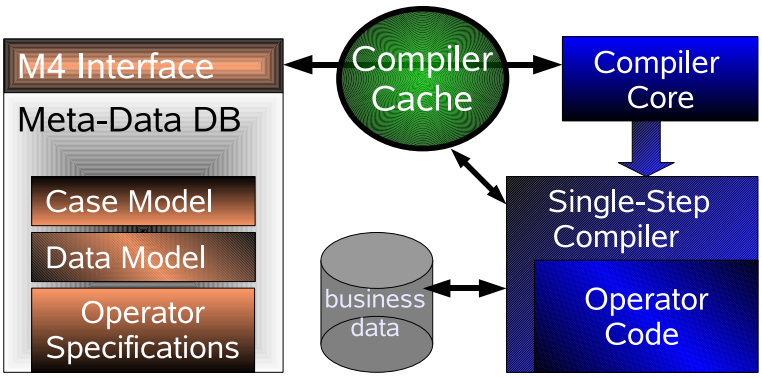


Figure 9.3.: The figure depicts the active modules during a Case compilation. Inside of the M4 compiler we find a *core module*, accessing the M4 meta-data via the JAVA M4 *interface module*. The core calls the *single-step compiler* which executes operators. These operators transform business data and M4 meta-data.

decomposed into the compiler core, the single-step compiler, and the operator library. The compiler core controls the execution of code. It runs the single-step compiler whenever appropriate. The single-step compiler is responsible for locating the operator code corresponding to a given Step, for initializing and for executing it. Operators from the library directly access and manipulate the business database. After successfully compiling a Step, the single-step compiler and operator update the meta-data in cache, which is automatically written back at an appropriate point in time.

The different compiler modules and several interesting aspects of data- and control-flows are discussed in more detail in the remainder of this chapter.

9.3.2. Reducing Case execution to sequential single-step compilation

The focus of this chapter lies on the M4 compiler, and how it makes Cases given in a formal representation operational. At a formal level (see figure 8.2, p. 177) each Case consists of a set of Chains, each of which is composed of a number of different Steps. Whenever the compiler receives the message to compile a complete Case, it needs to analyze the structure of this Case, which then allows to compile single Steps in a valid order. The following paragraphs discuss how the structure of a Case is represented in M4. The descriptions omit most of the database fields (attributes) not used by the compiler.

An anchor object for each Case is stored in an M4 meta-data table named CASE_T[5]. Most of a Case's meta-data is connected indirectly by referencing this object. The tuples of table CASE_T only have an ID, a name, and a status flag indicating the degree of completion, one of DESIGN, TEST, and FINAL.

Each Chain is represented by a tuple in the relation (table) CHAIN_T. It has an ID, a name, a free-text description, and a reference to its embedding case. With a single exception, the integrity of all *references* between M4 objects is guaranteed at a technical level due to foreign key constraints. The IDs of all M4 objects are assigned automatically by a database sequence, and are hence unique across the system.

Steps are M4 entities that refer to an embedding Chain and to an operator they embed. Hence,

[5] In the remainder of this chapter both descriptions are used, the UML representation of M4 classes, and the M4 database tables realizing them.

in table STEP_T there is a reference to table CHAIN_T, and one to table OPERATOR_T. Steps have a name and carry some additional information, which is discussed later on.

The first objective when compiling a Case is to serialize the embedded Steps, which means to compute a valid order of execution. For each pair of two Steps, the M4 representation indicates whether the output of one of the Steps is used as an input by the other one. If (and only if) this is true, then there is a *dependency* to be respected during serialization. This is represented as a corresponding tuple in the M4 table STEPSEQUENCE_T, with two foreign key references to STEP_T, one for each step. Dependency information is prepared by the M4 interface module, hence, the compiler core just needs to perform a depth-first search through a directed acyclic graph, where the nodes refer to Steps and the arcs to dependencies found in STEPSEQUENCE_T. The result is a sequence of Steps valid to be executed one after another by the single-step compilation module.

9.3.3. Constraints, Conditions, and Assertions

Step execution generally requires operator-specific parameter sets. Most operators read an input Concept and either add a Feature to it, or define a similar output Concept. In MININGMART terminology all Concepts, Relationships, and Features addressed during the execution of a Step are referred to as *operator parameters*. In particular, parameters in the more common sense, like the value of C or the kernel type when running a support vector machine are also Parameters of MININGMART operators. They constitute the most simple kind of Parameters, namely *Values*, which are fixed constants after case design.

For handling operator-specific Parameters, the M4 meta-model contains a subset of classes particularly designed for this purpose. This subsection describes this part of the meta-model. Further details on the representation of parameter constraints, conditions, and assertions in M4 can be found in (Scholz et al., 2002). The representation formalism, corresponding extensions of the M4 schema, and large parts of the operator specifications are the work of the author of this thesis (see Scholz (2002a)). An instructive example of how to use the presented formalism, and a complete list of all operator constraint specifications up to December 2002 are also part of (Scholz et al., 2002). Later versions are an integral part of more recent M4 installer scripts[6]. The compiler code that interprets the constraints has also been implemented by the author of this work. This excludes the part of the Human-Computer interface that supports automatic generation of operator output at the conceptual level. The latter heavily relies on the parameter constraint specifications described in this section, however.

Examples for operator specifications and applicability checking

In MININGMART, each data transformation is embedded into a *Step*. Hence, whenever a case designer specifies an operation to transform the data, the Human-Computer interface will set up a new Step referring to a single operator. As illustrated below for two examples, each operator has its own specific set of Parameters:

Example 1: Replace missing values with TDIDT (i) specification of an input Concept (ii) a specific Feature of that Concept with missing values (iii) name and data type of the new Feature to be constructed (without missing values) (iv) parameters of the TDIDT classifier induction algorithm, like the kind of pruning to be applied. If no pruning parameters are provided the defaults are used.

[6]The current installer script is provided at the download page of MININGMART:
`http://mmart.cs.uni-dortmund.de/downloads/downloads.html`

Example 2: Row selection by query (i) specification of an input Concept (ii) a query condition (SQL query) (iii) output Concept (exploit / assume that conceptually an output Concept is to be constructed that is equivalent to the input Concept)

Not only the number of Parameters typically varies between operators, but also the kind of information required during execution, and the sensitivity towards different kinds of data-related details, which may hinder a successful application. Next, some of the characteristic properties for operator specifications are illustrated, before moving on to the formal framework.

First of all, for the examples above, the input Concepts need to be connected to valid tables or view definitions at the relational level before the Steps can be compiled. If this is not the case, the Step is invalid anyway, since no data can be processed. Please note, that before compilation most of the input Concepts are *not yet* connected, but after compiling some of the preceding Steps of a Chain, the views will be defined for the input Concepts, making subsequent Steps valid for execution. Hence, it should at least be guaranteed, that the example Steps above have valid input concepts if all preceding Steps compile properly.

The first example operator above requires an input Feature with missing values as an additional Parameter. Considering the semantics of the operator, it should be guaranteed that this Feature is part of the input Concept. This kind of referential integrity can be verified without any connection to the relational level, because Features are directly linked to Concepts at the conceptual data level. The constraint of a specific Feature having missing values depends on the business data at hand, however, so it can only be evaluated at runtime. The validity of the pruning and learning Parameters also has to be checked by the Human-Computer interface during Case construction, which obviously requires detailed knowledge about each operator to be encoded into the system. Typical examples contain valid Parameter ranges, data types, and whether Parameters are optional or mandatory.

The output Feature as specified by the user needs to be added to the input Concept. This tedious procedure should also be addressed automatically by the HCI and requires additional knowledge, e.g., about the data types of derived Features. As the row selection example suggests, outputs may also contain Concepts, the schema of which can be derived from the input Concept. The precise knowledge about how the HCI should prepare outputs to meet the operator constraints also needs to be encoded explicitly.

Finally, the second example contains a free-text SQL statement. The validity of statements that are formulated at the relational level can only be verified for specific business data schemas, none of which might be available during Case construction. This setting requires specific mechanisms to guarantee integrity.

The examples are simplified, but illustrate a broad variety of the problems actually found in the real MININGMART system. For simplicity it has been decided to transfer the constraints and integrity checking mechanisms to the operator code for some rather specific problems. The majority of the sketched problems can be addressed more conveniently, however, by just specifying operator characteristics in the formal M4 framework of constraints, conditions, and assertions. In the following paragraphs, a structure for the different operator-related constraints is presented.

Operators and parameter constraints

First of all, there is a major difference between the constraints that can already be verified during Case design, and those requiring the business data to be accessible. To simplify reference, only the former kind of constraints is henceforth referred to as *Constraints*. The main characteristic of M4 Constraints is that they only make statements about objects at the conceptual level. This holds for the transformational and for the data-related meta-data.

In both examples we find a list of pre-defined Parameters. In MININGMART there is an M4 class Operator, which holds an instance for each operator available in the system, assigns names to these operators, and provides some high-level descriptions which are discussed later on in this section. Another M4 class called OpParam references the Operator class, and it specifies a Parameter list for each operator. Each OpParam instance refers to a single entry in the list of its referenced operator.

As the optional pruning Parameter for the first example operator illustrates, the corresponding constraints do not necessarily require each Parameter to be present exactly once. Other operators even expect an array of Features as a (single) Parameter in their input list. In order to provide a general solution to these three different kinds of Parameters, all Parameters are handled as *arrays*. The OpParam class holds attributes for the minimal and maximal array size for each operator Parameter. Optional Parameters have a minimal size of 0, while mandatory Parameters have set both, their minimal and maximal size to 1. Arrays generally have a minimal size of 1 and an unrestricted maximal size ($+\infty$), but other combinations like optional arrays (minimum 0) or arrays with at least 2 elements are also supported.

In addition to the cardinality of Parameters, it is generally required that Parameters have a specific *type*. In the examples there is an input *Concept* for both operators, the first of which requires an additional input *Feature*, and pruning/learning Parameters in the form of constant *Values*. The second operator requires an SQL query condition, which also needs to be a constant *Value*. The outputs, a *Feature* in the first example, a *Concept* in the second, are also specified by OpParam objects. In order to enforce the correct data types, the OpParam class contains a dedicated type attribute. Supported types are Concept, Relationship, BaseAttribute, MultiColumnFeature, Feature, and Value. Please note, that if an operator can handle MultiColumnFeatures as well as single BaseAttributes then the type may be set to the abstract type "Feature".

It is important to distinguish between input and output parameters, so that

- the Human-Computer interface is able to support the creation of conceptual output objects as good as possible, and
- the compiler can distinguish at runtime between Concepts required to be connected to views or tables (inputs), and those, for which the compiler is supposed to create a view (outputs).

To this end, the OpParam class contains a corresponding field with possible values Input and Output. The specification of the example operators as discussed up to this point, with M4 classes mapped to canonically renamed M4 database tables, is depicted in table 9.1. The entries are similar to those of two real MININGMART operators. The parameter ThePredictingAttributes is an array of BaseAttributes, which needs to contain at least one attribute in order to allow for tree induction. The PruningConfidence, on the other hand, is an optional Parameter, thus required to have a cardinality of at least 0, but at most 1 Value can be supplied.

Please recall, that all the Parameter objects of class OpParam refer to *static* properties of corresponding operators. Whenever a Step embeds an operator, then the OpParam constraints demand that there is a Parameter *instance* matching each of these static Parameters. If this constraint is met, the operator gets instantiated. Instantiated parameters are represented as objects of the M4 class Parameter. They refer to instances of conceptual M4 objects, like specific Concepts, which are represented as objects of the M4 class Concept, or Values objects, stored in the M4 class Value. Table 9.2 shows extracts of an M4 representation of an example Step embedding the operator with ID 59. There is a separate Parameter for the input Concept, the condition Value, and the output Concept, which reference the Step, and the Parameters to be

ID	Operator name	...
54	MissingValueWithDecisionTree	...
59	RowSelectionByQuery	...

ID	Operator	Parameter name	I/O	Type	Min	Max
685	54	TheInputConcept	Input	Concept	1	1
686	54	TheTargetAttribute	Input	BaseAttribute	1	1
687	54	ThePredictingAttributes	Input	BaseAttribute	1	$+\infty$
688	54	SampleSize	Input	Value	1	1
689	54	TheOutputAttribute	Output	BaseAttribute	1	1
690	54	PruningConfidence	Input	Value	0	1
723	59	TheInputConcept	Input	Concept	1	1
724	59	TheCondition	Input	Value	1	1
725	59	TheOutputConcept	Output	Concept	1	1

Table 9.1.: Simplified entries in tables OPERATOR_T (above) and OP_PARAMS_T

accessed at execution time. For historical reasons and to maintain compatibility, the identification of the OpParam object corresponding to each Parameter instance is done *by name*, rather than by OpParam ID. This means that Parameter objects are assumed to instantiate an OpParam instance, if the name of the latter is a prefix of the former. Considering prefixes allows to aggregate parameters to arrays conveniently.

As illustrated in the following paragraphs, the specification framework discussed so far is insufficient in general. For more complex operators it does not allow to capture the intuitive meaning of operator Parameters.

Other kinds of constraints

The previously shown operator specifications allow to enforce the presence of Parameters with given names and pre-defined types in specific cardinality intervals, distinguished by their I/O-class. Considering the example shown in table 9.1 it is not yet clear without considering additional knowledge how TheOutputAttribute and TheTargetAttribute are related to TheInputConcept of operator 54. Similarly, it is not clear whether there are any restrictions on TheOutputAttribute of operator 59. Furthermore, without additional knowledge about operator 54 the type of TheOutputAttribute cannot be inferred, although it is clear from the context that the attribute constructed by an operator for replacing missing values should have the same datatype as the original attribute. This kind of knowledge can be formulated with respect to conceptual meta-data only, so the term *constraint* as defined before is justified. The remaining constraints are less specific than those captured by the previously discussed M4 tables. In order to allow for a broad spectrum of constraints, an extendable library of pre-defined constraint types has been realized, and another M4 class Constraints has been added to the meta-model. It references constraint types from the library and has two further fields that allow to either reference objects by their OpParam names, or to provide a `String` with further information.

Table 9.3 shows a subset of the original example operators' constraints from the M4 table OP_CONSTR_T. The table contains four important types of constraints. The type IN allows to state that the first argument needs to be included in the second, for instance that a Feature needs to be part of a specific Concept. The type SAME_FEAT is an abbreviation for "same features". The two arguments have to be Concepts whose features have to have identical names and data

ID	Name	Case	Chain	Operator
999	ExampleRowSelection	...	...	59

M4 **table STEP_T**

ID	Name	Step	Type	ObjID	I/O	Nr
...	TheInputConcept	999	Concept	...	IN	1
...	TheOutputConcept	999	Concept	...	OUT	2
...	TheCondition	999	Value	1400	IN	3

M4 **table PARAMETER_T**

ID	Name	Value
1400	TheCondition	City='Sydney'

M4 **table VALUE_T**

Table 9.2.: A simplified example illustrating the instantiation of previously specified Operator RowSelectionByQuery, ID 59 (see table 9.1). All Parameters are referenced by type and M4 ID. In the example the Value Parameter is shown.

types. All BaseAttributes have data types in MININGMART, and without providing sufficient information on the type no BaseAttribute can be created by the Human-Computer interface. The two constraint types TYPE and SAME_TYPE allow to specify such a data type in two different ways, both naming a BaseAttribute as their first argument. The former specifies a valid type as its second argument; a list of valid types can be specified using a set of such tuples. The latter specifies a further BaseAttribute as a second argument, requiring that both BaseAttributes share the same data type.

It is worth to note, that even similar tuples of this table may be used for quite different purposes. The Human-Computer interface uses the tuple with ID 696 to offer a menu of attributes in the case editor that contains only those BaseAttributes, that are part of the previously selected TheInputConcept. Constraints like 703 may even reduce the number of valid candidates. In contrast, tuple 698, which is very similar to 696 is used by the HCI to derive the fact, that it needs to construct a new BaseAttribute, the type of which can be concluded from constraint 699, and that it has to append this new BaseAttribute to TheInputConcept. At all preceding Steps this new BaseAttribute has to be ignored. After its creation it is a regular Feature of the Concept. The compiler uses the same constraints to automatically provide the correct runtime Parameters to each operator, to check the data types if required, and to identify to which entities of the meta-data the resulting views or tables should be linked.

Having a separate static layer of meta-data to specify the operator constraints allows each MININGMART module to exploit these constraints in an individual way. Much of the flexibility and the generic way of handling meta-data would inevitably be lost, if the properties of each operator had to be hard-coded into the different modules. This showed in an earlier phase of the MININGMART project, where the Human-Computer interface contained a specific manually coded class for each operator, which was responsible for creating the meta-data of each operator's output. The maintenance of these classes turned out to consume an unreasonable amount of time, while the generic handling just requires to update specifications or implementations once per adaptation.

ID	Operator	Type	Argument 1	Argument 2
696	54	IN	TheTargetAttribute	TheInputConcept
697	54	IN	ThePredictingAttributes	TheInputConcept
698	54	IN	TheOutputAttribute	TheInputConcept
699	54	SAME_TYPE	TheTargetAttribute	TheOutputAttribute
700	54	TYPE	ThePredictingAttributes	SCALAR
701	54	TYPE	ThePredictingAttributes	CATEGORIAL
702	54	TYPE	ThePredictingAttributes	ORDINAL
703	54	TYPE	TheTargetAttribute	CATEGORIAL
704	54	TYPE	SampleSize	NUMERIC
705	54	TYPE	PruningConf	NUMERIC
728	59	SAME_FEAT	TheOutputConcept	TheInputConcept

Table 9.3.: Incomplete constraints in table OP_CONSTR_T for example operators

ID	Operator	Type	Argument 1	Argument
706	54	HAS_NULLS	TheTargetAttribute	–
707	54	HAS_VALUES	TheTargetAttribute	–
708	54	NOT_NULL	ThePredictingAttributes	–

Table 9.4.: Conditions in table OP_COND_T for example operator

Conditions and assertions

Those applicability aspects of specific operators that are not subsumed by the notion of constraints as defined above are referred to as *conditions* or *assertions* in MININGMART. Both have in common that they directly refer to business data.

Conditions are similar to constraints in that they describe preconditions for operator execution. They are represented in a similar format as constraints, but belong to a separate class of M4. There are three conditions for the first example operator on page 202; the first one is, that the target attribute contains NULL values (type HAS_NULLS), the second, that it does not only contain NULL values (type HAS_VALUES), because otherwise it is not possible to induce a reasonable decision tree to fill in the blank fields. The third condition is that none of the predicting attributes contains missing values (type NOT_NULL). The corresponding entries are shown in table 9.4. Other important kinds of conditions allow to constrain the range of an attribute, for example to avoid negative values when applying logarithmic scaling, or to compare the values of different attributes, e.g., to guarantee that the end of a time interval never precedes its starting point.

Assertions are business-data-related statements that hold after a successful execution of an operator. These statements are sometimes useful to derive subsequent conditions the compiler would have to check otherwise, or to recognize that a concept is a sub-concept or a projection of another one. Two examples are shown in table 9.5. The tuple with ID 709 states that TheOutputAttribute does not contain missing values after a successful application of operator 54. Tuple 733 asserts that after executing the RowSelectionByQuery operator the resulting extension of TheOutputConcept is a subset of the TheInputConcept's extension. Together with constraint 728 (see table 9.3) this implies that TheOutputConcept is a sub-concept of TheIn-

ID	Operator	Type	Argument 1	Argument 2
709	54	NOT_NULL	TheOutputAttribute	–
733	59	SUBSET	TheInputConcept	TheOutputConcept

Table 9.5.: Assertions in table OP_ASSERT_T for example operator

putConcept, which is exploited when maintaining the sub-concept hierarchy. For a full list of constraints, conditions, and assertions please refer to (Scholz et al., 2002).

Further forms of applicability checking

In addition to the previously discussed meta-data driven mechanisms of checking applicability, there are several kinds of runtime errors that are handled individually by each operator. Maintaining a generic framework for each very specific condition is too costly, so it has been taken care that the code yields understandable exception messages whenever too specific kinds of conditions are violated. These messages are communicated to the user via the Human-Computer interface. The user may then adjust the parameter settings, the business data, or the general layout of the Case in order to circumvent the described problem. The generic compiler framework discussed later on in this chapter calls initialization methods and fetches parameters automatically. This still allows to make use of generic structures to catch most of the exceptions not avoided by checking constraints and conditions in advance.

As mentioned after the motivating example on page 203, there are some powerful operators for which the responsibility of providing correct parameters rests with the user. One example is an operator called GenericFeatureConstruction. It allows to specify arbitrary SQL statements to define a new feature based on existing features of an input Concept. This operator is meant to be a fall-back option in situations in which a user requires a very specific kind of computation that is not covered by other MININGMART operators. The draw-back of this approach is, that a full condition checking would require to parse the SQL statement, taking into account any functions, procedures, and other DBMS-specific dialects, and to explain to the user in detail any encountered problems with the syntactical structure of statements, divisions by 0, or references to attributes not present in the data.

As a cheaper alternative, the operator allows to address attributes by their conceptual (BaseAttribute) names, which are replaced by the relational names by the compiler to avoid dependencies on the business data schema that are hard to be controlled, but if the statement fails, the DBMS error message is just propagated without any modifications.

In other situations, problems can be avoided by a fool-proof operator design. The real MININGMART counterpart to the example operator RowSelectionByQuery uses a more complex way of setting up the condition parameter, for example. Each base condition statement consists of three parts, a BaseAttribute, a condition operator (like $<$, $=$, or $>$), and a constant value or string to be compared to the BaseAttribute in terms of that condition operator. Such base condition statements can then be assembled into a full statement, which is finally translated into an SQL query and embedded into the output view definition by the compiler. This design has basically two advantages. First, it avoids to address attributes at the relational level, which otherwise would not only create schema-specific dependencies, but also make it much harder to verify referential integrity. Second, conditions are constructed in a way that makes it hard to produce an SQL exception. This increases the overall robustness of the system.

