A Statistical Approach to Rule Learning

Ulrich Rückert

Institut für Informatik, Lehrstuhl für Bioinformatik

A Statistical Approach to Rule Learning

Ulrich Rückert

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften

genehmigten Dissertation.

Die Dissertation wurde am 30.8.2007 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 20.2.2008 angenommen.

Contents

The golden rule is that there are no golden rules.

George Bernard Shaw

Abstract

This dissertation investigates rule learning as a statistical classification problem. Rule learning has a long history in machine learning and is known for inducing interpretable and comprehensible classifiers. Methods from statistical machine learning, on the other hand, have traditionally focused on predictive accuracy, often at the expense of interpretability. The goal of this work is to combine the approaches to obtain highly predictive, yet interpretable classifiers. Unlike many of the heuristics applied in traditional rule learning, statistical principles allow for a better theoretical investigation of rule learning systems, possibly leading to valuable insights. To this end, we investigate two types of rule sets, propositional logic formulae in disjunctive normal form (DNF formulae), and weighted rule sets. For DNF formulae, we investigate the NP-hard problem of empirical risk minimization of DNF formulae of at most k terms. We propose a stochastic local search (SLS) algorithm to solve this problem efficiently. Then, we apply the SLS algorithm in a structural risk minimization procedure to gain small yet predictive rule sets. Alternatively, we combine SLS with an ensemble approach to reduce the estimation part of the prediction error and to upper-bound it analytically. For weighted rule sets, we devise a novel convex optimization criterion, Margin Minus Variance (MMV), a feasible relaxation of the infeasible empirical risk minimization problem. For capacity control we derive a novel concentration inequality. An implementation of these ideas, the Rumble system, yields favorable results in experiments dealing with structure-activity relationships of small molecules. Finally, for multi-relational learning, we propose and use a framework to assess and compare learning systems according to the flow of information. We evaluate different strategies analytically and experimentally and present a novel rule generation criterion that aims at highly diverse rule sets.

Zusammenfassung

Diese Dissertation untersucht Regellernen als ein statistisches Klassifikationsproblem. Regellernen ist seit langem Teil des maschinellen Lernens und bekannt dafür, einfach interpretierbare und verständliche Klassifizierer zu erzeugen. Methoden des statistischen maschinellen Lernens, auf der anderen Seite, konzentrieren sich üblicherweise auf die Vorhersagegenauigkeit, des öfteren auch auf Kosten der Interpretierbarkeit. Das Ziel dieser Arbeit ist es, die beiden Ansätze miteinander zu kombinieren, um Klassifizier mit hoher Vorhersagegenauigkeit und Interpretierbarkeit zu erstellen. Außerdem erlauben statistische Prinzipien, anders als die Heuristiken, die oft in herkömmlichen Regellernern verwendet werden, eine bessere theoretische Analyse, die möglicherweise zu wertvollen Einsichten führen könnte. Wir untersuchen zu diesem Zwecke zwei Arten von Regelmengen, nämlich aussagenlogische Formeln in disjunktiver Normalform (DNF-Formeln) und gewichtete Regelmengen. Im Falle der DNF-Formeln untersuchen wir das NP-harte Problem, den empirischen Fehler von DNF-Formeln mit maximal k Konjunktionen zu minimieren. Wir schlagen einen Algorithmus basierend auf stochastischer lokaler Suche (SLS) vor, um das Problem effizient zu lösen. Danach wenden wir SLS innerhalb einer Methode zur strukturellen Fehler-Minimierung an, um kleine Regelmengen mit hoher Vorhersagegenauigkeit zu generieren. Als einen alternativen Ansatz verbinden wir SLS mit Ensembles, um den Abschätzungsanteil des Vorhersagefehlers zu minimieren und auf analytische Weise abzuschätzen. Im Falle der gewichteten Regelmengen entwerfen wir das neue konvexe Optimierungskriterium "Abstand minus Varianz" (engl. Margin Minus Variance, MMV), eine realisierbare Vereinfachung des praktisch nicht realisierbaren Problems, den empirischen Fehler zu minimieren. Für die Kapazitätsüberwachung leiten wir eine neue Ungleichung zur Dichteabschätzung her. Eine Implementation der vorgestellten Ideen, das RUMBLE-System, erzielt positive Ergebnisse in Experimenten mit Struktur-Aktivitäts-Beziehungen von kleinen Molekülen. Für multi-relationales Lernen, schließlich, entwerfen und verwenden wir ein theoretisches System, in dem verschiedene Lernsysteme anhand des Informationsflusses bewertet und verglichen werden können. Wir bewerten verschiedene Strategien auf analytische und experimentelle Art und Weise und stellen ein neues Regel-Generierungskriterium vor, das auf hohe Diversität in Regelmengen abzielt.

Acknowledgements

Many people think that writing a dissertation is a daunting, time-consuming, and difficult task, which needs to be well-planned and demands a high degree of discipline and diligence. This is not consistent with my experience. I found that writing a dissertation is a daunting, time-consuming, and difficult task, which never proceeds as one would expect, but is nonetheless a great adventure and generally a heck of a fun thing to do. I am very glad that my experience differs in this detail from the common expectation and I am thankful to the people who helped making my work so enjoyable. Without any doubts, I am most indebted to my advisor, Stefan Kramer. Stefan not only provided the guidance, encouragement, and support one would expect from an excellent mentor, he also managed to create a working environment where even a tight conference deadline proved to be a great challenge rather than a strenuous death march. Indeed, he was more than just an excellent mentor in many ways. Other advisors may specify the overall direction of research for a dissertation, but Stefan also encouraged me to find and follow my own research interests. Other advisors may only contribute their knowledge of the field, but Stefan also spent his personal time in countless discussions on exciting open questions, leading me to valuable inspirations and insights. I am deeply grateful for having had the chance to work with him.

Before coming back to Munich, I had a wonderful time in Freiburg. I would like to thank Luc De Raedt for his tremendous support and his valuable work on the papers he co-authored with me. It was a pleasure to work with the Freiburg group, most