9.3.4. Operators in MiningMart

The last subsection illustrated how Steps are represented in M4, how Parameter integrity is defined in terms of static M4 objects, and how Parameters are linked to Steps depending on the embedded operators.

As the final issue in this section it is discussed, how operators are realized technically. Related discussions on the meta-data driven data- and control-flows are postponed to section 9.4.

How MiningMart benefits from object oriented design

We can roughly distinguish between three active modules during case compilation, the *compiler core*, which controls the execution, the M4 *interface*, which offers a convenient way to access the M4 model, and the classes of the *operator taxonomy*.

The MININGMART compiler and all of its associated taxonomy classes are implemented in the JAVA programming language. Some learning tools and DBMS specific code fragments are exceptions to this rule, but are still connected to the overall framework via JAVA wrappers.

High-level interfaces Much of the required functionality that is not directly related to specific operators has been "sourced out" to the compiler core, or to the M4 interface layer. Accessing this functionality by calls to standardized convenient interface methods increases the understandability of the code and eases its maintenance significantly. An intuitive example for *encapsulated lower-level functionality* in MININGMART are database accesses like SQL queries and other statements. As argued before, the philosophy of MININGMART is to bridge the gap between conceptual Case and data descriptions on the one hand, and low-level access to a virtual SQL machine, on the other. In practice, however, some of these low-level accesses depend heavily on the specific underlying DBMS. ORACLE allows to execute JAVA stored procedures inside the database, for example, while POSTGRES databases do not[7]. Several details on PL/SQL functions, procedures, sequences, name spaces, transaction management, and system tables also vary from one DBMS to another. Hence, the layer of a virtual SQL machine needs to be established artificially, which is to a large extent realized by the `edu.udo.miningmart.db` package. This package encapsulate all DBMS related issues, like connecting users to the database, creating statements, closing JDBC `ResultSets`, and transaction management. An object of class `DB` plays the role of the interface to the MININGMART operators and to the M4 interface. It offers methods directly addressing the virtual SQL machine, like

```
public void executeBusinessSqlWrite(String sqlStatement),
```

which executes an SQL statement on the business data schema, or

```
public String executeM4SingleValueSqlRead(String sqlQuery),
```

which returns the one-value-result of an SQL query to the M4 schema as a `String`. Other examples for provided methods are

```
public M4Object getM4Object(long m4Id, Class m4Class),
```

which returns an arbitrary M4 object identified by its class and ID, and

```
public void commitBusinessTransactions(),
```

[7] The manual reports that there are ways to extend the base functionality of POSTGRES databases to run all kinds of code, but this requires additional efforts and is not discussed in this thesis.

which commits all changes to the business data schema. The class `DB` itself does not access the database directly, but it acts as a convenient interface to lower-level DBMS-specific sub-classes of `DbCore`, currently one of `DbCoreOracle` and `DbCorePostgres`. Other global services, like for printing debug messages, are encapsulated in a similar fashion.

Top-down specialization of operators The MININGMART architecture for operators is highly modular. Each operator can be considered to be realized as a small module of its own, connected to the compiler core by implementing against a JAVA API. The operator names specified in the M4 class Operator are not only used in GUI menus, but are interpreted as unqualified operator *class names* (case sensitive) at the same time, located in a specific JAVA package of the compiler module. When it comes to executing a specific operator, the compiler core identifies a corresponding JAVA class via the self-reflection API, and it instantiates it with its runtime parameters, according to the operator specifications discussed in section 9.3. Details of the initialization procedure follow in section 9.4.

From a technical point of view, many operators have a similar functionality. To avoid redundancies, and in accordance with the object oriented programming paradigm, most of the operator-specific functionality is hence realized in abstract super-classes of those classes finally run by the compiler core. These abstract classes are not referenced by the class Operator or any other M4 entities. In MININGMART executable operator classes usually add just a few fields and lines of code to the inherited fields of their super-classes, reflecting their very own characteristics.

Among the convenience methods that *all* operators already inherit from the most abstract operator class `ExecutableOperator` are several methods to directly access the operator's runtime parameters and related M4 objects, like the embedding Step. Other methods of the class `ExecutableOperator` allow to print or log debug and status messages, and to directly access the virtual SQL machine.

The set of operators is organized hierarchically. Only the leaves of this taxonomy are non-abstract, thus executable and registered in the M4 class Operator. All abstract classes in between can be considered to cluster operators by technical homogeneity. Level one starts with distinguishing by the kind of output the operators create: Some operators create a new Concept that contains the results of the preprocessing step, while others just add a new Feature to an existing input Concept. The former kind of operators are called `ConceptOperators`, the latter are referred to as `FeatureConstructionOperators`. Both of these groups are described briefly in the following paragraphs. Only some very specific operators fit neither of these categories. Please refer to (Euler, 2002b) for a full list of all MININGMART operators[8] with the OpParam specification and a brief description for each of them.

ConceptOperators

The first major group of operators reads (among other parameters) an input Concept and yields an output Concept. This allows to prepare a more specific framework as for general operators.

To this end, an abstract class `ConceptOperator` is introduced. `ExecutableOperator` already implements the `execute()` method run by the compiler core. The method enforces a specific structure for all operators, but apart from printing status messages it just calls two abstract methods, which need to be implemented by all operators. The first of these methods is responsible for generating an SQL statement reflecting the result, while the second one tests whether the statement is executable. The class `ConceptOperator` provides specific getter

[8] The document is slightly outdated, as it only lists operators added before April 2003.

- mmart.compiler.operator.ConceptOperator
  - mmart.compiler.operator.MultipleCSOperator
    - mmart.compiler.operator.Segmentation
      - mmart.compiler.operator.SegmentationStratified
      - ...
    - mmart.compiler.operator.Unsegment
  - mmart.compiler.operator.SingleCSOperator
    - mmart.compiler.operator.FeatureSelection
      - mmart.compiler.operator.FeatureSelectionByAttributes
      - mmart.compiler.operator.FeatureSelectionWithSVM
    - mmart.compiler.operator.ModelApplier
      - ...
    - mmart.compiler.operator.MultiRelationalFeatureConstr.
    - mmart.compiler.operator.RowSelection
      - mmart.compiler.operator.DeleteRecordsWithMissingVal.
      - mmart.compiler.operator.RowSelectionByQuery
      - mmart.compiler.operator.RowSelectionByRandomSampling
      - ...
    - mmart.compiler.operator.TimeOperator
      - mmart.compiler.operator.ExponentialMovingFunction
      - mmart.compiler.operator.Windowing
      - ...
    - mmart.compiler.operator.Union
    - ...

Figure 9.4.: Incomplete taxonomy of ConceptOperators.

methods for the fixed parameters TheInputConcept and TheOutputConcept, and it exploits the fact that (parts of) input and output are known. Both abstract methods of the super-class are implemented to realize the construction of a view according to an operator's returned SQL statement as part of a Columnset, to test it for validity, and to connect it to the output Concept. All that sub-classes have to do when extending class ConceptOperator is to yield *Columnsets* that reflect the results of their Steps. A subset of the ConceptOperator taxonomy is depicted in figure 9.4.

Examples To illustrate how abstract operators are refined level-wise until an executable operator is reached, we exemplarily follow the taxonomy down to the simple operator DeleteRecordsWithMissingValues. The group of ConceptOperators can further be divided by distinguishing between those operators that yield exactly one output Columnset per input Columnset, and those discussed shortly, that may yield several Columnsets. Each operator of the former group extends class SingleCSOperator, which basically just simplifies class ConceptOperator by implementing the abstract methods as to ignore non-applicable loop-like facilities, and to just call the abstract method generateSQLDefinition(...) that yields a single String, instead. One class extending SingleCSOperator is RowSelection.

It implements the abstract method `generateSQLDefinition(...)` so that it yields a complete SQL statement that creates a new view, including a `select`, `from`, and a `where` part. Just the condition of the latter part is the result of a new abstract method called `generateConditionForOp(...)`. The executable `DeleteRecordsWithMissingValues` class finally just extends `RowSelection` by implementing `generateConditionForOp(...)`, so that it reads the parameter `TheTargetAttribute`, determines the relational counterpart of this conceptual BaseAttribute, and returns "`<attribute name> IS NOT NULL`". Without exception handling this takes 3 lines of code. To understand why the additional layers of abstraction simplify the code of the operator taxonomy, please note that there are already 5 executable operators extending the very specific class `RowSelection`, and 31 that extend `SingleCSOperator`. But even in the case of `ComputeSVMError`, an operator that is the only subclass of `EvaluateResults`, the abstract class in between realizes a more general idea in a way that allows for easy extensions of the operator library at this point in the future.

Another, quite different example of a `SingleCSOperator` worth being mentioned is the `MultiRelationalFeatureConstruction`. Despite its name, which may be misleading at a technical level, it does not construct a single Feature, but it rather constructs a new Concept which is composed of Features found at several different input Concepts. All of these Concepts can be joined using a set of specified Relationships. This operator illustrates, how the MINING-MART parameter specification framework supports very different kinds of parameter sets, arrays of input Concepts and Relationships of arbitrary (but matching) cardinality. The inherited fixed Parameter `TheInputConcept` of cardinality 1 is still present, but there is another Parameter with an array of Concepts linked to `TheInputConcept`, a further one consisting of an array of selected Features, and a final one, containing an array of Relationships to join all the Concepts. Two entries in class Constraint make sure that Concepts and Relationships match, three further constraints guarantee that the selected Features are in fact selected from the input Concepts, and that they are all copied to `TheOutputConcept`.

Multi-Steps An interesting feature of MININGMART is that it allows to execute operators for multiple inputs without requiring any changes to the conceptual case model. It can generally be assumed that there is exactly one view or table registered per input Concept when executing a Step. In M4, tables and views are both represented by Columnsets, so in this case a Concept is linked to a single Columnset, which references a view or table in the database. If the compiler core finds more than one Columnset linked to a Concept, then it executes the Step once for each input, which results in as many output Columnsets[9] as found in the input. The only group of operators that may increase the number of output Columnsets compared to the number found in the input extends the abstract class `MultiCSOperator`. This is, for instance, done by `SegmentationStratified`. The objective of this operator is to split a single input view or table to a set of different views, one for each value of a specified target attribute. As a marker for operators with this property there is an extra boolean field Multi in the M4 class Operator.

A possible application is to split collected sales data of a supraregional supermarket, for instance by shop. While the original table contains all the data, after segmentation we have one separate view per shop. All of these views, referenced in M4 by Columnset objects, are linked to *a single output Concept*. Subsequent analysis steps based on this Concept are executed once for each shop-related view, and they again produce one output view and Columnset per shop that is related to a single Concept. In several contexts this procedure eases the analysis (Euler, 2005b).

[9] This implicitly assumes that the Step embeds a `ConceptOperator`, but for other kinds of operators the result is similar.

mmart.compiler.operator.FeatureConstruction
- mmart.compiler.operator.Discretization
  - mmart.compiler.operator.ManualDiscretization
    - mmart.compiler.operator.NumericalIntervalManualDiscr.
    - mmart.compiler.operator.TimeIntervalManualDiscr.
- mmart.compiler.operator.GenericFeatureConstruction
- mmart.compiler.operator.Mapping
  - mmart.compiler.operator.MappingWithDefaultValue
  - mmart.compiler.operator.PartialMapping
- mmart.compiler.operator.MissingValues
  - mmart.compiler.operator.AssignDefault
  - mmart.compiler.operator.MissingValuesWithRegressionSVM
  - ...
- mmart.compiler.operator.Scaling
  - mmart.compiler.operator.LinearScaling
  - mmart.compiler.operator.LogScaling
- mmart.compiler.operator.SVMforDataMining
  - mmart.compiler.operator.SupportVectorMachineForClassif.
  - mmart.compiler.operator.SupportVectorMachineForRegress.
- ...

Figure 9.5.: Incomplete taxonomy of `FeatureConstruction` operators.

This is referred to as the *multi-step* feature of MININGMART. Please note, that the number of corresponding views / Columnsets depends on the business data, so it is not possible in general to circumvent segmentation by preparing *separate Concepts* for the different shops. A re-union of all views is also possible after a segmentation Step. It is sufficient to insert a Step embedding an `Unsegment` operator, referencing the original Segmentation condition. As a test feature the M4 compiler offers a *lazy mode*, which avoids multi-step execution by only processing the first Columnset of each Concept.

Multi-step handling allows to perform an analysis of an arbitrary number of views without changing the case model. If MININGMART is considered to be a graphical programming language for knowledge discovery in databases, then this element resembles loop structures in standard programming languages.

FeatureConstructionOperators

The second large group of MININGMART operators is characterized by the property of constructing a new BaseAttribute as the only output. The new feature is added to the input Concept of the Step. One of the existing BaseAttributes is selected as `TheTargetAttribute` parameter. It serves as a blueprint for the new Column, which is then linked to the output BaseAttribute.

Operator hierarchy The class `FeatureConstructionOperator` is an abstract superclass of all operators of this group. It provides getter methods for all three parameters mentioned above, but it also exploits the predefined input/output properties of the operator group to prepare most of the necessary functionality. Similar to the `ConceptOperator` class, it implements all abstract methods of `ExecutableOperator`, which are responsible for creating relational meta-data and for validity checking. Unlike for `ConceptOperator`s, however, creating the output does not require the specification of a full view definition, but is much simpler for adding a single new attribute. It is sufficient to update the relational M4 meta-data, namely to register a new Column to the Columnset linked to `TheInputConcept`. This Column needs to provide an SQL sub-statement suited to define a new attribute if added to the SELECT-part of the corresponding Columnset definition. That way, attributes are added just *virtually* to tables or views. This is possible, because the Column definitions are correctly substituted by all modules of the MININGMART system. The only abstract method all sub-classes of `FeatureConstructionOperator` implement is

```
abstract String generateSQL(Column targetColumn),
```

which reads the Column linked to `TheTargetAttribute` and returns the SQL specification for the virtual column to be constructed.

Virtual Columns shall be illustrated by the following example: We consider a table sales with two attributes number_of_items and price_per_item, a Columnset linked to the input Concept of a Step and referencing the sales table, and a Column per database attribute, both linked to the Columnset. In MININGMART, a feature construction operator might just add a new Column named total_price to the Columnset, which has the SQL definition `(number_of_items * price_per_item)`. In subsequent Steps this Columnset is interpreted as

```
SELECT number_of_items, price_per_item, (number_of_items *
       price_per_item) AS total_price FROM sales.
```

This is equivalent to defining a view that adds the constructed feature explicitly. Because there is only one comparatively simple abstract method left to implement, the hierarchy of feature construction operators is shallow. Figure 9.5 shows most of the corresponding part of the operator taxonomy. A single layer of intermediate abstract classes aggregates similar operators, forming groups of discretization, grouping, mapping, and scaling operators. Most of these classes do not add much functionality. The abstract class `Scaling`, for instance, does not provide *any* code, but merely groups all scaling operators together.

Looped parameters All `FeatureConstructionOperator`s share a specific property not discussed so far, which allows to reduce the number of required Steps if a similar operation has to be applied to a Concept several times. The mechanism is referred to as *looping*. Similar as with the multi-step property, there is a boolean marker field named Loop in the M4 class Operator, which indicates whether an operator supports being applied in loops. Operators supporting this functionality can be supplied with multiple Parameter sets and are executed once for each of them. More precisely, each Parameter of a loopable operator may itself share the loopable property or not. This is indicated by objects of the M4 class Constraints. Parameters not loopable are supplied once per Step, loopable ones have to be supplied once per loop.

An example application for a looped `FeatureConstructionOperator` is a scaling operation, for example `LinearScaling` to normalize attributes. Without exploiting the loop mechanism the corresponding scaling operation would take one Step per attribute. Looping allows to address another attribute in each loop of the *same* Step, so that a single Step – maybe

ID	Operator name	Loop	Multi	...
44	LinearScaling	YES	NO	...

M4 **table OPERATOR_T**

ID	Operator	Parameter name	I/O	Type	Min	Max
553	44	TheInputConcept	Input	Concept	1	1
554	44	TheTargetAttribute	Input	BaseAttribute	1	1
555	44	NewRangeMin	Input	Value	1	1
556	44	NewRangeMax	Input	Value	1	1
557	44	TheOutputAttribute	Output	BaseAttribute	1	1

M4 **table OP_PARAMS_T**

ID	Operator	Type	Argument 1	Argument 2
558	44	IS_LOOPED	TheTargetAttribute	–
559	44	IS_LOOPED	NewRangeMin	–
560	44	IS_LOOPED	NewRangeMax	–
561	44	IS_LOOPED	TheOutputAttribute	–
562	44	IN	TheTargetAttribute	TheInputConcept
563	44	IN	TheOutputAttribute	TheInputConcept
564	44	SAME_TYPE	TheTargetAttribute	TheOutputAttribute
565	44	TYPE	NewRangeMin	NUMERIC
566	44	TYPE	NewRangeMax	NUMERIC
567	44	GT	NewRangeMax	NewRangeMin

M4 **table OP_CONSTR_T**

ID	Operator	Type	Argument 1	Argument 2
568	44	NOT_NULL	TheTargetAttribute	–

M4 **table OP_COND_T**

ID	Operator	Type	Argument 1	Argument 2
569	44	NOT_NULL	TheOutputAttribute	–
570	44	GE	TheOutputAttribute	NewRangeMin
571	44	LE	TheOutputAttribute	NewRangeMax

M4 **table OP_ASSERT_T**

Table 9.6.: Specification of the loopable operator `LinearScaling`.

ID	Name	Step	LoopNr	Type	ObjID	I/O	Nr
...	TheInputConcept	997	0	Concept	1477	IN	1
...	TheOutputAttribute	997	1	BaseAttribute	1423	OUT	2
...	TheTargetAttribute	997	1	BaseAttribute	1470	IN	3
...	NewRangeMin	997	1	Value	1480	IN	4
...	NewRangeMax	997	1	Value	1482	IN	5
...	TheOutputAttribute	997	2	BaseAttribute	1484	OUT	6
...	TheTargetAttribute	997	2	BaseAttribute	1473	IN	7
...	NewRangeMin	997	2	Value	1489	IN	8
...	NewRangeMax	997	2	Value	1490	IN	9

Table 9.7.: Example of PARAMETER_T entries for a looped `LinearScaling` with ID 997. Only the Concept (type *CON*) is a global parameter provided only once. For the BaseAttributes (*BA*) and values (type *V*) a loop number is specified for reference.

followed by a manual feature selection to discard the original attributes – is sufficient to normalize all attributes. The input Concept would typically not be modeled as loopable, to reduce unintended side-effects, but `TheTargetAttribute` and all scaling Parameters would.

Hence, for loop Steps the returned set of Parameters depends on a *loop number*. For Steps embedding loopable operators the total number of loops are specified by the attribute LoopNr of the M4 class Step; the mapping of runtime parameters to the different loops is realized by another attribute of the M4 class Parameter, named StLoopNr. The example operator `LinearScaling` has the specification depicted in table 9.6. Exactly those parameters for which a IS_LOOPED constraint exists are treated as loopable. Constraint 567 states, that the new upper interval bound after scaling (NewRangeMax) has to be strictly greater than the lower bound NewRangeMin. As a condition of the operator, the target attribute (which is to be scaled) must not contain missing values (tuple 568). The operator asserts that in this case the output attribute will also contain no missing values (tuple 569), and that all values will lie inside the target interval of scaling (tuples 570, 571).

For each loop iteration a different set of parameters `TheTargetAttribute`, `TheOutputAttribute`, and scaling parameters are used. An example of runtime parameter settings for a Step embedding a `LinearScaling` operator is shown in table 9.7. For each parameter there is a separate tuple in the M4 meta-data table PARAMETER_T. All parameter entries have an M4 ID, name, unique parameter number, and I/O type. To reference a parameter object they specify the target object, e.g. Concept, and the M4 ID of the parameter. There are two loops in the example. Only TheInputConcept is not looped, so it is specified only once, by convention with loop number 0. All looped parameters refer to one of the two loops, so they have a loop number of 1 or 2. They are ignored by the compiler when executing other than these specified loops.

Extending the library of operators

As mentioned before, the MININGMART architecture for operators is highly modular. In principle, one can think of each operator as a separate module that is loaded and executed by the compiler core according to specifications found in static M4 meta-data tables. The operator taxonomy just simplifies the code. It helps to recognize underlying similarities between different operators like `DeleteRecordsWithMissingValues` and `RowSelectionByRandomSampling`, but even more important, it allows to avoid redundancies. All the functionality

common to a group of operators is implemented just *once* in the most abstract class of that group.

As a consequence, if a developer wants to extend the operator library, it is possible to avoid most of the low-level programming, just by finding the point in the operator taxonomy where the own operator fits in best, and by extending the corresponding abstract super-class. The extended class should lie as deep as possible down in the taxonomy, as this reduces the required amount of additional code. In this case, it is sufficient to implement very specific methods, just characterizing the very own properties of the new operator. As a rule of thumb, a new operator should rather be designed as a `FeatureConstructionOperator` than as a `ConceptOperator` whenever possible, because the former class usually allows for simpler implementations. Relationships and multiple input Concepts are the hardest parts to be handled technically, although they are best suited to reflect the multi-relational structure of relational databases.

In order to make a new operator visible to the different components of MININGMART, any operator developer has to provide a full specification in terms of the M4 classes discussed in subsection 9.3.3. The name of the new operator, which is at the same time its JAVA class name, its reference ID, and markers whether loopable and/or multi-stepable or not have to be appended to the M4 class `Operator`, represented by the system table OPERATOR_T. The full list of operator Parameters has to be specified in table OP_PARAMS_T, while their inter-dependencies, supported conceptual data types for input Features, and unique data types to be assigned to output Features have to be added to OP_CONSTR_T. After adding the operator to one of the operator groups found in OP_GROUP_T, the Human-Computer interface is already able to create, update, and verify Steps that embed the new operator. Missing conditions and assertions can currently safely be considered to be optional.

The new operator class is only required when a Case that contains a Step embedding the new operator is compiled. The operator class has to be located in `mmart.compiler.operator`. Convenient methods that support to read the runtime parameters, to return results, to output log and status messages, and to throw understandable exceptions are available at all branches of the operator taxonomy. For a detailed description on how to implement new MININGMART operators precisely, please refer to (Euler, 2002a).

9.4. Meta-data-driven handling of control- and data-flows

The control-flow mechanisms underlying the MININGMART compiler software rely on M4 meta-data. As discussed in section 9.2, M4 offers static and dynamic parts, as well as several kinds of abstraction. In section 9.3 the framework for operator specifications was described. It was shown how operators are instantiated in M4, and how Cases are sequentialized with respect to the input-output-dependencies of their Steps. The M4 compiler executes a JAVA class which is referenced in M4 table OPERATOR_T, and it provides all runtime Parameters linked to the embedding Step. The constraints define the set of valid Parameter settings, which are (to a large extent) verified by the Human-Computer interface while setting up preprocessing Cases.

In this section, the control- and data-flows are discussed in more detail, which addresses the question of how the meta-data tables are precisely accessed by the M4 compiler. In MININGMART such functionality is provided following a generic framework, as opposed to methods implemented individually for each compiler and M4 interface class. This is illustrated exemplarily by showing how the M4 compiler loads the Parameters of Steps and verifies their validity, how M4 entities are accessed and represented by the M4 compiler software, how integrity of data is achieved at the JAVA level, and how Cases are exported and imported to and from XML files.

9.4.1. The cache – an efficient interface to M4 meta-data

The M4 compiler uses an own internal JAVA-based cache for all the entities read from the M4 model, and for all newly created M4 objects. More precisely, only previous versions of the compiler had a separate cache, which was different from the M4 interface based on an application server architecture. After the end of the MININGMART project, the compiler cache was extended at the University of Dortmund, and has, by the end of 2003, fully replaced the old M4 interface for reasons of efficiency.

There are several reasons for caching M4 meta-data in main memory. The most important one is a significant decrease of the communication overhead and runtime, compared to querying each meta-data tuple directly from the database. Another reason is, that it allows to set up a unified JAVA framework for storing and accessing meta-data, following the object-oriented paradigm. Structural aspects of M4 meta-data, i.e. the references between M4 objects, are very important during Case compilation, so it is straight-forward to represent objects and references by corresponding JAVA objects in main memory. There is a JAVA class extending the super-class `edu.udo.miningmart.m4.core.M4Object` for each M4 table. Every M4 object read from the database or created by the compiler (or by other modules in later versions) is internally represented as a JAVA object of the corresponding class. To allow for convenient caching, it is important that all M4 objects have an ID that is unique throughout the system. To this end, MININGMART uses a single database sequence that yields a unique primary key whenever a new meta-data tuple is created, regardless of the target M4 table. A final reason for caching is that objects representing meta-data allow to naturally associate additional runtime information. Whether a Step has already been compiled or not is stored along with the corresponding `Step` object, for example, although there is no corresponding M4 field for this kind of information.

The M4 cache is realized by the class `edu.udo.miningmart.db.DB` mentioned above, which encapsulates all direct accesses to the database. From a technical point of view, the cache is a private dictionary of the only instance of class `DB`, using the M4 ID as the key attribute. There is one method to clear the cache, which is e.g. invoked when closing a Case, one method to store objects in the dictionary, which only works for objects already having a valid and unique ID, one method to remove objects that have become invalid, and one to query for an object by its ID.

M4 objects are never directly read from the database by operators, but there is a unique access method

```
public M4Object getM4Object(long id, Class m4Class)
```

for this purpose. It first checks whether the requested object has been cached before. If not, it initializes an `M4Object` of the specified type according to the data of the corresponding tuple, which is returned and stored in the cache. The JAVA representation actually used by the M4 compiler consists of linked objects of type `M4Object`. Associated `M4Objects` can be fetched using getter methods of the returned object. Some of the related objects are directly loaded together with the explicitly requested objects, while others will be read on demand using the same method `getM4Object(...)` as mentioned above. All objects that have once been read from the meta-data tables can also be accessed by ID via the M4 JAVA cache.

Maintaining inter-M4 relations

The most typical kind of links between pairs of related M4 classes is $1 : n$, that is, each object of a first class references a *set* of aggregated objects of a second one. An example at the conceptual level is the relation between Concepts and Features. While Concepts contain sets of Features

each Feature belongs to exactly one Concept. Similarly, each Case contains a set of Chains, each Chain a set of Steps, each Step a set of Parameters etc. The same kind of relation is found at the relational level, where Columnsets refer to sets of Columns and to sets of ColumnsetStatistics. At the intersection of conceptual and relational level, Concepts are linked to sets of Columnsets and BaseAttributes reference sets of Columns[10].

The compiler and other modules need to follow references between related M4 objects in both directions; on the one hand, it has to be possible to query the set of Features for a given Concept, on the other hand, each Feature refers to a unique Concept, which also has to be accessible efficiently, given just the Feature. In order to allow for efficient random access along any of the links and starting from arbitrary M4 objects, each object stores JAVA references to all directly related objects locally. The class `Concept` contains

```
private Collection myFeatures;
```

as a field with corresponding public access methods, for example, but the class `Feature` contains corresponding methods for its field

```
private Concept myConcept;
```

as well. Whenever an object like a Feature f is *added* to a collection part of another object, for example to the Feature set that belongs to a Concept c_1, then there are several integrity issues to be considered. First of all, now that the Feature belongs to c_1, the back-reference of the Feature f has to be set consistently to this new Concept. This requires to check, whether the Feature used to belong to another Concept c_2 before. Because each Feature belongs to exactly one Concept, Feature f needs to be removed from the Feature collection of Concept c_2, in this case. These kinds of considerations, and hence the update methods, are independent of the specific M4 classes involved.

For handling the updates of $1:n$ relations between all M4 classes, there is a single generic class `edu.udo.miningmart.m4.utils.InterM4Communicator` in MININGMART.

For each pair of related classes a new communicator class is derived from this abstract class. This mechanism aims to support convenient and consistent updates of references at both classes. All that needs to be filled in when deriving a specific communicator class from the abstract super-class are the references to individual getter methods, and to a "primitive" setter method that update references without considering consistency issues. In the example above, the latter method is the setter of class Feature. All further individual methods for accessing and updating relations between these classes just directly invoke convenience methods inherited from the abstract class `InterM4Communicator`.

The abstract and convenience methods of the super-class are depicted in figure 9.6, together with the original MININGMART class that handles the references between Concepts and Features. A method like `addFeature(Feature f)` in class `Concept` is implemented by just two `InterM4ConceptFeature`-method calls, one to `checkNameExists`, to avoid name clashes, and one to `add`, which adds the `Feature` to the `Collection`.

Loading database M4 objects

The next paragraphs discuss the generic mechanisms for reading requested tuples from the M4 database, and for initializing new M4 JAVA objects. As mentioned before, each M4 meta-data

[10] Please recall, that a single Concept may be linked to multiple views or tables. In this case, single BaseAttributes will also refer to multiple Columns, which requires to model both the Concept-to-Columnset and the BaseAttribute-to-Column links as $1:n$ relations.

```
/** Super-class of all inter-M4 communication classes */
public abstract class InterM4Communicator {
    // References to getter and setter of the classes:
    abstract Object getSingleRef(M4Object src);
    abstract Collection getCollection(Object src);
    abstract void setSingleRefPrimitive(M4Object m4o, Object container);

    // Convenience methods used by the referenced classes:
    public void add(Object container, M4Object m4o) { ...}
    public boolean remove(Object container, M4Object m4o) { ...}
    public void setCollectionTo(Object container, Collection coll) { ...}
    public void updateReferenceTo(M4Object m4o, Object container) { ...}
    public void checkNameExists(M4Object m4o, Object container) { ...}
}

/** Example of an extended class for Concept-to-Feature mapping */
public class InterM4ConceptFeature extends InterM4Communicator {
    public Object getSingleRef(M4Object feature) {
        return ((Feature) feature).getConcept();
    }
    public Collection getCollection(Object concept) {
        return ((Concept) concept).getFeatures();
    }
    public void setSingleRefPrimitive(M4Object feature, Object concept) {
        ((Feature) feature).primitiveSetConcept((Concept) concept);
    }
}
```

Figure 9.6.: Code for maintaining relations between M4 classes, throws clauses omitted.

table has a corresponding JAVA class which also reflects the reference to other classes. Each relevant substructure of an M4 case can hence be represented using a corresponding copy in main memory. For efficiency reasons the compiler does not pre-fetch all the meta-data of the Case under consideration, but most objects are loaded into the cache on demand. References to collections of objects, like from Concepts to their linked Columnsets, are often set to `null` until a first access to the field by an active getter method. Such active getters are used to load all the objects of a collection, and to store them internally. The first object loaded by the compiler is the specified Case or Step to be compiled, followed by Operator specifications, Step parameters, and all the required meta-data referenced during execution.

The loading procedure for *individual* objects exploits meta-data even at the JAVA level; M4 JAVA classes implement the interfaces `edu.udo.miningmart.m4.utils.M4Table` and `edu.udo.miningmart.m4.utils.M4Info`. The former allows to access all the relevant information for loading an object's data from the database. This covers the name of the M4 database table and the name of the primary key attribute holding the M4 IDs. The latter interface allows to query for complementary information, first of all for mappings between the relevant database attributes and setter/getter methods of the corresponding JAVA class. The combination of the database- and class-related specifications at all M4 JAVA classes allows the compiler core to load M4 objects using a generic mechanism. First, the tuple with the specified ID is read from the target table. Next, the objects implicitly referenced, for example Parameters of a Step that may come from several M4 tables, are collected by recursively calling the generic load mechanism, exploiting the `M4Info` knowledge about the target parameter's class. Finally, all

parameters are set using the self-reflection API. Using the `InterM4Communication` mechanism for maintenance at all intersections between M4 classes guarantees referential integrity.

A second generic mechanism allows to read *collections* of M4 objects, for example the Columnsets of a Concept. It returns all M4 objects of a specified class that reference an object under consideration. This encapsulates a more complex kind of database query than when loading single objects, but it can as well be addressed by exploiting the information provided by the `M4Table` and `M4Info` interfaces. When querying the set of Columnsets for a given Concept, for example, it is sufficient to consider the references of class `Columnset` to class `Concept`, which allows to derive the involved database tables. The condition of a corresponding SQL query is that the ID of the Concept in table COLUMNSET_T matches the ID of the given Concept[11]. The result of the SQL query is the set of the collection's object IDs. The previously described mechanism allows to fetch all of these objects sequentially, either from the cache or from the database. It just has to be taken care during the load procedure that all recursions terminate, for instance by remembering all objects currently in the process of being loaded.

To summarize, the load mechanism of MININGMART relies to a large extent on the specifications provided via interface methods. It allows to create SQL queries for finding objects and reading the corresponding tuples automatically, and it further allows to execute the corresponding setter methods via self-reflection. The result is a framework close to declarative programming, in which the specification of the database table name for a class, and the mapping of fields to getter and setter methods is sufficient to get an automatically set up JAVA representation of all M4 objects under consideration.

9.4.2. Operator initialization

The generic load mechanism described in the previous subsection is consequently used for providing all operator instances with their specified Parameters. The parts of M4 that store the static operator specifications have been discussed in subsection 9.3.3, example instantiations of operators by Steps are shown in subsection 9.3.4. The next paragraphs describe how the M4 instances are prepared for direct compiler access, and how the problematic issue of automatic feature selection is realized in MININGMART.

Aggregating parameters to loop-indexed arrays

The M4 representation of Steps uses a separate tuple for each single Parameter and loop. Hence, using the previously described methods for loading M4 objects returns one object per contained Parameter tuple. In the context of operators accessing these tuples, the aggregation of single objects to arrays according to the M4 `OpParam` specification is a reasonable simplification. Another one is to offer access to Parameters depending on the current loop number for all looped operators. Objects of type `ParameterArray` represent arrays of Parameters in appropriate form. The data structure serving as a Parameter interface between the compiler core and the single operators is implemented by the class `edu.udo.miningmart.m4.ParamDict`. Internally, a `ParamDict` data structure is populated at the first call to an active getter. It iterates through all Parameter *tuples* associated to the current Step. These tuples are directly yielded by the more general `getM4Object` methods. Comparing OpParam data types and names to types and names of Parameters suffices to construct `ParameterArrays` indexed by loop number and name. The `ParamDict` class contains methods like

[11] More complex situations like multiple links between classes or cross-tables are also supported with minor additional efforts.

```
public ParameterArray get(String parameterName, int loopNr),
```

and offers the same level of abstraction as when specifying operator parameter constraints (subsection 9.3.3)[12]. Missing Parameters are identified easily by comparing the cardinalities of all Parameter collections in the dictionary to the OpParam specifications. The dictionary is internally stored together with the meta-data for each Step.

How feature selection deselects columns

In principle, the operator specification is used by the compiler core to make sure that operators are not executed without all the specified Parameters being available. However, there is one exception to this rule, caused by a property of the feature selection group of operators. Please recall from subsection 9.2.2, that the conceptual parts of case and data model are considered static for the M4 compiler. This means, that the compiler is not supposed to change the conceptual part of the data description. The compiler even depends on knowing this part when creating and updating the relational counterparts in M4. When an automatic feature selection is applied to a previously unseen table or view, then the selected set of Features, and hence the correct relational meta-data representation is unknown until the corresponding Step is executed. This requires to design case modeling in a way that covers all possible results of such an automatic feature selection with a single conceptual data model.

In MININGMART, Features are interpreted as defining *supersets* of the attributes actually required to be supported by a Concept. With this interpretation, input Concepts are still considered to be valid if just a few of the Features are linked to Columns. As a consequence, feature selection operators just have to create outputs (Concepts) that have missing attributes at the relational level, i.e. they may just copy Columns from the input to the output Concept, while omitting those Columns that were not selected. This principle applies to `ConceptOperators` in general, and is a robust solution for missing, but conceptually foreseen Features: `ConceptOperators` in general usually copy all input Features to the output Concept. Applying the principle sketched above, missing Features may simply be ignored without any lack of consistency.

The procedure for handling missing parameters has hence been relaxed. It checks, whether a missing Parameter still allows for reasonable operator execution, and throws an exception, otherwise. Specific operator requirements that do not allow for any missing Features can be enforced by overwriting local methods, and by calling the active getter of the Parameter dictionary with stricter settings at the time of initialization.

9.4.3. Transaction management

Up to this point the M4 cache has been presented for read-only access to the database. During a normal run the compiler needs to change the dynamic parts of M4 in order to store its results. Similarly as for any other kind of potential multi-user access to databases, this requires strategies enforcing write statements from different users to be consistent.

Conflicts in multi-user databases

In relational databases it is common that each SQL session starts as a separate *transaction*. All changes are accumulated until the user either decides to *commit* the changes, which lets these changes take effect for all subsequent sessions, or to *rollback*, which discards all changes.

[12] For non-looped Parameters such getter and store methods are also available in non-looped form.

Consistency in the presence of concurrent write access is the main reason for transaction management. The goal is to allow only serializable sets of transactions in parallel, for which there exists a permutation of all transactions that induces a valid schedule. For such schedules, each individual transaction can be considered to be run as an atomic block, starting and ending in a consistent state (Eswaran et al., 1976).

In MININGMART, problems with conflicting session types are circumvented by a trivial *locking* mechanism. Whenever a user tries to access a Case in the database, a M4 meta-data table named M4_ACCESS_T is consulted. It holds the name of all Cases that are currently accessed, the name of the corresponding user, and the kind of access, one of read-only or read-write. If a Case is open for writing, then the MININGMART system denies any further access. If a Case is opened read-only, then only further read-only sessions are permitted.

The business data schema is only accessed in a way that does not bear any potential for conflicts. Existing tables and views (Concepts of type DB) are never changed. The compilation procedure just creates *additional* views, it does not change existing data. Views created by the compiler have an automatically generated name that contains the ID of the creating Step. Since these IDs are globally unique, and since each Step belongs to exactly one Case which is locked during write access, there cannot be any read/write conflict when multiple users access the same database using MININGMART the regular way.

Updating the database

Setting up a consistency framework for write access is drastically simplified by the described locking mechanism: Whenever a MININGMART user changes the database, it may be assumed that the main memory copy contains the only valid version of the represented M4 objects. This assumption is valid, because no other user will simultaneously have read or even write access to the same M4 Case.

M4 objects changed in main memory or created anew are not written back to the database before an explicit call to `updateDatabase()`, a method of the M4 interface triggered by the Human-Computer interface. This guarantees consistency of data and meta-data even in situations where the MININGMART system crashes unexpectedly, or where a compiler run does not succeed. Furthermore, it allows a user to retract all changes, simply by aborting an active session without saving. Only after successfully writing back all M4 objects from memory to the database, a commit command is sent simultaneously to *both* the M4 and business database. This prevents inconsistent states, even in the case of system crashes when writing back the JAVA cache. Whenever a write-back fails, the last consistent state will be recovered automatically by the DBMS.

Writing back *all* cached entities to the database would cause superfluous I/O costs, so there is an internal `dirty` flag for each cached M4 JAVA object. As soon as an internal field of an object is altered, which is a copy of an attribute stored in the corresponding M4 meta-data table, the dirty flag is set. If another internal field is altered, then this generally causes updates of back-references by the `InterM4Communicator` mechanism, which will result in setting the dirty flag of a referenced object, because at the database level only that other object contains the reference.

A second flag, similar to the `dirty` flag, indicates whether objects are to be removed from M4. This can for example be enforced if an attribute with a NOT NULL constraint is explicitly set to `null` at the JAVA level. To allow for an automatic handling of this kind of reasoning, the `M4Info` specification of all M4 JAVA classes contains all the required information on integrity constraints at the database level.

The method `updateDatabase()` is realized in the previously mentioned class DB, and it is invoked by the `store()` methods[13] of all JAVA M4 classes. It works similarly to the generic load mechanism, hence it also relies on the interface methods of `M4Table` and `M4Info`. All objects to be written back to the database are analyzed using the specified getter methods. This is generally sufficient to create SQL statements for updating the corresponding tuples in the M4 tables, for inserting new tuples, which get a new ID from the database sequence in this case, and for deleting the accordingly flagged objects from the tables. More complex references, for example based on cross-tables, may be realized easily by additional local methods.

Care must be taken that the insert, update, and delete statements are executed in a valid order. Please recall, that the consistency at the database level is formulated in terms of integrity constraints. Operations like deleting referenced objects, or inserting links to objects not yet inserted, violate these constraints, which causes SQL exceptions. As a first step to circumvent this problem, when invoking the delete method of an object, this object removes all of its M4 references at the JAVA level. This may remove further objects, for example ColumnStatistics no longer accessible after removing their Column, but more importantly, it allows to remove the object itself without violating any constraints after all back-references have been deleted from the database. Obviously, deleting tuples from the database is a step which should be performed *after* all dirty tuples have been updated.

The dependencies between the meta-data tables have to be respected throughout these processes, since new tuples may reference other new tuples that have not yet been inserted, and tuples not yet deleted could still reference tuples that the M4 interface would like to delete next. The reference graph between all M4 classes is static and acyclic, however, so there is a static valid order in which tuples may be inserted and updated according to their class memberships, without further checks. Analogously, the inverse of this order is used for deleting tuples. Apart from a hard-coded static order of tables that reflects their foreign key dependencies, the process of updating the database is entirely generic, just exploiting the declarative `M4Info` information.

9.4.4. Serialization

The M4 model has originally been specified in terms of a UML model. Representing M4 classes as database tables is a natural choice, particularly because MININGMART presupposes a DBMS anyway; maintaining case models in a database provides integrity checks for free and allows for efficient meta-data manipulations. There are still alternative representations of M4 Cases, one of which is the subject of the following paragraphs.

There are disadvantages of the database representation of M4 in the context of publishing and sharing Cases, for instance in a public best-practice case-base. The re-use of successful preprocessing cases at different institutes requires to move the meta-data of cases from one M4 database to another one. Addressing this task by a DBMS-specific "dump-to-file approach" makes it hard to load the same case to a different kind of DBMS. Another issue is, that the privacy policy of MININGMART permits to publish the conceptual parts of cases only, which are assumed not to contain any sensitive information regarding the relational data model of the publishing institute. Please note, that the relational meta-data is of no use in practice anyway, as it reflects a specific schema of a database not present at any other institute. Any other user will have to map the conceptual meta-data to the local relational schema. Clearly, end-users should not have to identify and dump the conceptual meta-data tables manually when publishing their Cases.

[13] The `store()` methods can be considered to be deprecated, however, since storing individual objects of a Case is no longer reasonable when using the new M4 interface.

To overcome the problems with database-only storage of M4 there is an export facility in MININGMART, which stores Cases as XML files. These files can easily be exchanged, and they can be published in a case-base. A corresponding import facility allows to import Cases into MININGMART, as well. During a regular export, all M4 objects at the relational level are simply skipped, so only the useful and non-sensitive parts of Cases are shared. However, for backup purposes there is a specific menu point in the Human-Computer interface that triggers a dump of a Case's complete M4 model to a file. This procedure works exactly as the normal export feature, it just does not filter out any of the meta-data classes. Please recall from the paragraphs on loading from and writing to the M4 database that, in principle, the M4 interface could load all the meta-data of a Case to main memory in advance, and that it stores the meta-data only after an explicit `updateDatabase()` command. Hence, it would just take some minor adjustments to the interface code to run MININGMART without an M4 database, reading and writing the meta-data to an XML file, instead. Although this proves that the concepts underlying the system do not depend on a specific kind of data management architecture, there was no demand to implement this feature so far.

The main issue when converting an M4 model to a flat file is to serialize the graph of objects in a reasonable way. A straight-forward idea is to reflect the structure of M4 classes, for example so that between opening and closing Case tags all Steps are described. This idea is not realized in the current serialization module for two reasons. First, there are often different ways to reach one kind of object from another one in M4, using different paths of links, which cannot be reflected by such structures. Second, as for handling relations between M4 classes, loading, and storing M4 objects to the database, the serialization module should be generic, that is, independent of the specific classes and links. The required information should be provided in a declarative form, rather than by implementing at each class the consequences of all of its references. The XML format actually used by the interface is flat, that is, no object embeds another one syntactically, but there are unique IDs assigned to each object, and all references are represented in terms of these IDs. To allow for an easy import, references are only allowed to objects previously described in the same XML file.

There is an interface `edu.udo.miningmart.m4.utils.XmlInfo` implemented by all the M4 classes that may be imported or exported. The specification in terms of `XmlInfo` is similar to that in terms of `M4Info`, but the information itself may be set up independently. For example, the database table name of `M4Info` is replaced by a tag name, and an array defines internal fields to be imported and exported, maps them to getter and setter methods, and specifies the M4 JAVA classes these methods expect as parameters. The interface also requires a generic getter method, in order to be able to read details about objects via self-reflection, and a generic setter method, to set up objects according to XML specifications. As for database communications, exceptions can be handled by implementing a local import and export method.

The export method as such is realized by a separate class `M4Xml`. The IDs it uses are different from the M4 IDs, basically for simplicity, since it cannot be assumed that the same IDs are valid (no clashes) at another M4 database where the case is imported. An export starts with the Case object, which does not reference any other object, but is referenced by other objects. Dependent and depending objects are considered in a fashion that reaches all objects of the Case, and that delays the step of writing an object to disk until all of its referenced objects have been written. The import is rather simple, because it is sufficient to create each object according to the specifications read from file, replacing ID references by references to the corresponding JAVA objects. This is operationalized by using a dictionary for all imported objects.

9.4.5. Garbage collection

The main objective of the M4 compiler is to create new views in the business data schema that reflect the output of each Step, and to create relational M4 meta-data, that connects these views to the conceptual part of the M4 data model. After a Case has been set up for the first time, only the Concepts of type DB and corresponding M4 Relationships are connected to business data entities. Each (valid) Concept of a different type is connected during compilation of the embedding Case. In this initial situation the compiler can simply add the additional meta-data and create views. Afterwards, the situation gets more complicated, though, because the old entities still exist. When running the compiler again now, for example after changing the specification of a Step, new views will be defined in the business data schema; these views should replace the old views. However, this could cause problems with view definitions of subsequent Steps if those rely on any of the attribute names and data types we are about to replace. This kind of consistency problem becomes even more evident when considering the relational M4 meta-data created by the compiler. After changing, for example, the specification of a feature selection Step, some previously selected Columns may suddenly become deselected. This compromises the consistency of the relational meta-data for subsequently created output Concepts.

The solution to this problem realized as part of the M4 compiler is to maintain a list of all compiler-created entities, and to remove them as required before re-compiling Steps. The objects created by the compiler are distinguished by their schema, which separates the created M4 meta-data, stored in an M4 meta-data table M4TRASH_T, from objects created in the business data schema, which are stored in a similar meta-data table called DBTRASH_T. All objects in these tables are indexed by the creating Step, hence, at the JAVA level objects of class `Step` store these "trash instances" locally. Before a Step is compiled, a *garbage collection module* collects the list of all subsequent Steps with respect to the input/output dependencies. The meta-data and business data created by the compiler are removed stepwise, inverting the order of Steps used during compilation. Finally, the procedure reaches the first Step, which can afterwards safely be recompiled. The results of preceding Steps are not affected. Because this garbage collection is performed before each compilation, no matter whether the user requested compilation of a single Step or of a complete Case, deprecated results of previous compilations are removed automatically. This also prevents an accumulation of unreachable M4 objects over time, which could otherwise easily happen for objects like statistics of "overwritten" Columnsets. As a consequence of running the garbage collection, only valid M4 entities are kept in the meta-data model.

9.4.6. Performance optimization

At its current state, MININGMART hardly optimizes runtime by allocating additional disk space. Indices are created for the results of specific operators that create views which typically suffer from massive access to a single attribute. An example is `SegmentationStratified`, which creates a view for each value of a specified attribute. Any subsequent access to one of these views inevitably conditions on this attribute, hence an index on this attribute clearly helps to reduce computational and I/O costs. Complete views are only materialized at explicit user-request. A few operators that do not allow for efficient view generation are exceptions to this rule.

The generic operator framework allows for several future extensions regarding systematic performance tuning. A straightforward next step towards efficient compilation is to represent the runtime characteristics of all operators in a well-suited formalism that allows to reason about interdependencies between steps. The expected space requirements of indices, materializations etc. often have to be traded off against compilation runtime, as presented by Hairnarayan et al.

(1996) for Data Cubes. Such an optimization is only possible with reasonable efforts when using a generic representation and a monolithic reasoning module. This module might also take several DBMS-specific characteristics into account.

Different kinds of optimization are possible and desirable for linear sequences of steps. A promising work of Gimbel et al. (2004) that fits well into this context addresses performance optimization for complete KDD processes that are run in a DBMS. The authors identify *blocking* operators, like segmentation, that slow down the overall execution, because the subsequent step cannot start before the segmentation step has been completed. For optimization purposes, sequences containing blocking operators can sometimes be replaced by equivalent non-blocking sequences. This allows to decrease the overall runtime requirements.

9.5. Code at various locations

One of the main motivations for using an abstract layer for the transformational M4 case model is that it allows to specify a Case without having to care about any DBMS- or implementation-specific details. This does not only ease the editing of preprocessing Cases, but it also allows to *change* the implementation of operators at the technical level. Operators may become more and more efficient over time, and it is very easy to continuously upgrade operators in this scheme. Even more interesting is the option of selecting the most efficient operator type by taking into account DBMS-specific properties. This section presents some examples of operators and other kinds of code that run directly *inside* a database, which clearly deviates from the JAVA framework presented in earlier sections that runs all operator outside.

9.5.1. Functions, procedures, triggers

The first version of MININGMART relied to a large extent on code implemented in a procedural programming language extension of SQL called PL/SQL. This language allows to define functions and procedures that run inside of ORACLE databases. Further, specific actions that are also implemented in PL/SQL can be triggered by events like *insert into* or *update* to specified tables. This mechanism is simply referred to as *triggers*.

An earlier version of the M4 interface, specified and implemented by the external MININGMART partner Perot Systems Netherlands, used to check constraints and enforce integrity to a large degree directly inside the database by triggers, functions, and procedures. Validity checking for objects was realized by automatically deriving annotations in the form of boolean attributes for meta-data entities in the database. One kind of information maintained this way was whether all required parent and child objects were present. Integrity was mainly enforced by mechanisms similar to (but more complex than) cascaded deletes; unreachable M4 objects were automatically deleted after deleting the parent object.

As described in section 9.4, in later versions of MININGMART these update mechanisms have all been moved to the JAVA M4 interface, simply because the PL/SQL code turned out to be slower by several orders of magnitude, which delayed compilation times so drastically that the system became practically useless. Although turning off transaction management and similar DBMS services might have helped to circumvent this problem, it would have drastically reduced the benefits gained by storing meta-data in databases.

Hence, the M4 interface of the compiler was extended to replace the old interface, and the PL/SQL interface code was replaced by JAVA code running outside the database. Similarly, in earlier versions all statistic computations were performed by PL/SQL code inside the database.

This turned out to be slower than an optimized JAVA re-implementation, connecting to the database via JDBC.

These counter-intuitive results illustrate, why the current MININGMART system uses only very few PL/SQL functions and procedures. Additionally, the JAVA code allows for easier migration to other kinds of DBMSs, since PL/SQL-like code is inherently DBMS-specific. The successful re-implementation of the interface and of several procedures and functions in JAVA, without any need to change M4 or stored Cases, illustrates that MININGMART's abstraction from the implementational level is sound.

Examples of PL/SQL-code still in use include functions that allow for a convenient embedding into view definitions. There is an operator `TimeIntervalManualDisretization`, for example, which was implemented by the external MININGMART partner National Institute of Telecommunications, Warsaw. This operator discretizes the date and time value of a given attribute with respect to a user-defined mapping. There is a PL/SQL-function that compares each input value to the values of a small helper table containing all the intervals, and it outputs the corresponding interval label. The main advantage of using this function is, that it conveniently and efficiently allows to process the target attribute value of each tuple from inside a database view definition. Corresponding SQL statements defining an output_attrib calling a function for target attribute A1 are of the form

SELECT A1, A2, . . . , function(A1) AS output_attrib FROM . . .

To embed the complete output attribute definition into a SELECT statement is considerably more complex. As an advantage, compared to processing the statement outside of the database, there is no need to e.g., struggle with the complex date and time format conventions of different DBMSs. The SQL support for such data types allows to simply compare the interval boundaries to the target attribute values by employing arithmetic operators.

As a disadvantage, it is still necessary to adapt each PL/SQL function to each supported DBMS, and make sure that only supported functions are selected and invoked by all operator implementations.

9.5.2. Operators based on Java stored procedures

The code discussed in the previous subsection contained only helper functions and methods. Apart from efficiency issues, this is due to the fact that many programmers find programming in PL/SQL more tedious than programming in languages like JAVA. A specific ORACLE feature allows to store procedures implemented in JAVA in the database. They are loaded by a specific ORACLE tool, and can be executed indirectly by calls to an embedding PL/SQL procedure that specifies all parameters. The JAVA code communicates with the DBMS via a JDBC interface, which is very similar to using this interface from outside the database. The minor modifications to JAVA code required to run operators inside the database motivated a framework, in which some operators query a method

```
protected boolean storedProceduresAvailable()
```

of `ExecutableOperator` to decide whether to run the code inside of the database, or to run the same code the common way, as a JAVA operator that connects to the DBMS via the general interface provided by class `DB`. This is done by all classes derived from `TimeOperator`, i.e. `ConceptOperators` running over time series, and aggregating several input tuples to a single output tuple. This kind of functionality can hardly be provided efficiently based on view definitions, so at the relational level the output is a materialized table for all of these operators.

Currently the list contains `Windowing`, an operator that stores in n distinct attributes of each tuple the n most recent values of a single target attribute of the input, `Simple/Weighted/ExponentialMovingFunction`, three operators that average input values of the last tuples by different weighting schemes to define a single output value, and `SignalToSymbolPreprocessing`, which aggregates each interval with similar values of a target attribute to a single tuple. In several experiments, running these operators as JAVA stored procedures reduced the runtime significantly, probably because much of the communication overhead between database and compiler was avoided. Running these operators outside the database requires to transfer the complete input and output table via JDBC. The current framework for these operators when run as stored procedures, and the operator implementations as such, have both been implemented by the author of this thesis.

As a final example of an operator running inside of a database, Rüping (2002) reimplemented his original MYSVM support vector machine (Rüping, 2000) in JAVA. One variant can be executed as a JAVA stored procedure in ORACLE databases. In contrast to the previously described operators, the author did not report a reduction of computational costs, but it may still be attractive for several applications not to transfer any of the (potentially sensitive) data to another computer, but to run all computations locally on the database server. The corresponding MININGMART operators are basically just wrappers. It is still interesting to note, that for the same learning operator system-dependent native code implementations based on flat-file input are available in MININGMART. This illustrates once again, that the abstraction from executables which is part of M4 is both sound and useful. The decision of how to execute Steps can be made based on arbitrary application- and system-dependent details. The related issue of embedding operators that are available as native code only is discussed in the next subsection.

9.5.3. Wrappers for platform-dependent operators

Many machine learning tools are available only as stand-alone executables. If runtime is a critical issue, then optimized programs written in languages like C may be an attractive alternative to platform-independent solutions like the YALE toolbox. The MININGMART compiler exploits a variety of learning operators that are available in native code, all of which but the last one are embedded in terms of a wrapper.

Apriori This operator for frequent itemset and association rule mining has been implemented by Bart Goethals[14]. It reads a flat file containing all transactions into main memory. Four different input formats are supported. The operator outputs all itemsets that are more frequent than a user-specified threshold.

C4.5 As a decision tree learner for discrete and continuous predictor variables, the implementation of Quinlan (1993) is used. It reads the training data from a flat file and outputs a tree in ASCII representation.

KMeans This clustering algorithm by the MININGMART partner DISTA has been extended to allow for constraint specification in a semi-supervised clustering framework. For a description please refer to (Saitta et al., 2000).

mySVM This is the original implementation of the support vector machine by Rüping (2000), which has later been adapted to databases. The platform-dependent executable file (or *binary*) of this support vector machine for learning reads examples from a file or from

[14] `http://www.adrem.ua.ac.be/~goethals/software/`

standard input, respectively, one example per line. The output is a support vector model that can be stored and applied to unclassified data sets by another binary. In MININGMART, this learner can be applied to classification and regression tasks.

SubgroupMining This operator for subgroup discovery has been developed by the MININGMART partner Fraunhofer, AiS institute. Unlike the other operators, it is written in JAVA, which allows for an easier integration into MININGMART. It is only listed here, because it still is another *external* learner, which is integrated into the compiler by adapter classes.

The following paragraphs exemplarily describe a MININGMART wrapper for the support vector machine. In this and all other cases, there is a regular JAVA operator that acts as an interface between the JAVA compiler framework and the native code. The considered operator uses a support vector machine to replace missing values by predicted ones. It is a regular `FeatureConstruction` operator called `MissingValuesWithRegressionSVM`. It has an additional list of SVM-specific parameters, and offers to specify a subsample size for training. If a sample is requested, then a class for generic subsample construction is instantiated, which creates a temporary table, just containing the randomly selected row numbers, and an output table, which is a materialized join of the original table with the temporary table. The binary is executed in a separate JAVA thread, using the `exec` mechanism of JAVA `Runtime`. There is a separate binary for all supported platforms, SunOS, Linux, and Windows. All user-specified parameters are provided at initialization time of the thread. The training data is read from the Columnset linked to TheInputConcept, respecting the projection defined by the parameter ThePredictingAttributes. The data types are converted automatically where possible, and are written in appropriate form to a pipe that serves as an input to the SVM thread. After the SVM has finally processed the data, it writes the induced model to a specific place inside the local MININGMART directory, which has also been specified when initializing the SVM. The model file is then parsed by the wrapper, which basically means to extract all support vectors. These vectors are written to an intermediate helper table in the business data schema. Additionally, a function is defined in the same schema, which iterates through this helper table of support vectors, and combines them in accordance with the selected kernel type in order to predict a value. By embedding a call to this function into the SELECT part of an SQL view definition, the target attribute is replaced by a version without missing values in the newly created view. Registering the new business schema entities at the relational and conceptual level of M4 completes the job of the operator.

The only wrapper implemented completely by the author of this thesis is the one for APRIORI. It is not discussed in detail, since frequent itemset mining is a typical data mining step, which is a bit out of the scope of the MININGMART system; it makes less sense to consider frequent itemsets during preprocessing. The transactions are read from TheInputConcept and written to a flat file. Sampling is supported the same way as for the SVM wrapper. As a next step, the APRIORI binary for the current platform is executed as a separate thread, and the resulting association rules are written to a database table in a pre-defined format. These sets can be used as data mining results. The table is finally represented by a Columnset, which is connected to the output Concept.

9.6. The interface to learning toolboxes

The focus of the MININGMART system are the various preprocessing phases that are necessary to transform the raw data available at real-world databases and data warehouses into a format

that allows to apply data mining tools. One of the main characteristics of MININGMART is that all results are represented as parts of the (data) ontology. In contrast, patterns, decision trees, and support vector models may establish a basis for data transformations performed by operators, but there are no natural mechanisms to inspect or store such models in MININGMART, to reload and apply them to unseen data, or to validate them by standard techniques like cross-validation. Such objectives are addressed by standard data mining toolboxes, and – as will be illustrated in this section – there is no need to re-implement this functionality in MININGMART, because there is a natural interface to these systems. The system YALE[15] (see p. 190 or Mierswa et al. (2006)) is used for illustration at this point.

9.6.1. Preparing the data mining step

Considering supervised learning tasks, each example is represented by a single line of a flat input file in YALE, which represents a vector of fixed dimensionality. In contrast to the ontology and relational data model of MININGMART, the flat representation of YALE example tables does not allow to reflect any references between objects or any other kind of higher-order structure. If the data mining task at hand does not inherently depend on structured data, then a phase of propositionalization of all multi-relational data is possible during the preprocessing phase with MININGMART. In any case, in order to ease the access to any preprocessing results, the operator `Materialize` should be applied to all resulting views. This operator creates a materialized table with a given name, which speeds up the reading procedure, on the one hand, and allows to reference the table using an intuitive name, on the other. For real-world databases, the table size will often exceed reasonable sizes for execution by YALE in main memory. An easy solution in such a case is to draw uniform subsamples for training models. A MININGMART operator for this purpose is `RowSelectionByRandomSampling`. More complex and powerful alternatives have been discussed in depth in previous chapters of this work.

YALE supports reading from database tables. It is necessary to specify some details, like the host, the name and the port of the database, the table name, or a complete SQL query. YALE experiments can be run as usual after a single materialized table of tractable size has been created by MININGMART. All that is necessary is to replace the YALE operator for reading from flat-files with an operator that reads a (single) table from the database. That way, one can induce a model with YALE from the output of MININGMART and store this model in file format. The MININGMART operator `PrepareForYale` sets up the framework for a YALE experiment based on a specified MININGMART view for data mining.

9.6.2. Deploying models

A user may want to perform the data mining step with an external tool, after the preprocessing has been done in MININGMART. A typical application is to induce a predictive model for a supervised learning task. If the induced model shall be applied to previously unseen data during the application phase, then the target data table might not fit into main memory, and the predictions are usually not made persistent in appropriate form by main memory learning toolboxes. Hence, users may prefer to have the results of applying their models also stored in the database, which is most flexible with respect to further applications and feasible even for large-scale data sets.

As discussed, it is possible to write a model to a flat file with YALE. There is an operator in MININGMART that allows to apply a flat-file model, supplied as an operator parameter, to a view or database. The main constraint is, that the schema used for learning matches

[15] `http://yale.sf.net/`

the schema used at application time, except for the target attribute to be predicted. A further, rather technical constraint is that the target table must have a primary key. The operator is called `YaleModelApplier`, and it is a `ConceptOperator`. It relies on several service methods of the YALE core, which can directly be accessed, since YALE is also implemented in JAVA. All models available in YALE are serialized, and they may be applied to data sets after loading them with the YALE core. To this end, it is sufficient to import a single `jar`-file. The source table is read into main memory block-wise, which permits to process even very large database tables. In order to relate the new predictions to the existing data tuples, an intermediate table is created by the operator. It just contains the primary key of the source table, and the new target attribute to be created. The type of the target attribute can be derived from the conceptual data type and the type stored inside the YALE model. The model is applied to each block and writes the primary key and the prediction for each corresponding tuple to the intermediate table. After the complete source table has been processed, the operator creates a view that extends the source table by a column holding the predictions. This is done by joining the source with the intermediate table. This operator is loopable, which allows to apply a set of models to the source table, all of which are finally joined. The view containing the predictions is created in the business data schema, registered as part of the M4 relational model, and linked to the output Concept.

This illustrates, that it is easily possible to "source out" only the data mining step to one of the commonly used main memory learning toolboxes, but to still do any preprocessing and model application steps in MININGMART. The natural interface offered by the MININGMART system are single materialized tables of appropriate size for learning. This representation is supported by most of the available data mining environments.

10. Conclusions

Knowledge discovery in databases is a field of highest potential. The impact of finding novel, unexpected, and potentially useful patterns on business domains is high, but so are the technical and theoretical efforts required to solve the problems convincingly well.

10.1. Principled approaches to KDD – theory and practice

The initial problems addressed by KDD are *business problems*, stated in business terminology and with solutions being assessed in terms of business-related criteria, e.g., return of investment or customer satisfaction. Mapping these problems to formal *data mining problems* is not trivial a task; the well known tasks do not always contain a precise formal counterpart to a business problem at hand, so the set of data mining tasks discussed in the literature is constantly being augmented by new variants. The techniques used to address these formal problems are usually based on assumptions stemming from theoretical models underlying the formal data mining scenarios. As an important example, most learning algorithms, sampling techniques, and the different PAC learning frameworks assume the data to be sampled i.i.d.

It is desirable to address data mining tasks in a principled fashion. Referring to theoretical frameworks helps to decouple data mining techniques from the specific problems they were developed for, and thereby allows to identify a reusable set of basic methods that can be *proven* to perform well in terms of formal criteria. This leaves us with the problem to understand the relations between the formal criteria, and between criteria and formal tasks, respectively. In this thesis, very general novel theoretical results were derived by augmenting the existing theory. On the one hand, these results help to foster a better understanding of the nature of existing and novel tasks, and to increase the overall transparency of data mining in general. On the other hand, the results constitute the well-based foundations of effective novel techniques that were derived for a variety of different tasks. These techniques were designed to meet two constraints, to (i) allow for guarantees regarding the results, and (ii) to be as general as possible. Examples of the latter include a consequent preference for black-box approaches throughout this work, and constructive illustrations of how techniques generalize to different utility metrics.

At the practical side, additional constraints must be considered. In this thesis, the applicability of novel methods to the large-scale data sets found in practice was considered to be a hard constraint, so for all proposed techniques the runtime complexities were shown to meet the practical demands. Providing practical support for end-users also involves the development of systems that apply to the challenging real-world KDD tasks. In the past, machine learning techniques were usually implemented as stand-alone operators and pre-processing was e.g., performed by manually entering SQL statements. As a result, practitioners met with many technical burdens, and pre-processing became a bottleneck of KDD. In contrast, the novel algorithms and approaches presented in this work are all available as parts of broader open source KDD toolboxes that allow to transparently organize complete KDD applications in terms of operational meta-data.

10.2. Contributions

The following section summarizes the contributions presented in this thesis in more detail. Following the argument above, it is structured from abstract to operational.

10.2.1. Theoretical foundations

This thesis extended the existing data mining theory by *combining* statistics and PAC learning theory, both well-suited to give probabilistic guarantees on data mining results, with the ROC analysis framework for analyzing the behavior of different utility functions in data mining contexts, and, moreover, a corresponding decomposition of utility functions, e.g., into BIAS and COV. Only this combination covered all aspects relevant to this thesis in a way that allowed for the desirable general analysis. The theoretical results that were derived by extending the theoretical framework are summarized in subsequent sections.

The statistical framework that was built upon in this work is similar to the agnostic PAC model, because usually (i) no fixed target concept class may be assumed, (ii) there is no deterministic dependency between labels and feature vectors, and (iii) there are different kinds of noise in practice. Samples are assumed to be independently and identically distributed in the agnostic PAC model, an assumption shared by most sub-sampling approaches and data mining algorithms. This framework allows to provide guarantees for a large class of problems, because it is not based on unrealistic assumptions. As a general technique repeatedly applied in this work, transformations of distributions underlying the data were explicitly defined. Specifically tailored transformations have been shown to be beneficial for many different data mining problems, and to provide an effective way of analyzing the behavior of data mining algorithms. They can be realized by Monte Carlo techniques like rejection sampling. The decomposition of utility metrics turned out to be very effective to derive results that apply to many different tasks.

General analysis of evaluation metrics

At many points in this thesis, the properties of specific evaluation metrics were analyzed in order to gain a better understanding of the properties of learning strategies. These abstract findings have very general implications.

As a first novel result, for known class priors the weighted relative accuracy metric (WRACC) was shown to meet the definition of an instance averaging function, and to share the confidence bounds of predictive accuracy (ACC). This connection allows to compute simpler and tighter confidence bounds in an adaptive sampling framework. If class priors are unknown, then it is cheap to get precise estimates from large data sets.

This result was complemented by a direct reduction of the task of subgroup discovery with the most common utility function WRACC to the better supported task of classifier induction, aiming to maximize ACC. The original preference ordering of rules induced by the former is identical to the ordering induced by ACC after a step of stratification. In other words, WRACC-based subgroup discovery can as well be solved by rule induction algorithms optimizing predictive accuracy after a preprocessing step of stratification, so it is not necessary to adapt rule discovery techniques to WRACC maximization.

A similar, but less general connection has been shown to hold between maximizing the area under the ROC curve (AUC), a ranking metric, and maximizing WRACC, or maximizing ACC after stratification, respectively. For boolean decision trees (i) a stratification at each leaf before selecting the most accurate split or (ii) selecting splits that maximize WRACC greedily maximizes the AUC. More generally, AUC has also been shown to be maximized indirectly when

aiming to maximize ACC in the context of boosting. This result has been derived for the family of REAL ADABOOST-like learners.

Results presented from the literature illustrate that class skews and class-dependent misclassification costs can be combined to a single kind of skew, having a unique slope in ROC space. In turn, data with class skew can be considered to implicitly change the class-dependent costs of the optimization problem addressed by classification techniques.

In combination, the results above, all related to the prior class distributions, foster a better understanding of the role of this kind of skew in data mining; this is a prerequisite for utilizing approaches that change the class priors, for example by stratifying the data.

Analysis of evaluation metrics for distributed data mining

Evaluation criteria were also analyzed in the context of distributed supervised learning. Compared to sub-sampling uniformly, the complexity of identifying a set of best rules is higher if the data is distributed. The main result is that, without very strong assumptions, the sets of locally and globally best rules may differ drastically. This result holds for all utility functions that are monotone in coverage and bias. Hence, in the general case, subgroup discovery can not be solved approximately, in the sense of the PAC-like approximately k-best rules problem, by focusing on the k locally best rules at all sites and evaluating them globally. The set of all locally best rules may be completely disjoint from the globally best rules. Novel bounds were derived for a more restricted class of utility functions, including WRACC and the binomial test function, for the case in which strong assumptions regarding the pdfs underlying the data at all sites are appropriate. These bounds allow to translate local into global rule utilities with bounded uncertainty by exploiting specific properties of the considered class of functions.

Finally, the evaluation metric of the novel task of distributed relative local subgroup discovery (discussed below) was analyzed. It scores subgroups by their deviations of local from global utilities and was shown to be at least as complex as global subgroup discovery.

Optimistic scoring functions

In combination with a refinement operator that extends rule bodies during a search for interesting rules by one literal at a time, it is important for efficient pruning strategies to be able to compute optimistic scores. Such scores are upper-bounds of utility scores that are required to hold for each refinement of a rule under consideration. If in a data mining context only rules exceeding any given utility score threshold are relevant, then the refinements of rules with optimistic scores below this threshold do not have to be considered. Loose optimistic scores increase the number of candidates that have to be evaluated, because they do not prune optimally. Optimistic scores depend on the specific choice of a utility function.

A tight upper bound was derived for global distributed subgroup discovery. This bound applies in combination with partial counts, that is, if counts for a rule are available only for a subset of all nodes, but counts for more general rules are given for the remaining nodes. This is a desirable property in distributed data mining. The derived optimistic scoring function is tighter than the one used by the original MIDOS algorithm. Its correctness was shown explicitly only for the WRACC metric, but as discussed, similar optimistic scoring functions can be derived for a larger family of utility functions that share a monotonicity constraint which is reasonable for utility functions in general.

For the utility function used by the relative local subgroup discovery task, a different tight bound (optimistic score) was derived, also capable of exploiting partial counts.

ROC analysis of weighting schemes

ROC analysis is a flexible tool for analyzing evaluation metrics and soft classifier performances. It allows for simple illustrations of e.g., how to compensate varying class skew, or how to incorporate asymmetric misclassification costs. In this work, the first illustration of a boosting algorithm in ROC spaces was presented. This integrates boosting into the ROC framework, allowing to conveniently perform the kind of analysis above. In particular, the difference between potential selection metrics to be used by base classifiers may be studied in more detail, as successfully done for rule selection metrics in the literature.

It was shown that the progress of the novel stratification-based boosting technique ADA^2-BOOST can naturally be visualized in nested coverage spaces, just like separate-and-conquer rule induction algorithms. The latter remove "explained" subsets after each iteration of inducing an additional rule. Boosting can be considered to discard examples probabilistically, which results in coverage spaces shrinking in a comparative fashion. Implicitly, the base classifiers utilized by ADA^{2}BOOST maximize the WRACC metric, because the data is stratified. WRACC is a subgroup discovery metric used to identify interesting rules. In this light, the similarity between boosting and sequential subgroup discovery discussed below is not surprising.

A novel and tighter AUC bound for the class of REAL ADABOOST-like boosting algorithms was derived by a simple and intuitive ROC space analysis. Apart from the specific result, the visualization and the connection to stratification that was pointed out foster a better understanding of the nature of boosting algorithms and the potential of stratification for data mining techniques. Exploiting the novel results, ROC analysis has become a promising approach for developing novel ensemble methods in the future.

10.2.2. Novel data mining tasks and methods

Many of the aspects of KDD motivated in the introduction are not sufficiently well reflected by existing formal data mining tasks, or are not supported optimally by existing techniques. The proposed adaptations of tasks and methods will be summarized in this section. The methods are based on the theoretical findings summarized in the last section.

We start with the proposed generalization of subgroup discovery, which is then adapted to different other settings. The goal of subgroup discovery is to find interesting subsets of a classified example set. The search is guided by a utility function, trading the size of subsets (coverage) against their statistical unusualness. By choosing the utility function accordingly, subgroup discovery is well suited to find "interesting" rules, e.g., with smaller coverage and higher bias than directly supported by standard classifier induction algorithms. The result of the data mining step is a set of understandable rules characterizing a target variable.

Subgroup discovery is a good starting point for descriptive tasks, because it handles the utility function to be optimized as a parameter. The most typical model class utilized in this context are classification rules. Clearly, this leads to interpretable findings in descriptive analysis tasks. However, except for the distributed analysis (chapter 7) the results derived in this thesis do not rely on any assumptions regarding the syntactical form of models. The step of finding a model that optimizes a utility function was consequently assumed to be handled by an arbitrary learning algorithm in a black-box fashion. This implies that the results basically apply to arbitrary model classes.

Adapting the task of subgroup discovery

A major shortcoming of subgroup discovery addressed in this work is that the definition of the task only partially reflects the intuitive concepts of interestingness and unexpectedness. Both of these concepts can usually only be stated with respect to prior knowledge; any finding might exactly match the expectation of a user, which makes it far less interesting. Without incorporating prior knowledge into the formal data mining task, the task will often be solved correctly by reporting known and irrelevant patterns. The incorporation of prior knowledge is not supported by existing methods. However, the BIAS used by many utility functions has the effect of mining patterns relative to the class priors. This idea can be generalized: If more precise forms of prior knowledge than class priors become available, then the utility functions should be made sensitive to this new information in order to mine patterns relatively to any expected class distributions.

This novel idea has been realized taking a two-step approach, a first step of specifying and analyzing the goals more precisely at a theoretical level, and a second one of realizing this idea technically. The goal was to support the incorporation of prior knowledge as a preprocessing step, so that any induction technique could be applied subsequently. As a second constraint, the method was supposed to scale to very large databases. Sampling-based techniques are a natural choice in this case. The resulting samples should no longer support the prior knowledge, but the remaining patterns should still be observable in the data. A corresponding specification at a theoretical level was derived based on a set of intuitive constraints. The goal of the constraints was to narrow down the choice of a probability density function to sample from. In fact, the constraints were shown to uniquely induce a target density function which can conveniently be written in closed form. By construction, for this density function the class label is independent of the predictions that can be derived from the prior knowledge. At the same time, the new density function is as close as possible to the original function. As a consequence, any subsequently applied data mining algorithm will focus on novel patterns, because the patterns that have already been formalized are no longer observable in the data.

Knowledge-based rejection sampling

Sampling from the probability density function defined above is referred to as *knowledge-based sampling* in this thesis. A novel algorithm to realize this kind of sampling has been proposed and analyzed. It resamples directly from a database or utilizes a provided procedure that operationalizes a step of i.i.d. sampling from any fixed distribution, respectively. The algorithm takes a rejection sampling-like approach, which, despite its simplicity, allows to "sample out" correlations between prior knowledge and class labels exactly. This means that the algorithm precisely samples from the specified target density function, although this function is defined with respect to (i) the true but unknown performance of the prior knowledge and (ii) the unknown density function underlying the original data. Decision trees, classification rules, and crisp base classifiers in general are well supported representation languages to formulate prior knowledge in this context.

The correctness of the knowledge-based rejection sampling algorithm was formally shown, and its runtime and sample complexity were analyzed. An important result is that the algorithm allows for large-scale applications. The novel kind of sampling can easily be integrated with adaptive sampling to allow for PAC-like guarantees, and with other techniques based on rejection-sampling, e.g., incorporating example-dependent misclassification costs. It may be combined with many different utility functions, e.g., with the binomial test function to favor significant subgroups, and it applies to a broad variety of supervised learning tasks where mining relative to prior knowledge is desirable.

Sequential subgroup discovery and predictive subgroup ensembles

The knowledge-based sampling framework yields intuitive distributions and allows to apply common data mining tools in a black-box fashion. It was even shown to be capable of operationalizing an additional refinement of the subgroup discovery task. Using common approaches, in many of the reported subgroups the same literals are often observed in different combinations. This kind of redundancy is the result of the task definition: Subgroups are just required to be intensionally different, but are allowed to be extensionally similar or even identical. This motivated the novel task of finding diverse sets of subgroups. The link to knowledge-based sampling is the observation that each time a new pattern is identified it can be assumed to refine the prior knowledge. The generic knowledge-based sampling strategy allows to turn pattern mining into an iterative process. In each iteration, one unexpected pattern is identified and the target density function is refined to no longer support the pattern, which shifts the focus of the data mining technique towards those patterns that are unexpected with respect to (i) the prior knowledge and (ii) all previously discovered patterns. The goal of sequential subgroup discovery is to report a small diverse set of interpretable rules that – as a set – characterize the unexpected aspects of a specified property of interest.

Due to the sampling-based nature of knowledge-based sampling, a sequential application of this technique scales well to large-scale data sets. The facts that each new transformation of the density function can exactly be operationalized, e.g., by knowledge-based rejection sampling, and that the functions are defined with respect to the unknown true performances of prior knowledge and discovered patterns, have a further advantage. All performance estimates can easily be refined based on subsequently read data. Such refinements neither affect the definitions of subsequent density functions nor the true or estimated performances of models induced based on these density functions. This is a desirable property in data streaming environments. For example, the estimated conditional class distributions at the leaves of a decision tree may be too optimistic when estimates are based on the training data. Refining the estimates based on subsequently read data has no effect on any distribution or other induced rule or model.

In cases where sampling is not appropriate, e.g., because the data easily fits into main memory, the novel sequential subgroup discovery technique can as well be operationalized by introducing example weights.

In any case, sequentially identifying and characterizing deviations between prior knowledge, including previously discovered rules, and the true distribution of the target label allows to efficiently construct ensembles for predictive purposes in the next step. It was shown that the sequential transformation strategy of density functions as defined by the novel subgroup discovery technique naturally corresponds to a NAÏVEBAYES-like combination of models for making predictions. Combining models this way does not cause any computational overhead. Technically seen, this combination works by computing odds ratio estimates by multiplying the LIFT ratios of models with the odds ratio of the complete data set. Besides being computationally efficient, this strategy has another advantage: the LIFT ratio estimates of models are invariant to class skews; the skew just needs to be estimated once for the complete data set. Even this step can be avoided, by exploiting the theoretical result that subgroup discovery with WRACC can be reduced to classifier induction by stratifying the data. This has been exploited to derive the very general algorithm KBS-SD; it allows to substitute any classifier induction technique in a black-box fashion.

KBS-SD was empirically evaluated on several benchmark datasets in combination with a classification rule induction algorithm, to asses both its descriptive and predictive performance. It was shown to outperform existing techniques with respect to robustness, ranking performance,

monotonicity of learning curves, diversity, and the number of rules required until convergence. The lower average coverage and WRAcc of resulting rule sets confirmed the intuition that the method is capable of focusing on smaller deviations between expectation and data set when all larger subgroups have been discovered.

Boosting crisp base classifiers based on stratification

Boosting is one of the most popular learning strategies for predictive learning tasks in practice. Although the original goal of knowledge-based sampling is different, it also allows to boost "weak" classifiers if applied sequentially. This connection bridges the gap between descriptive and predictive learning tasks. The performance of a marginally altered version of KBS-SD, when used as a boosting procedure, was analyzed in this work. In a first step, the predictive performance of KBS-SD-like algorithms was indirectly explained by pointing out their similarity to the well known AdaBoost algorithm. Moreover, it was shown that a variant, referred to as Ada^{2}Boost, simplifies and improves AdaBoost at the same time, while sharing several aspects of confidence-rated boosting techniques like Real AdaBoost.

The main difference to AdaBoost is, that Ada^{2}Boost takes more advantage of its base classifiers, which usually improves the learning rate and final accuracy of resulting ensembles. This gain has been shown in theory and it was confirmed empirically. Ada^{2}Boost simplifies AdaBoost, because it just stratifies each subset for which the most recent base classifier makes the same prediction, referring to a single estimate per subset, namely the odds ratio. Just like KBS-SD, the final odds ratio prediction is based on the product of the odds ratio of the example set and the Lift ratios corresponding to the predictions made by the base classifiers. The latter simplify to odds ratios, due to the stratification property, so the odds ratios are the only estimates required for both reweighting and making predictions. It is worth noting that the models can be rewritten so that Ada^{2}Boost uses the same class of linear base classifier combinations as AdaBoost, up to a single additive constant.

Confidence-rated boosting algorithms are capable of incorporating confidence scores of their base classifiers. Depending on the base classifier, such scores are not necessarily well calibrated, i.e. they may not reflect the class priors well. In contrast, Ada^{2}Boost supports boolean crisp classifiers, and it implicitly adds confidence scores as part of the boosting procedure; these scores match the predictive strength of the corresponding base classifiers and preserve the class priors. The fact that the ensembles can finally be rewritten as AdaBoost models illustrates that confidence scores are introduced only in a very moderate form, which prevents overfitting.

As discussed above, the coverage space analysis of Ada^{2}Boost helped to point out more general properties of boosting techniques. In particular, it justified the technique of layered stratification for predictive data mining algorithms, the common foundation of KBS-SD and Ada^2-Boost. It suggests to "sample out" all identified correlations between base models and the target class until reaching convergence, a distribution for which no model reveals any information about the label. It was shown that this strategy increases predictive accuracy, the area under the ROC curve metric, and estimates for conditional class distributions better than AdaBoost. Naturally, the invariance to skewed classes reported for KBS-SD also holds for Ada^{2}Boost, that is, the performance estimates of models used to make predictions are assessed relative to any given class skew. It was shown that this allows to skew classes artificially, which is a straightforward strategy to introduce a confidence threshold the base learner must overcome to improve over the default hypothesis.

Boosting from data streams with concept drift

In many domains, new data becomes available continuously for inducing and refining predictive models. A practically highly relevant question is how to adapt data mining techniques to cases in which the i.i.d. assumption is unrealistic, e.g., because the user or system behavior to be predicted may change over time. This should not compromise the performance of the data mining technique if the distributions underlying the data are stationary.

It was shown that the knowledge-based sampling framework naturally adapts to this scenario. The main observation is that KBS is able to decompose any distribution into a correctly predicted and an independent component. A first step to adapt KBS to data streams was to test in each iteration whether adding a new model or refining the latest model yields a better predictive performance. To account for drifting concepts, the algorithm was changed in a second step, so that all base model weights are continuously re-estimated on the latest available data.

The resulting novel $\text{KBS}_{\text{stream}}$ algorithm is a generic boosting algorithm that (i) can be used in combination with any base learner, that (ii) automatically determines when to stop growing a model and when to add a new one, and that (iii) adjusts its base model weights to any changes in the underlying distribution. If concepts change slowly, then this algorithm exploits the fact that it is able to incorporate prior knowledge (the previous model), to characterize just the new target distribution not yet fully reflected by the training data. This allows for a quick adaptation. Sudden changes in the target distribution are addressed by continuously re-estimating the performances of all ensemble members. The novel boosting procedure was empirically shown to outperform approaches ignoring concept drift, and to be competitive to computationally much more expensive approaches that cannot handle huge streams and are hence practically irrelevant for many real-world applications. An interesting aspect is that only the latest model is refined and all other models are "frozen" after convergence. This allows to quantify the drifting rate of concepts in terms of weights assigned to the base models.

Distributed subgroup discovery

Another important aspect of real-world data mining techniques is to support the distributed nature of many databases. For the important task of subgroup discovery no distributed approaches have yet been proposed. The theoretical results discussed above show that the problem may not be addressed by local search strategies and by relying on similarities between locally and globally best subgroups, unless very strong distributional assumptions can be made. Further, existing distributed algorithms have been shown not to be appropriate for the task. In particular, subgroup discovery lacks the strong monotonicity exploited by distributed algorithms for frequent itemset and association rule mining, so it is more efficient to address this task by different techniques tailored towards the specific properties of utility functions to be optimized.

A novel distributed subgroup discovery algorithm based on exhaustive distributed search was proposed and analyzed. It exploits the optimistic scoring function discussed above, which computes a tight upper-bound on the best possible refinement of each rule and allows for pruning based on partial counts. The communication costs of the algorithm are linear in the number of nodes and evaluated rule candidates. Compared to non-distributed exhaustive search, the total computational costs are even reduced due to the parallel search that evaluates the same candidates: If the data is distributed to different nodes then the aggregated super-linear query costs imposed by the index structures of database management systems are lower than when mining from a single database containing all the examples. Distributed subgroup discovery can be combined with knowledge-based sampling in a straightforward manner. In this case, each example

weight (or probability to be sub-sampled) will be identical to the weights in the non-distributed case after each iteration.

As a final novel task, relative local subgroup discovery was proposed. This task aims to identify subgroups that are interesting because they score much higher with respect to a utility function at any specific node than in the aggregated global data set. The optimistic scoring function above, offering capabilities to prune based on partial counts, is utilized by an adaptation of the distributed subgroup discovery algorithm above. Although this task is more complex, the communication costs of the algorithm still scale linearly with the number of nodes and evaluated rule candidates.

Both novel subgroup discovery algorithms yield exact solutions and are the first algorithms addressing their corresponding novel data mining tasks.

10.2.3. Practical support by specific KDD environments

Desirable features of KDD environments

In subsection 8.3, an overview of different tools specifically designed for KDD applications was given. While the data mining step is well supported, this is not yet the case for earlier phases of the KDD process, including data understanding and preprocessing. The latter phase is often still tackled by manually entering SQL-statements, or programming PERL scripts, which is highly prone to various kinds of errors. Clearly, solutions realized in such a fashion can hardly be re-used when addressing similar tasks in the future.

An obstacle to data understanding are the representations (schemas) used in database systems, which are typically designed by technicians. They are usually optimized for efficient access and specific applications (other than data mining), but e.g., lack intuitive table and attribute names that directly refer to real-world entity classes. Terminological problems are explicitly mentioned in the CRISP-DM model, which even suggests to set up domain dictionaries. Domain and data understanding should at least be supported by introducing a more natural representation based on domain-dependent terminology. Reasoning about the relevant business concepts, rather than database tables, helps to identify those parts of the data that might be relevant for data mining, and to identify missing information, that might e.g., be provided from external sources. Ontologies are a formally sound and well-suited choice for organizing the conceptual data that is subject to KDD applications, but are hardly found in any commercial KDD environment.

To provide support for creating KDD applications, preprocessing steps and operator chains should be formulated in terms of such higher-level data representations, based on higher-level operators; this eases the formulation of preprocessing steps and increases the understandability of case studies. A modern KDD tool should hence allow to represent KDD cases in a higher-level formalism, which is operational and understandable at the same time. Crucial preprocessing steps contain selecting relevant tables and attributes, joining them to reflect semantically meaningful concepts, and sub-sampling, so that state-of-the-art learning toolboxes can process the data in main memory. Besides, most learning algorithms can handle the data in specific representations only, like one fixed-sized vector per example. The question of how to transform raw data into a format that fits a given learner's demands turns out to be highly non-trivial, and is – up to now – solved best by adapting successful solutions from the past. The adaption of best-practice cases from the past should hence be well supported. Best practice cases should be collected and indexed, in order to allow for case-based reasoning approaches that shorten the trial-and-error efforts common to many KDD applications.

To address the issues of the second paragraph above, that are related to work of the author, the following list of properties desirable for KDD toolboxes was motivated in chapter 8: Prepro-

cessing operators should scale to real-world databases. A library of operators should be provided that can flexibly be composed to preprocessing cases specified at a high level, referring only to high-level descriptions of the data involved. The hierarchy that is part of the CRISP-DM phases should be reflected by a hierarchical organization of preprocessing cases. All parts of a KDD preprocessing case should be specified in terms of an operational meta-data representation language. It should be possible to check the validity of cases automatically. General interfaces should ease the application of different tools for different parts of the KDD process; storing the data in databases is one way to support the exchange of data between different tools.

Contributed software

The open-source (GPL) MININGMART system presented in this thesis was designed to meet many of the above-mentioned criteria for KDD environments. It is an integrated environment specifically tailored towards supporting the preprocessing phase. Both the data and its transformations are organized in terms of hierarchically organized high-level specifications. This abstract level is exclusively referred to when a user specifies data transformations. The abstract specifications only need to be linked to low-level counterparts for executing cases. All operators directly access the data in relational databases, and write their results back to the database, preferably in the form of views. The validity of cases can be checked based on a set of constraints that are part of the specification of each operator. The meta-data compiler that operationalizes the high-level specifications, many of its operators, their corresponding specifications, and many aspects of system integration and maintenance are contributions by the author of this thesis.

The MININGMART system supports e.g., the transformation of raw multi-relational data to attribute-value (single table) representations, as well as as many other transformations required to fit the demands of specific learning algorithms. It also supports steps of increasing the quality of data.

In the next step, learning environments like YALE may be used to sub-sample the data from the target relation. YALE contains several operators for learning and validating results, and operators to apply models to previously unseen data (deployment). The novel algorithms presented in this thesis have been implemented in YALE by the author. This includes main memory implementations of KBS-SD and ADA^{2}BOOST with integrated options to stratify and use different reweighting strategies. SDRI has been implemented for evaluation purposes. Another operator has been implemented that realizes $\text{KBS}_{\text{stream}}$ presented in chapter 6 for mining from data stream with concept drift. Further, a MIDOS-like operator that – in the current implementation – yields just the best rule with respect to a specified utility function has been provided. In cooperation with Michael Wurst it has been adapted to distributed data and is now part of the distributed data mining plugin. Also related to this thesis are operators by the author for visualizing soft classifier performances in ROC space, the identification of a corresponding threshold for mapping continuous confidence scores to boolean classifications, and various selection metrics, that are applied in a YALE-internal decision stump operator, for example.

10.3. Summary

In this work a number of challenging problems preventing practitioners from applying KDD techniques were pointed out. KDD is a discipline that requires to overcome many technical burdens, to introduce and analyze sound theoretical concepts and techniques, and to finally compile all the available building blocks into practical solutions that answer high-level business questions.

This thesis has illustrated how different levels of abstraction interact in KDD, why a good theoretical foundation is inevitable if guarantees on results are required, how a theoretical analysis allows to reduce complex tasks to better supported alternatives, and how to formally adapt a small number of approved methods to challenging novel tasks.

The severe obstacles for real-world KDD applications sketched in the introduction were mitigated by the results presented in this work. It was illustrated along the CRISP-DM model and a theoretical framework combining statistics, PAC learning theory, and ROC analysis, how to make accurate supervised data mining from real-world databases practical. Discovering novel, unexpected patterns based on sampling strategies for large-scale databases, or reweighting strategies for small data sets, respectively, and combining novel findings to ensembles have been motivated as central building blocks in the data mining part of this work. Alternative solutions were proposed for distributed data and learning under concept drift. All data mining techniques are very general. Most of them can be combined with arbitrary supervised learning algorithms and a variety of different utility functions. This generality is no coincidence, but the result of a novel constructivist approach to learning that was consequently evaluated in this work. The decomposition of utility functions into their building blocks, the decomposition of distributions into explained and unexplained components, and the computation of confidence bounds that allow for probabilistic guarantees have been identified as a collection of theoretically well-based analysis techniques that can be composed to address novel task in a principled way. This result was complemented by a constructive illustration of how to support phases of the KDD process like data understanding and preprocessing, taking the principled approach of increasing transparency in KDD by establishing an intuitive layer based on operational meta-data that compiles into scalable applications.

A. Joint publications

Some parts of this thesis are related to joint work and publications. The following list describes the contributions of the author of this thesis in detail.

Chapter 6 : The chapter on boosting in the presence of concept drift is based on two joint publications, an earlier publication at a workshop (Scholz & Klinkenberg, 2005), and a subsequent journal article (Scholz & Klinkenberg, 2006).

Ralf Klinkenberg has worked on the topic of concept drift for several years, and provided an overview of related work in this field. Throughout the chapter all his relevant publications were cited wherever appropriate. He implemented the concept drift plugin of YALE which was used for the experiments. The results were compared to his previous experimental studies.

The contributions of the author of this thesis are

- the related work section on ensemble methods for data streams inhibiting concept drift.
- the novel knowledge-based sampling algorithm for learning in the presence of concept drift, which includes the data stream adaptation of KBS-SD for boosting classifiers and the continuous re-estimation of model performances that allows to adapt to drifting concepts.
- the ideas on how to quantify concept drifts based on estimated model weights.
- the implementation of the $\text{KBS}_{\text{stream}}$ YALE operator used in the experiment section to evaluate the approach.

In particular, section 6.3 is solely the work of the author of this thesis. The experiment section is joint work with equally weighted contributions.

Chapter 7 : The chapter on distributed subgroup discovery is only in parts based on a joint publication with Michael Wurst (Wurst & Scholz, 2006). The formal framework for distributed subgroup discovery, the theoretical results on the tractability of rule learning in distributed settings, the corresponding bounds, and the formal definitions of the novel task are the work of the author of this thesis, published prior to the article above (Scholz, 2005d; Scholz, 2005c).

Sections 7.6.3 to 7.8 of chapter 7 are joint work with Michael Wurst. The author of this thesis derived the optimistic scoring functions utilized by the distributed algorithms, showed their correctness, conceptionally designed the novel algorithms, and implemented a MIDOS-like algorithm based on (Wrobel, 1997) in YALE.

Michael Wurst implemented the distributed data mining plugin of YALE used to evaluate the novel algorithms, adapted the non-distributed MIDOS-like operator to this plugin, and contributed the overview of distributed association rule mining.

The experiments are joint work with equally weighted contributions.

Chapter 8 and 9 : In the European research project MININGMART the author of this thesis collaborated with several partners from academia and industry to realize the conceptional ideas underlying the MININGMART system. The joint publications (Morik & Scholz, 2002; Euler et al., 2003; Morik & Scholz, 2004) provide an overview of the work of this project.

The main contributions of the author of this thesis are related to the meta-data compiler. This includes the conceptional design of the version that was finally released and large parts of their implementation.

The cited work on ontologies for KDD (Euler & Scholz, 2004) has been published at an early stage; this line of research has been continued by Timm Euler without any further contributions by the author of this thesis.

The meta-data language M4 was one of the foundations of the project. It has been adapted by the author of this thesis, and finally been documented in cooperation with Timm Euler (Scholz & Euler, 2002).

Constraints, conditions, and assertions in M4 have been documented in a joint publication (Scholz et al., 2002). However, for this thesis only the aspects related to the MININGMART *system* are relevant. This part of the publication has solely been written by the author of this thesis.

B. Notation

Sets

$\mathbb{N}$: set of natural numbers

$\mathbb{R}$: set of real numbers, $\mathbb{R}^+$ refers to the positive subset including 0

$\times$: set product, e.g. $\mathbb{R} \times \mathbb{N}$ refers to the set of all tuples (r, n) with $r \in \mathbb{R}$ and $n \in \mathbb{N}$

$\overline{S}$: complement of set S, usually with respect to a fixed instance space

$\mathcal{P}(\mathcal{S})$: the power set of S

$S^{\mathbb{N}}$: the set containing all finite sequences of elements from set S

Functions

$I[\cdot]$: indicator function, $I : \{\text{true}, \text{false}\} \to \{1, 0\}$, evaluates to 1 iff the argument is true

$\exp(\cdot)$: exponential function, $\exp(x) = e^x \approx 2.71828^x$

$\ln(\cdot)$: natural logarithm, logarithm to the base $e \approx 2.71828^x$

$\log(\cdot)$: logarithm to base 2

Probabilities, distributions

$\mathbb{E}_D[X]$: expected value of a random variable X with respect to a probability density function (pdf) D; the subscript may be omitted if clear from context

$\sigma_D[X]$: standard deviation of random variable X with respect to pdf D; the subscript may be omitted if clear from context

$N(\mu, \sigma)$: the normal distribution with mean μ and standard variation σ

z_x : the inverse standard normal distribution, yields the value z that corresponds to $\Pr(X < z) = x$ for $X \sim N(0, 1)$

$B(m, p)$: the Binomial distribution with mean p and m repetitions of Bernoulli trials

$\Pr(\cdot)$: probability, e.g. $\Pr_{(x,y)\sim D}(y \mid x)$ denotes the conditional probability of label y given observation x under pdf D; subscripts are omitted if clear from the context

$\Pr[\cdot]$: similar to $\Pr(\cdot)$, but indicates that the argument is a set

$\widehat{\Pr}(\cdot)$: a function providing probability *estimates*

$\sim$: denotes that a random variable is sampled with respect to a specific distribution / pdf: $x \sim N(0, 1)$ indicates that x is sampled from the standard normal distribution

Instance space, samples, examples

d : number of attributes (dimension)

A_i : a single attribute of a data set$i$ ranges from 1 to d

$\mathcal{X}$: instance space, composed of attributes: $\mathcal{X} = A_1 \times \ldots \times A_d$; $\mathcal{X}$ denotes the set of all possible unlabeled observations. For a given underlying distribution the set $\mathcal{X}$ may also be referred to as a random variable.

x : a single unlabeled observation $x \in \mathcal{X}$ in attribute-value representation; e.g., a vector, where component i refers to attribute A_i

$\mathcal{Y}$: label / target attribute / property of interest, often assumed to be boolean: $\{0, 1\}$ is preferred when conditional probabilities are estimated; $\{-1, +1\}$ is more convenient in boosting contexts; using $\mathcal{Y} = \{y_+, y_-\}$ subsumes both cases

y : specific label $y \in \mathcal{Y}$

D : a probability density function, $D : \mathcal{X} \to \mathbb{R}^+$, or $D : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^+$; sometimes sloppily referred to as a probability *distribution*, if the function itself is not used, but is only mentioned for reference

$D_{\mathcal{E}}$: uniform distribution over a set $\mathcal{E}$

e : labeled example, $e = (x, y) \in \mathcal{X} \times \mathcal{Y}$

n : number of examples in an example set

$\mathcal{E}$: classified set of examples e, $\mathcal{E} = \{e_1, \ldots, e_n\}$; $\mathcal{E}$ may contain duplicates, so example *sequence* or *multi-set* would be more precise, but is uncommon in the literature

m : sample size, used in various contexts

S : sub-sample of an example set, usually containing multiple examples; samples are often referred to as $\mathcal{E}$, if the superset is not relevant

s : number of different notes / sites in distributed settings

Models

k : number of rules or models, number of iterations, ...

t : runtime index for learning iterations etc.

h : a model, often annotated with an index t if multiple models are induced; crip classifiers are of the form $h : \mathcal{X} \to \mathcal{Y}$, while soft classifiers have a domain of $[0, 1]$

$r : A \to C$: a classification rule; r is used for reference (optional), A is the antecedent and C the consequence; A is a boolean expression $A : \mathcal{X} \to \{\text{false}, \text{true}\}$, C predicts a class $y \in \mathcal{Y}$; if several rules are used, the antecedent of rule r_t is denoted as $A^{(t)}$

$\mathcal{H} \subseteq \mathcal{P}(\mathcal{X})$: hypothesis space, the set of potential models of a learning algorithm

$\mathcal{C} \subset \mathcal{P}(\mathcal{X})$: concept class, contains the true *target concept* $c \subset \mathcal{X}$ in the PAC model

$\text{VCdim}(\mathcal{H})$: Vapnik-Chervonenkis dimension of hypothesis space $\mathcal{H}$

Contigency table

P : total number of positive examples in an example set

N : total number of negative examples in an example set

TP, or p : absolute number of *true positives*, the number of examples correctly identified as positive by a boolean prediction model

FP, or n : absolute number of *false positives*, the number of examples incorrectly identified as positive by a boolean prediction model

FN, or $\overline{p}$: absolute number of *false negatives*, the number of examples incorrectly identified as negative by a boolean prediction model

TN, or $\overline{n}$: absolute number of *true negatives*, the number of examples correctly identified as negative by a boolean prediction model

TPr : true positive rate, equals TP/P, fraction of positives that are classified as positives by a boolean prediction model, also referred to as "recall"

FPr : false positive rate, equals FP/N, fraction of negatives that are classified as positives by a boolean prediction model

$p_i(r)$: absolute number of true positives of rule r at site i in Chap. 7

$n_i(r)$: absolute number of false positives of rule r at site i in Chap. 7

Model performance

$\epsilon \in [0,1]$: error rate of a classifier; more general: permitted deviation from optimum; in Chap. 7.1 ϵ denotes point-wise deviation of local from global conditional class distribution

$\delta \in [0,1]$: confidence parameter; tolerable probability that an algorithm fails

$Q_D : \mathcal{H} \to \mathbb{R}$: a utility function defined with respect to a pdf D; maps each model $h \in \mathcal{H}$ to a real-valued utility score

$\hat{Q}$: estimator function $\hat{Q} : \mathcal{H} \times (\mathcal{X} \times \mathcal{Y})^{\mathbb{N}} \to \mathbb{R}$ corresponding to utility function Q_D; defined as $\hat{Q}(h, \mathcal{E}) := Q_{D_\mathcal{E}}(h)$ for uniform distribution $D_\mathcal{E}$ over example set $\mathcal{E}$

ACC : accuracy of a model, probability of predicting correctly, or fraction of correct predictions when referring to a training set

COV : coverage of a rule, alias "support", $\mathrm{COV}_D(A \to C) := Pr_D[A]$

PREC : precision of a rule, alias "confidence", $\mathrm{PREC}_D(A \to C) := Pr_D[C \mid A]$

BIAS : bias of a rule or subset, $\mathrm{BIAS}_D(A \to C) := Pr_D[C \mid A] - Pr_D[C]$

$Q^{(\alpha)}$: class of utility functions $Q_D^{(\alpha)}(A \to C) := (\mathrm{COV}_D(A \to C))^\alpha \cdot \mathrm{BIAS}_D(A \to C)$

WRACC : $\mathrm{WRACC}_D(A \to C) := \mathrm{COV}_D(A \to C) \cdot \mathrm{BIAS}_D(A \to C)$
("weighted relative accuracy", equal to $Q_D^{(1)}$)

$\textsc{Lift}$: $\textsc{Lift}_D(A \to C) := \textsc{Prec}_D(A \to C)/\Pr_D[C]$

$\widehat{\textsc{Lift}}(r)$: an estimate of the $\textsc{Lift}$ of rule r

LR : lift ratio, $\mathrm{LR}_D(A \to C) := \textsc{Lift}_D(A \to C)/\textsc{Lift}_D(A \to \overline{C})$

$\mathrm{AUC}_D(h)$: area under the ROC curve of boolean soft classifier h under pdf D

$\mathrm{AOC}^*_D(h)$: area over the curve in coverage spaces of soft classifier h under pdf D

$\textsc{rq}$: $\textsc{rq}^{(\alpha)}_{D_i}(r) := \textsc{Cov}_{D_i}(r)^\alpha \cdot (\textsc{Bias}_{D_i}(r) - \textsc{Bias}_D(r))$ (in distributed settings)

$\textsc{Cov}_i, \textsc{Bias}_i, \mathrm{WRAcc}_i$: values when evaluating locally at site i (distributed data)

RLU : relative local utility $\mathrm{RLU}_i(r) := \textsc{Cov}_i(r) \cdot (\textsc{Prec}_i(r) - \textsc{Prec}(r))$ at site i

C. Reformulation of gini index utility function

This section proves the following connection between the Gini index

$$\text{gini}(T) \;:=\; \sum_{t\in T} \Pr[t] \sum_{y_i,y_j\in\mathcal{Y},i\neq j} \Pr[y_i \mid t]\Pr[y_j \mid t]$$

for a set T of (disjoint) partitioning subsets, and the utility function

$$u(A \to C) := \frac{\text{Cov}(A \to C)}{1-\text{Cov}(A \to C)}\text{Bias}(A \to C)^2.$$

Proposition 15 *For any rule $A \to C$ partitioning the instance space $\mathcal{X}$ into the subsets A and $\overline{A}$, the following equality holds for boolean target attributes:*

$$\text{gini}(A) = -u(A \to C) + \Pr(C)\Pr(\overline{C}) \tag{C.1}$$

Proof
For boolean target attributes and two nodes the gini index simplifies to

$$\text{gini}(\{A,\overline{A}\}) \;=\; \Pr[A]\cdot\Pr[C \mid A]\cdot\Pr\left[\overline{C} \mid A\right] + \Pr\left[\overline{A}\right]\cdot\Pr\left[C \mid \overline{A}\right]\cdot\Pr\left[\overline{C} \mid \overline{A}\right].$$

With simple term manipulations this can be rewritten as

$$\begin{aligned}
& \frac{\Pr[C,A]\Pr\left[\overline{C},A\right]}{\Pr[A]} + \frac{\Pr\left[C,\overline{A}\right]\Pr\left[\overline{C},\overline{A}\right]}{\Pr\left[\overline{A}\right]} \\
= \;& \frac{\Pr[C,A]\cdot(\Pr[A]-\Pr[C,A])}{\Pr[A]} + \frac{\Pr\left[C,\overline{A}\right]\Pr\left[\overline{C},\overline{A}\right]}{1-\Pr[A]} \\
= \;& \Pr[C,A] - \frac{\Pr[C,A]^2}{\Pr[A]} + \frac{\Pr\left[C,\overline{A}\right]\Pr\left[\overline{C},\overline{A}\right]}{1-\Pr[A]} \\
= \;& \Pr[C,A] - \frac{\Pr[C,A]^2}{\Pr[A]} + \frac{(\Pr[C]-\Pr[C,A])\cdot\left(\Pr\left[\overline{C}\right]-\Pr\left[\overline{C},A\right]\right)}{1-\Pr[A]}
\end{aligned} \tag{C.2}$$

Substituting $\pi_{+/-}$ for $\Pr[C]$ / $\Pr\left[\overline{C}\right]$, π_A for $\Pr[A]$, and q for $\Pr[C, A]$, eqn. (C.2) can further be rewritten as

$$\begin{aligned}
& q - \frac{q^2}{\pi_A} + \frac{(\pi_+ - q) \cdot \left(\pi_- - \Pr\left[\overline{C}, A\right]\right)}{1 - \pi_A} \\
= \; & \frac{q(1-\pi_A)}{1-\pi_A} - \frac{q^2}{\pi_A} + \frac{(\pi_+ - q) \cdot (\pi_- - \pi_A + q)}{1 - \pi_A} \\
= \; & -\frac{q^2}{\pi_A} + \frac{q - q\pi_A + \pi_+\pi_- - \pi_+\pi_A + \pi_+ q - q\pi_- + q\pi_A - q^2}{1 - \pi_A} \\
= \; & -\frac{q^2}{\pi_A} + \frac{q + \pi_+(1-\pi_+) - \pi_+\pi_A + \pi_+ q - q(1-\pi_+) - q^2}{1 - \pi_A} \\
= \; & -\left(\frac{q^2}{\pi_A} + \frac{q^2}{1-\pi_A}\right) + \frac{\pi_+ - \pi_+^2 - \pi_+\pi_A + 2\pi_+ q}{1 - \pi_A} \\
= \; & -\frac{q^2}{\pi_A(1-\pi_A)} + \pi_+ \frac{1-\pi_A}{1-\pi_A} + \frac{2\pi_+ q - \pi_+^2}{1 - \pi_A}
\end{aligned} \tag{C.3}$$

Substituting back and simplifying eqn. (C.3) we reach at

$$\begin{aligned}
& \text{gini}(\{A, \overline{A}\}) \\
= & -\frac{\Pr[C, A]^2}{\Pr[A]\,(1 - \Pr[A])} + \frac{2\Pr[C]\Pr[C, A] - \Pr[C]^2}{1 - \Pr[A]} + \Pr[C] \\
= & -\frac{\Pr[A]}{\Pr\left[\overline{A}\right]} \cdot \Big(\Pr[C \mid A]^2 - 2\Pr[C]\Pr[C \mid A]\Big) + \Pr[C] - \frac{\Pr[C]^2}{\Pr\left[\overline{A}\right]} \\
& - \frac{\Pr[A]}{\Pr\left[\overline{A}\right]}\Pr[C]^2 + \frac{\Pr[A]}{\Pr\left[\overline{A}\right]}\Pr[C]^2 \\
= & -\frac{\Pr[A]}{\Pr\left[\overline{A}\right]} \cdot (\Pr[C \mid A] - \Pr[C])^2 + \Pr[C] - \Pr[C]^2 \left(\frac{1}{\Pr\left[\overline{A}\right]} - \frac{\Pr[A]}{\Pr\left[\overline{A}\right]}\right) \\
= & -u(A \to C) + \Pr[C] - \Pr[C]^2 \\
= & -u(A \to C) + \Pr[C]\Pr\left[\overline{C}\right],
\end{aligned}$$

which proves the proposition. □

Bibliography

Agrawal, R. and Shafer, J. C. (1996). Parallel mining of association rules. *IEEE Trans. On Knowledge And Data Engineering*, *8*.

Agrawal, R. and Srikant, R. (1994). Fast algorithms for mining association rules in large data bases. *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB '94)* (pp. 478–499). Santiago, Chile.

Allan, J. (1996). Incremental relevance feedback for information filtering. *Proc. 19th Annual ACM SIGIR Conf. on Research and Development in Information Retrieval (SIGIR '96), Zurich, Swiss, August 18-22, 1996* (pp. 270–278). New York, NY, USA: ACM Press.

Atzmüller, M. and Puppe, F. (2006). SD-Map – A Fast Algorithm for Exhaustive Subgroup Discovery. *Proceedings of the 10th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD-06)* (pp. 6–17). Springer.

Atzmüller, M., Puppe, F., and Buscher, H.-P. (2005). Exploiting Background Knowledge for Knowledge-Intensive Subgroup Discovery. *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI)* (pp. 647–652). Professional Book Center.

Aubrecht, P., Zelezny, F., Miksovsky, P., and Stepankova, O. (2002). SumatraTT: Towards a Universal Data Preprocessor. *Cybernetics and Systems* (pp. 818–823). Vienna: Austrian Society for Cybernetics Studies.

Auer, P., Holte, R. C., and Maass, W. (1995). Theory and Applications of Agnostic PAC-Learning with Small Decision Trees. *International Conference on Machine Learning* (pp. 21–29).

Balabanovic, M. (1997). An adaptive web page recommendation service. *Proc. First Int'l Conf. on Autonomous Agents* (pp. 378–385). New York, NY, USA: ACM Press.

Bauer, E. and Kohavi, R. (1999). An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting, and Variants. *Machine Learning*, *36*, 105–139.

Bernstein, A., Hill, S., and Provost, F. (2002). *An intelligent assistant for the knowledge discovery process* (Technical Report IS02-02). New York University, Leonard Stern School of Business.

Bernstein, A., Hill, S., and Provost, F. (2005). Toward Intelligent Assistance for a Data Mining Process: An Ontology-Based Approach for Cost-Sensitive Classification. *IEEE Transactions on Knowledge and Data Engineering*, *17*, 503–518.

Blake, C. and Merz, C. (1998). UCI repository of machine learning databases. http://www.ics.uci.edu/~mlearn/MLRepository.html.

Blumer, A., Ehrenfeucht, A., Haussler, D., and Warmuth, M. (1989). Learnability and the Vapnik–Chervonenkis dimension. *Journal of the ACM*, *36*, 929–965.

Brazdil, P., Soares, C., and da Costa, J. P. (2003). Ranking Learning Algorithms: Using IBL and Meta-Learning on Accuracy and Time Results. *Machine Learning*, *50*, 251–277.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, *13*, 30–37.

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

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and regression trees*. Belmont, CA: Wadsworth.

Brin, S., Motwani, R., Ullman, J., and Tsur, S. (1997). Dynamic Itemset Counting and Implication Rules for Market Basket Data. *Proceedings of ACM SIGMOD Conference on Management of Data (SIGMOD '97)* (pp. 255–264). Tucson, AZ.

Burges, C. (1998). A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, *2*, 121–167.

Cannataro, M., Congiusta, A., Mastroianni, C., Pugliese, A., Talia, D., and Trunfio, P. (2004). Grid-Based Data Mining and Knowledge Discovery. In N. Zhong and J. Liu (Eds.), *Intelligent Technologies for Information Analysis*. Springer.

Cestnik, B. (1990). Estimating Probabilities: A Crucial Task in Machine Learning. *9th European Conference on Artificial Intelligence (ECAI)* (pp. 147–149).

Chapman, P., Clinton, J., Kerber, R., Khabaza, T., Reinartz, T., Shearer, C., and Wirth, R. (2000). *Crisp–Dm 1.0* (Technical Report). The CRISP–DM Consortium.

Chawla, N., Bowyer, K., Hall, L., and Kegelmeyer, W. (2002). Smote: Synthetic minority oversampling technique. *Artifical Intelligence Research*, 321–357.

Chen, C. and Yang (2005). Progressive Sampling for Association Rules based on Sampling Error Estimation. *Proc. of the 9th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD-05)*. Springer.

Chernoff, H. (1952). A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *The Annals of Mathematical Statistics*, *23*, 493–507.

Cheung, D., Han, J., Ng, V., Fu, A., and Fu, Y. (1996). A Fast Distributed Algorithm for Mining Association Rules. *International Conference on Parallel and Distributed Information Systems*.

Cheung, D. W.-L. and Xiao, Y. (1998). Effect of Data Skewness in Parallel Mining of Association Rules. *Pacific-Asia Conference on Knowledge Discovery and Data Mining* (pp. 48–60).

Chudzian, C., Granat, J., and Traczyk, W. (2003). *Call Center Case* (Technical Report D17.2b). IST Project MiningMart, IST-11993.

Clark, P. and Boswell, R. (1991). Rule Induction with CN2: Some Recent Improvements. *Proceedings of Fifth European Working Session on Learning (EWSL-91)* (pp. 151–163). Springer.

Clark, P. and Niblett, T. (1989). The CN2 Induction Algorithm. *Machine Learning*, *3*, 261–283.

Codd, E. F. (1970). A relational model of data for large shared data banks. *Communications of the ACM*, *13*, 377–387.

Cohen, W. W. (1996). Learning rules that classify e-mail. *Proceedings of the 1996 AAAI Spring Symposium on Machine Learning in Information Access (MLIA '96)*. Stanford, CA, USA: AAAI Press.

Cunningham, P. and Carney, J. (2000). Diversity versus Quality in Classification Ensembles Based on Feature Selection. In de R. L. Mántaras and E. Plaza (Eds.), *Proceedings of the 11th Conference on Machine Learning (ECML 2000)*, vol. 1810 of *LNCS*, 109 – 116. Barcelona, Spain: Springer Verlag Berlin.

Dach, D. (2006). Effiziente Entdeckung unabhängiger Subgruppen in großen Datenbanken. Master's thesis, Universität Dortmund, Lehrstuhl Informatik VIII.

Dietterich, T. G. (2000). An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization. *Machine Learning*, *40*, 139–157.

Do, H.-H. and Rahm, E. (2002). COMA— a system for flexible combination of schema matching approaches. *Proceedings of the 28th VLDB Conference*. Hong Kong.

Domingo, C., Gavalda, R., and Watanabe, O. (2001). Adaptive sampling methods for scaling up knowledge discovery algorithms. In H. Liu and H. Motoda (Eds.), *Feature extraction, construction, and selection: A data mining perspective*, chapter 8, 133–150. Kluwer Academic Publishers.

Domingo, C. and Watanabe, O. (2000). MadaBoost: A Modification of AdaBoost. *Proc. of the Thirteenth Annual Conference on Computational Learning Theory* (pp. 180–189). Morgan Kaufmann.

Domingos, P. and Hulten, G. (2000). Mining High Speed Data Streams. *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '00)* (pp. 71–80).

Drummond, C. and Holte, R. C. (2006). Cost curves: An improved method for visualizing classifier performance. *Machine Learning*, *65*, 95–130.

Eswaran, K. P., Gray, J., Lorie, R. A., and Traiger, I. L. (1976). The Notions of Consistency and Predicate Locks in a Database System. *Communications of the ACM*, *19*, 624–633.

Euler, T. (2002a). *How to implement M4 operators* (Technical Report TR12-04). IST Project MiningMart, IST-11993.

Euler, T. (2002b). *Operator Specifications* (Technical Report TR12-02). IST Project MiningMart, IST-11993.

Euler, T. (2005a). An Adaptable Software Product Evaluation Metric. *Proceedings of the 9th IASTED International Conference on Software Engineering and Applications (SEA)*. Phoenix, Arizona, USA.

Euler, T. (2005b). Publishing Operational Models of Data Mining Case Studies. *Proceedings of the Workshop on Data Mining Case Studies at the 5th IEEE International Conference on Data Mining (ICDM)* (pp. 99–106). Houston, Texas, USA.

Euler, T. (To appear). *Knowledge Discovery in Databases at a Conceptual Level*. Doctoral dissertation, Fachbereich Informatik, Universität Dortmund.

Euler, T., Morik, K., and Scholz, M. (2003). MiningMart: Sharing Successful KDD Processes. *LLWA 2003 – Tagungsband der GI-Workshop-Woche Lehren – Lernen – Wissen – Adaptivitat* (pp. 121–122).

Euler, T. and Scholz, M. (2004). Using Ontologies in a KDD Workbench. *Workshop on Knowledge Discovery and Ontologies at ECML/PKDD '04* (pp. 103–108). Pisa, Italy.

Fan, W. (2004). Systematic Data Selection to Mine Concept-Drifting Data Streams. *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '04)* (pp. 128–137). Seattle, WA, USA: ACM Press.

Fan, W., Davidson, I., Zadrozny, B., and Yu, P. S. (2005). An Improved Categorization of Classifier's Sensitivity on Sample Selection Bias. *Proceedings of the 5th IEEE International Conference on Data Mining (ICDM)* (pp. 605–608). IEEE Computer Society.

Fawcett, T. (2001). Using Rule Sets to Maximize ROC Performance. *Proceedings of the IEEE International Conference on Data Mining (ICDM)* (pp. 131–138). IEEE Computer Society.

Fawcett, T. (2003). *ROC Graphs: Notes and Practical Considerations for Researchers* (Technical Report HPL-2003-4). HP Laboratories, Palo Alto, CA, USA.

Fawcett, T. and Flach, P. A. (2005). A Response to Webb and Ting's On the Application of ROC Analysis to Predict Classification Performance under Varying Class Distributions. *Machine Learning*, *58*, 33–38.

Fayyad, U. M. and Irani, K. B. (1993). Multi–interval discretization of continuous–valued attributes for classification learning. *Proceedings of the 13th International Joint Conference on Artificial Intelligence* (pp. 1022–1029). San Mateo, CA: Morgan Kaufmann.

Fayyad, U. M., Piatetsky-Shapiro, G., and Smyth, P. (1996). From Data Mining to Knowledge Discovery: An overview. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth and R. Uthurusamy (Eds.), *Advances in Knowledge Discovery and Data Mining*, chapter 1, 1–34. AAAI/MIT Press.

Ferri, C., Flach, P., and Hernández-Orallo, J. (2002). Learning Decision Trees using the Area Under the ROC Curve. *Proceedings of the 19th International Conference on Machine Learning (ICML)*. Morgan Kaufmann.

Ferri, C., Lachiche, N., Macskassy, S. A., and Rakotomamonjy, A. (Eds.). (2005). *ROC Analysis in ML, co-located with ICML'05*. Bonn, Germany. http://www.dsic.upv.es/ flip/ROCML2005/.

Fischer, P. (1999). *Algorithmisches Lernen*. Teubner Verlag.

Fischer, S., Klinkenberg, R., Mierswa, I., and Ritthoff, O. (2002). YALE*: Yet Another Learning Environment – Tutorial* (Technical Report CI-136/02). Collaborative Research Center 531, University of Dortmund, Dortmund, Germany. ISSN 1433-3325. http://yale.sf.net/ .

Flach, P. A. (2003). The Geometry of ROC Space: Understanding Machine Learning Metrics through ROC Isometrics. *Proceedings of the 20th International Conference on Machine Learning (ICML-03)* (pp. 194–201). Washington D.C., USA: Morgan Kaufman.

Foussette, C., Hakenjos, D., and Scholz, M. (2004). KDD-Cup 2004: Protein Homology Task. *ACM SIGKDD Explorations Newsletter*, *6*, 128 – 131.

Freund, Y., Iyer, R., Schapire, R. E., and Singer, Y. (2003). An Efficient Boosting Algorithm for Combining Preferences. *Journal of Machine Learning Research*, *4*, 933–969.

Freund, Y. and Mason, L. (1999). The alternating decision tree learning algorithm,. *Proceeding of the 16th International Conference on Machine Learning* (pp. 124–133). Morgan Kaufmann.

Freund, Y. and Schapire, R. E. (1996). Experiments with a New Boosting Algorithm. *Proceedings of the 13th International Conference on Machine Learning (ICML)* (pp. 148–156). Morgan Kaufmann.

Freund, Y. and Schapire, R. R. (1997). A decision–theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, *55*, 119 – 139.

Friedman, J. H., Hastie, T., and Tibshirani, R. (2000). Additive logistic regression: A statistical view of boosting. *Annals of Statistics*, 337–374.

Fürnkranz, J. (1999). Separate-and-Conquer Rule Learning. *Artificial Intelligence Review*, *13*, 3–54.

Fürnkranz, J. and Flach, P. (2005). ROC 'n' Rule Learning – Towards a Better Understanding of Covering Algorithms. *Machine Learning*, *58*, 39–77.

Fürnkranz, J. and Flach, P. A. (2003). An Analysis of Rule Evaluation Metrics. *Proceedings of the 20th International Conference on Machine Learning (ICML-03)* (pp. 202–209). Washington D.C., USA: Morgan Kaufman.

Gimbel, M., Klein, M., and Lockemann, P. C. (2004). Interactivity, Scalability and Resource Control for Efficient KDD Support in DBMS. In R. Meo, P. L. Lanzi and M. Klemettinen (Eds.), *Database Support for Data Mining Applications (LNAI 2682)*, 174–193. Berlin, Heidelberg: Springer.

Gold, E. M. (1967). Language identification in the limit. *Information and Control*, *14*, 447–474.

Grandvalet, Y. (2004). Bagging Equalizes Influence. *Machine Learning*, *55*, 251–270.

Guyon, I., Matic, N., and Vapnik, V. (1996). Discovering informative patterns and data cleaning. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth and R. Uthurusamy (Eds.), *Advances in knowledge discovery and data mining*, chapter 2, 181–204. Menlo Park, California: AAAI Press/The MIT Press.

Hairnarayan, V., Rajaraman, A., and Ullman, J. (1996). Implementing Data Cubes Efficiently. *Proc. ACM-SIGMOD Int. Conf. Management of Data*.

Han, J., Pei, J., and Yin, Y. (2000). Mining frequent patterns without candidate generation. *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data (SIGMOD)* (pp. 1–12). ACM Press.

Hand, D. (2002). Pattern detection and discovery. In D. Hand, N. Adams and R. Bolton (Eds.), *Pattern detection and discovery*. Springer.

Hand, D. J., Adams, N. M., and Bolton, R. J. (2002). *Proceedings of the ESF Exploratory Workshop on Pattern Detection and Discovery*, vol. 2447 of *LNAI*. Springer.

Hastie, T., Tibshirani, R., and J., F. (2001). *The Elements of Statistical Learning – Data Mining, Inference and Prediction*. Springer.

Heckerman, D. (1995). *A tutorial on learning with bayesian networks* (Technical Report MSR-TR-95-06). Microsoft Research, Redmond, Washington, 1995.

Heilemann, U. and Münch, H. J. (1999). *Classification of west german business cycles* (Technical Report 11). Collaborative Research Center on Reduction of Complexity for Multivariate Data (SFB 475), University of Dortmund, Germany.

Hernández-Orallo, J., Ferri, C., Lachiche, N., and Flach, P. A. (Eds.). (2004). *First Workshop of ROC Analysis in AI (ROCAI'04), co-located with ECAI'04*. Valencia, Spain. http://www.dsic.upv.es/~flip/ROCAI2004/.

Hoche, S. and Wrobel, S. (2002). Scaling Boosting by Margin-Based Inclusion of Features and Relations. *Machine Learning: ECML 2002* (pp. 148–160). Springer.

Hoeffding, W. (1963). Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, *58*, 13–30.

Holte, R. C. (1993). Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, *11*, 63–90.

Hulten, G. and Domingos, P. (2002). Mining Complex Models from Arbitrarily Large Databases in Constant Time. *2002 ACM SIGKDD Int. Conf. Knowledge Discovery in Databases (KDD'02)*.

Hulten, G., Spencer, L., and Domingos, P. (2001). Mining Time-Changing Data Streams. *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '01)* (pp. 97 – 106).

International Organization for Standardization (ISO) (2003a). *Information Technology – Database Language – SQL*. Standard No. ISO/IEC 9075:2003.

International Organization for Standardization (ISO) (2003b). *Information Technology – Database Language – SQL Multimedia and Application Packages – Part 6: Data Mining*. Draft Standard No. ISO/IEC 13249-6:2003.

Jacobson, I., Booch, G., and Rumbaugh, J. (1999). *The Unified Software Development Process*. Addison-Wesley.

Jaroszewicz, S. and Simovici, D. A. (2004). Interestingness of Frequent Itemsets Using Bayesian Networks as Background Knowledge. *Proceedings of the 10th International Conference on Knowledge Discovery and Data Mining (KDD-2004)*. AAAI Press.

Jin, R., Liu, Y., Si, L., Carbonell, Carbonell, J., and Hauptmann, A. G. (2003). A New Boosting Algorithm Using Input–Dependent Regularizer. *The Twentieth International Conference on Machine Learning (ICML 03), Washington, DC, 2003*.

Joachims, T. (2000). Estimating the generalization performance of a SVM efficiently. *Proceedings of the International Conference on Machine Learning* (pp. 431–438). San Francisco, CA, USA: Morgan Kaufman.

Joachims, T., Freitag, D., and Mitchell, T. (1997). WebWatcher: A tour guide for the world wide web. *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)* (pp. 770 – 777). Morgan Kaufmann.

John, G. H. and Langley, P. (1995). Estimating continuous distributions in Bayesian classifiers. *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence* (pp. 338–345). Morgan Kaufmann.

John, G. H. and Langley, P. (1996). Static Versus Dynamic Sampling for Data Mining. *Proceedings of the Second International Conference on Knowledge Discovery in Databases and Data Mining*.

JSR-73 Expert Group (2004). *Java Data Mining API*. Java Specification Request No. 73.

Kearns, M. and Vazirani, U. (1994). *An introduction to computational learning theory*. MIT Press.

Kearns, M. J., Schapire, R. E., and Sellie, L. (1992). Toward Efficient Agnostic Learning. *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory (COLT)* (pp. 341–352).

Khoussainov, R., Heß, A., and Kushmerick, N. (2005). Ensembles of Biased Classifiers. *Proceedings of the 22nd International Conference on Machine Learning (ICML)* (pp. 425–432). ACM.

Kietz, J.-U., Fiammengo, A., Beccari, G., and Zücker, R. (2000a). *Data Sets, Meta-data and Preprocessing Operators at Swiss Life and CSELT* (Technical Report D6.2). IST Project MiningMart, IST-11993.

Kietz, J.-U., Vaduva, A., and Zücker, R. (2000b). Mining Mart: Combining Case-Based-Reasoning and Multi-Strategy Learning into a Framework to reuse KDD-Application. *Proceedings of the 5th International Workshop on Multistrategy Learning (MSL2000)*. Guimares, Portugal.

Kietz, J.-U., Vaduva, A., and Zücker, R. (2001). MiningMart: Metadata-Driven Preprocessing. *Proceedings of the ECML/PKDD Workshop on Database Support for KDD*.

Kivinen, J. and Warmuth, M. K. (1999). Boosting as Entropy Projection. *Proc. of the twelfth annual conference on Computational learning theory (COLT'99)* (pp. 134 – 144).

Klinkenberg, R. (2003). Predicting Phases in Business Cycles Under Concept Drift. *LLWA 2003 – Tagungsband der GI-Workshop-Woche* Lehren – Lernen – Wissen – Adaptivität (pp. 3–10). Karlsruhe, Germany.

Klinkenberg, R. (2004). Learning drifting concepts: Example selection vs. example weighting. *Intelligent Data Analysis (IDA), Special Issue on Incremental Learning Systems Capable of Dealing with Concept Drift*, *8*, 281–300.

Klinkenberg, R. and Joachims, T. (2000). Detecting concept drift with support vector machines. *Proceedings of the Seventeenth International Conference on Machine Learning (ICML)* (pp. 487–494). San Francisco, CA, USA: Morgan Kaufmann.

Klinkenberg, R. and Renz, I. (1998). Adaptive information filtering: Learning in the presence of concept drifts. *Workshop Notes of the ICML/AAAI-98 Workshop* Learning for Text Categorization (pp. 33–40). Menlo Park, CA, USA: AAAI Press.

Klinkenberg, R. and Rüping, S. (2003). Concept Drift and the Importance of Examples. In J. Franke, G. Nakhaeizadeh and I. Renz (Eds.), *Text mining – theoretical aspects and applications*, 55–77. Physica-Verlag.

Klösgen, W. (1996). Explora: A Multipattern and Multistrategy Discovery Assistant. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth and R. Uthurusamy (Eds.), *Advances in knowledge discovery and data mining*, chapter 3, 249–272. AAAI Press/MIT Press.

Klösgen, W. (2002). Subgroup discovery. In *Handbook of data mining and knowledge discovery*, 354–367. Oxford University Press.

Klösgen, W. and May, M. (2002). Spatial Subgroup Mining Integrated in an Object-Relational Spatial Database. *Proceedings of the 6th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD)* (pp. 275–286). Springer.

Kohavi, R. (1995). A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. *Proceedings of the 14th International Joint Conference on Artificial Intelligence* (pp. 1137–1143). Morgan Kaufmann.

Kolter, J. Z. and Maloof, M. A. (2005). Using Additive Expert Ensembles to Cope with Concept Drift. *Proceedings of the 22nd International Conference on Machine Learning (ICML-2005)* (pp. 449–456). New York, NY, USA: ACM Press.

Komarek, P. (2004). *Logistic Regression for Data Mining and High-Dimensional Classification*. Doctoral dissertation, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA.

Lavrac, N., Cestnik, B., Gamberger, D., and Flach, P. (2004a). Decision support through subgroup discovery: three case studies and the lessons learned. *Machine Learning*, *57*, 115–143.

Lavrac, N., Flach, P., Kavsek, B., and Todorovski, L. (2002a). Rule Induction for Subgroup Discovery with CN2-SD. *2nd Int. Workshop on Integration and Collaboration Aspects of Data Mining, Decision Support and MetaLearning*.

Lavrac, N., Flach, P., and Zupan, B. (1999). Rule Evaluation Measures: A Unifying View. *9th International Workshop on Inductive Logic Programming*. Springer.

Lavrac, N., Kavsek, B., Flach, P., and Todorovski, L. (2004b). Subgroup discovery with CN2-SD. *Journal of Machine Learning Research*, *5*, 153–188.

Lavrac, N., Zelezny, F., and Flach, P. (2002b). RSD: Relational subgroup discovery through first-order feature construction. *12th International Conference on Inductive Logic Programming*. Springer.

Lazarevic, A. and Obradovic, Z. (2002). Boosting algorithms for parallel and distributed learning. *Distributed and Parallel Databases Journal*, *11*, 203–229.

Lee, H. K. H. and Clyde, M. A. (2004). Lossless Online Bayesian Bagging. *Journal of Machine Learning Research*, *5*, 143–151.

Leite, R. and Brazdil, P. (2004). Improving Progressive Sampling via Meta-learning on Learning Curves. *Proceedings of the 15th European Conference on Machine Learning (ECML)* (pp. 250–261).

Mackassy, S. A., Provost, F., and Rosset, S. (2005). ROC Confidence Bands: An Empirical Evaluation. *Proceedings of the 22nd International Conference on Machine Learning (ICML)* (pp. 537–544). ACM press.

Mackay, D. (1998). Introduction To Monte Carlo Methods. In *Learning in graphical models*, 175–204.

Mason, L., Baxter, J., Bartlett, P., and Frean, M. (1999). *Boosting algorithms as gradient descent in function space* (Technical Report). RSISE, Australian National University.

Mehta, M., Rissanen, J., and Agrawal, R. (1995). MDL-Based Decision Tree Pruning. *Proceedings of the 1st International Conference on Knowledge Discovery and Data Mining (KDD)* (pp. 216–221). AAAI Press.

Melville, P. and Mooney, R. J. (2003). Constructing Diverse Classifier Ensembles using Artificial Training Examples. *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI)* (pp. 505–512). Morgan Kaufmann.

Meo, R. and Psaila, G. (2002). Toward XML-Based Knowledge Discovery Systems. *Proceedings of the International Conference on Data Mining (ICDM)* (pp. 665–668). IEEE Computer Society.

Mierswa, I., Wurst, M., Klinkenberg, R., Scholz, M., and Euler, T. (2006). YALE: Rapid Prototyping for Complex Data Mining Tasks. *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2006)*. ACM Press.

Mitchell, T., Caruana, R., Freitag, D., McDermott, J., and Zabowski, D. (1994). Experience with a learning personal assistant. *Communications of the ACM (CACM)*, *37*, 81–91.

Mitchell, T. M. (1990). Becoming increasingly reactive. *Innovative Approaches to Planning, Scheduling and Control: Proc. of a Workshop* (pp. 459–467). San Diego, CA: Morgan Kaufmann Publisher.

Mitchell, T. M. (1997). *Machine learning*. New York: McGraw Hill.

Morik, K. (2002). Detecting Interesting Instances. *Proceedings of the ESF Exploratory Workshop on Pattern Detection and Discovery* (pp. 13–23). Berlin: Springer Verlag.

Morik, K., Botta, M., Dittrich, K. R., Kietz, J.-U., Portinale, L., Vaduva, A., and Zücker, R. (2001). *M4 – The MiningMart Meta Model* (Technical Report D8/9). IST Project MiningMart, IST-11993.

Morik, K., Boulicaut, J.-F., and Siebes, A. (2005). *Local pattern detection*, vol. 3539 of *Lecture Notes in Computer Science*. Springer.

Morik, K. and Rüping, S. (2002). A Multistrategy Approach to the Classification of Phases in Business Cycles. *European Conference on Machine Learning (ECML-2002)* (pp. 307–318). Springer.

Morik, K. and Scholz, M. (2002). The MiningMart Approach. *GI Jahrestagung* (pp. 811–818). GI, LNI Vol. 19, ISBN 3-88579-348-2.

Morik, K. and Scholz, M. (2004). The MiningMart Approach to Knowledge Discovery in Databases. In N. Zhong and J. Liu (Eds.), *Intelligent Technologies for Information Analysis*, chapter 3, 47–65. Springer.

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

Muggleton, S. (1995). Inverse entailment and progol. *New Generation Computing*, *13*, 245–286.

Musick, R. and Critchlow, T. (1999). Practical Lessons in Supporting Large-scale Computational Science. *ACM SIGMOD Record*, *28*, 49–57.

Neal, R. M. (1993). *Probabilistic Inference Using Markov Chain Monte Carlo Methods* (Technical Report). Department of Computer Science, University of Toronto.

Niculescu-Mizil, A. and Caruana, R. (2005). Predicting Good Probabilities With Supervised Learning. *Proceedings of the 22nd International Conference on Machine Learning (ICML)* (pp. 625–632). ACM press.

Otey, M. E., Parthasarathy, S., Wang, C., Veloso, A., and Meira, W. (2004). Parallel and Distributed Methods for Incremental Frequent Itemset Mining. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, *34*, 2439–2450.

Oza, N. C. and Russell, S. (2001). Online Bagging and Boosting. *Eighth International Workshop on Artificial Intelligence and Statistics*. Key West, Florida, USA.

Pearl, J. (1991). *Probabilistic reasoning in intelligent systems: Networks of plausible inference*. San Mateo, CA: Morgan Kaufmann. 2nd edition.

Piatetsky-Shapiro, G. (1991). Discovery, analysis, and presentation of strong rules. In G. Piatetsky-Shapiro and W. Frawley (Eds.), *Knowledge discovery in databases*, 229 –248. Cambridge, Mass.: AAAI/MIT Press.

Platt, J. C. (1999). Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods. *Advances in Large Margin Classiers*, 61–74.

Provost, F. and Fawcett, T. (1997). Analysis and Visualization of Classifier Performance: Comparison under Imprecise Class and Cost Distributions. *Proceedings of the 3rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)* (pp. 43–48). ACM press.

Provost, F. and Fawcett, T. (2001). Robust Classification for Imprecise Environments. *Machine Learning*, *42*, 203–231.

Provost, F. J., Jensen, D., and Oates, T. (1999). Efficient Progressive Sampling. *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 23–32). ACM press.

Quinlan, J. R. (1993). *C4.5: Programs for machine learning*. Machine Learning. San Mateo, CA: Morgan Kaufmann.

Quinlan, J. R. (1996). Bagging, Boosting, and C4.5. *Proceedings of the 13th National Conference on Artificial Intelligence* (pp. 725–730). AAAI Press, MIT Press.

Quinlan, R. (2001). Relational Learning and Boosting. In S. Dzeroski and N. Lavrac (Eds.), *Relational Data Mining*, chapter 12, 292–306. Springer.

Rahm, E. and Bernstein, P. A. (2001). A Survey of Approaches to Automatic Schema Matching. *The VLDB Journal*, *10*, 334–350.

Raspl, S. (2004). PMML Version 3.0—Overview and Status. *Proceedings of the Workshop on Data Mining Standards, Services and Platforms at the 10th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD)* (pp. 18–22).

Rätsch, G. and Warmuth, M. (2005). Efficient Margin Maximization with Boosting. *Journal of Machine Learning Research*, *6*, 2131–2152.

Richeldi, M. and Perrucci, A. (2002). *Churn Analysis Case Study* (Technical Report D17.2). IST Project MiningMart, IST-11993.

Rissanen, J. (1978). Modeling by shortest data description. *Automatica*, *14*, 465 – 471.

Romei, A., Ruggieri, S., and Turini, F. (2006). KDDML: A Middleware Language and System for Knowledge Discovery in Databases. *Data and Knowledge Engineering*, *57*, 179–220.

Rosset, S. (2004). Model Selection via the AUC. *Proceedings of the 21th International Conference on Machine Learning (ICML-04)*. Banff, Alberta, Canada.

Royce, W. W. (1970). Managing the Development of Large Software Systems. *Technical Papers of Western Electronic Show and Convention (WesCon)* (pp. 1–9). IEEE Computer Society Press.

Rudin, C., Cortes, C., Mohri, M., and Schapire, R. E. (2005). Margin-Based Ranking Meets Boosting in the Middle. *Proceedings of the 18th Annual Conference on Learning Theory (COLT)* (pp. 63–78). Springer.

Rüping, S. (1999). Zeitreihenprognose fur Warenwirtschaftssysteme unter Berucksichtigung asymmetrischer Kostenfunktionen. Master's thesis, Universität Dortmund.

Rüping, S. (2000). *mySVM Manual*. Universität Dortmund, Lehrstuhl Informatik VIII. http://www-ai.cs.uni-dortmund.de/SOFTWARE/MYSVM/.

Rüping, S. (2001). Incremental Learning with Support Vector Machines. *Proceedings of the IEEE International Conference on Data Mining (ICDM '01)* (pp. 641–642).

Rüping, S. (2002). Support Vector Machines in Relational Databases. *Pattern Recognition with Support Vector Machines — First International Workshop, SVM 2002* (pp. 310–320). Springer.

Rüping, S. (2006). *Learning Interpretable Models*. Doctoral dissertation, Universität Dortmund, Fachbereich Informatik.

Saitta, L., Botta, M., Beccari, G., and Klinkenberg, R. (2000). *Studies in Parameter Setting* (Technical Report D4.2). IST Project MiningMart, IST-11993.

Salton, G. and Buckley, C. (1988). Term weighting approaches in automatic text retrieval. *Information Processing and Management*, *24*, 513–523.

Sarawagi, S., Thomas, S., and Agrawal, R. (1998). Integrating Association Rule Mining with relational Database Systems: Alternatives and Implications. *Proceedings of the ACM SIGMOD, International Conference on Management of Data* (pp. 343–354).

Schapire, R. E. (1990). The Strength of Weak Learnability. *Machine Learning*, *5*, 197–227.

Schapire, R. E., Freund, Y., Bartlett, P., and Lee, S. (1998). Boosting the Margin: A New Explanation for the Effectiveness of Voting Methods. *Annals of Statistics*, 1651–1686.

Schapire, R. E., Rochery, M., Rahim, M., and Gupta, N. (2002). Incorporating Prior Knowledge into Boosting. *Proceedings of the 19th International Conference on Machine Learning (ICML)*. Morgan Kaufmann.

Schapire, R. E. and Singer, Y. (1999). Improved Boosting Using Confidence-rated Predictions. *Machine Learning*, *37*, 297–336.

Scheffer, T. and Wrobel, S. (2001). Incremental Maximization of Non-Instance-Averaging Utility Functions with Applications to Knowledge Discovery Problems. *Proceedings of the 18th International Conference on Machine Leraning (ICML-2001)*.

Scheffer, T. and Wrobel, S. (2002). Finding the Most Interesting Patterns in a Database Quickly by Using Sequential Sampling. *Journal of Machine Learning Research*, *3*, 833–862.

Schölkopf, B. and Smola, A. J. (2002). *Learning with Kernels – Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press.

Scholz, M. (2002a). *Representing Constraints, Conditions and Assertions in M4* (Technical Report TR18-01). IST Project MiningMart, IST-11993.

Scholz, M. (2002b). Using real world data for modeling a protocol for ICU monitoring. *Intelligent Data Analysis in Medicine and Pharmacology Workshop (IDAMAP 2002), at 15th European Conference on Artificial Intelligence* (pp. 85–90). Lyon, France.

Scholz, M. (2005a). *Comparing Knowledge-Based Sampling to Boosting* (Technical Report 26). Collaborative Research Center on the Reduction of Complexity for Multivariate Data Structures (SFB 475), University of Dortmund, Germany.

Scholz, M. (2005b). Knowledge-Based Sampling for Subgroup Discovery. In K. Morik, J.-F. Boulicaut and A. Siebes (Eds.), *Local pattern detection*, vol. LNAI 3539 of *Lecture Notes in Artificial Intelligence*, 171–189. Springer.

Scholz, M. (2005c). *On the Complexity of Rule Discovery from Distributed Data* (Technical Report 31). Collaborative Research Center on the Reduction of Complexity for Multivariate Data Structures (SFB 475), University of Dortmund, Germany.

Scholz, M. (2005d). On the Tractability of Rule Discovery from Distributed Data. *Proceedings of the 5th IEEE International Conference on Data Mining (ICDM '05)* (pp. 761–764). Houston, Texas, USA: IEEE Computer Society.

Scholz, M. (2005e). Sampling-Based Sequential Subgroup Mining. *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '05)* (pp. 265–274). Chicago, Illinois, USA: ACM Press.

Scholz, M. (2006). Boosting in PN Spaces. *Proceedings of the 17th European Conference on Machine Learning (ECML-06)* (pp. 377–388). Berlin, Germany: Springer.

Scholz, M. and Euler, T. (2002). *Documentation of the MiningMart Meta Model (M4)* (Technical Report TR12-05). IST Project MiningMart, IST-11993.

Scholz, M., Euler, T., and Saitta, L. (2002). *Applicability Constraints on Learning Operators* (Technical Report D18). IST Project MiningMart, IST-11993.

Scholz, M. and Klinkenberg, R. (2005). An Ensemble Classifier for Drifting Concepts. *Proceedings of the 2nd International Workshop on Knowledge Discovery in Data Streams* (pp. 53–64). In conjunction with ECML/PKDD'05.

Scholz, M. and Klinkenberg, R. (2006). *Boosting Classifiers for Drifting Concepts* (Technical Report 6/06). Collaborative Research Center on the Reduction of Complexity for Multivariate Data Structures (SFB 475), University of Dortmund, Germany.

Scholz, M. and Klinkenberg, R. (2007). Boosting Classifiers for Drifting Concepts. *Intelligent Data Analysis (IDA), Special Issue on Knowledge Discovery from Data Streams (accepted for publication)*.

Schuster, A. and Wolff, R. (2001). Communication-Efficient Distributed Mining of Association Rules. *Proceedings of ACM SIGMOD Conference*.

Shannon, C. and Weaver, W. (1969). *The mathematical theory of communication*. Chapman and Hall. 4 edition edition.

Silberschatz, A. and Tuzhilin, A. (1996). What makes patterns interesting in knowledge discovery systems. *IEEE Transactions on Knowledge and Data Engineering*, *8*, 970–974.

Stanley, K. O. (2003). *Learning Concept Drift with a Committee of Decision Trees* (Technical Report AI-03-302). Department of Computer Sciences, University of Texas at Austin.

Sterling, L. and Shapiro, E. (1994). *The Art of Prolog*. MIT Press. 2nd edition.

Street, W. N. and Kim, Y. (2001). A Streaming Ensemble Algorithm (SEA) for Large-Scale Classification. *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '01)* (pp. 377–382).

Suzuki, E. (2004). Discovering Interesting Exception Rules with Rule Pair. *ECML/PKDD 2004 Workshop, Advances in Inductive Rule Learning*.

Tang, E. K., Suganthan, P. N., and Yao, X. (2006). An analysis of diversity measures. *Machine Learning*, *65*, 247–271.

Taylor, C., Nakhaeizadeh, G., and Lanquillon, C. (1997). Structural change and classification. *Workshop Notes on Dynamically Changing Domains: Theory Revision and Context Dependence Issues, 9th European Conf. on Machine Learning (ECML '97), Prague, Czech Republic* (pp. 67–78).

The Data Mining Group (2004). Predictive Model Markup Language (PMML). `http://www.dmg.org`. Version 3.0.

Theis, W. and Weihs, C. (1999). *Clustering Techniques for the Detection of Business Cycles* (Technical Report 40). Collaborative Research Center on the Reduction of Complexity for Multivariate Data Structures (SFB 475), University of Dortmund, Germany.

Toivonen, H. (1996). Sampling large databases for association rules. *Proceedings of the 22nd VLDB Conference* (pp. 134–145). Morgan Kaufmann.

Tsymbal, A., Pechenizkiy, M., and Cunningham, P. (2003). *Diversity in ensemble feature selection* (Technical Report TCD-CS-2003-44). Trinity College Dublin.

Utgoff, P. E. (1989). Incremental induction of decision trees. *Machine Learning*, *4*, 161–186.

Valiant, L. G. (1984). A theory of the learnable. *Communications of the ACM*, *27*, 1134–1142.

Wagner, M. (2005). Schema-Abbildungen für die Falladaption in MiningMart. Master's thesis, Fachbereich Informatik, Universität Dortmund.

Wang, H., Fan, W., Yu, P. S., and Han, J. (2003). Mining Concept-Drifting Data Streams using Ensemble Classifiers. *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '03)* (pp. 226–235). Washington, DC, USA: ACM Press.

Webb, G. I. and Ting, K. M. (2005). On the Application of ROC Analysis to Predict Classification Performance under Varying Class Distributions. *Machine Learning*, *58*, 25–32.

Widmer, G. and Kubat, M. (1996). Learning in the presence of concept drift and hidden contexts. *Machine Learning*, *23*, 69–101.

Wiederhold, G. (1992). Mediators in the Architecture of Future Information Systems. *IEEE Computer*, *25*, 38–49.

Witten, I. and Frank, E. (2000). *Data mining – practical machine learning tools and techniques with java implementations*. Morgan Kaufmann.

Wolpert, D. and Macready, W. (1997). No Free Lunch Theorems for Optimisation. *IEEE Trans. on Evolutionary Computation*, *1*, 67–82.

Wrobel, S. (1997). An Algorithm for Multi–relational Discovery of Subgroups. *Principles of Data Mining and Knowledge Discovery: First European Symposium (PKDD 97)* (pp. 78–87). Berlin, New York: Springer.

Wu, X. and Srihari, R. (2004). Incorporating Prior Knowledge with Weighted Margin Support Vector Machines. *Proceedings of the 10th International Conference on Knowledge Discovery and Data Mining (KDD-2004)*. AAAI Press.

Wurst, M. and Scholz, M. (2006). Distributed Subgroup Discovery. *Proceedings of the 10th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD-06)* (pp. 421–433). Berlin, Germany: Springer.

Zadrozny, B. (2004). Learning and Evaluating Classifiers under Sample Selection Bias. *Proceedings of the 21th International Conference on Machine Learning (ICML-04)*.

Zadrozny, B., Langford, J., and Naoki, A. (2003). Cost–Sensitive Learning by Cost–Proportionate Example Weighting. *Proceedings of the 2003 IEEE International Conference on Data Mining (ICDM'03)*.

Zaki, M. J. (1999). Parallel and Distributed Association Mining: A Survey. *IEEE Concurrency*, *7*, 14–25.

Zelezny, F. and Lavrac, N. (2006). Propositionalization-based relational subgroup discovery with RSD. *Machine Learning*, *62*, 33–63.

Zhang, S., Zhang, C., and Yu, J. (2004). An Efficient Strategy for Mining Exceptions in Multi-databases. *Information Sciences*, *1-2*, 1–20.

Zhang, Y., Burer, S., and Street, W. N. (2006). Ensemble Pruning Via Semi-definite Programming. *Journal of Machine Learning Research*, *7*, 1315–1338.

Zhong, N., Liu, C., and Ohsuga, S. (1997). A Way of Increasing both Autonomy and Versatility of a KDD System. *Foundations of Intelligent Systems* (pp. 94–105). Springer.

Zhong, N., Liu, C., and Ohsuga, S. (2001). Dynamically Organizing KDD Processes. *International Journal of Pattern Recognition and Artificial Intelligence*, *15*, 451–473.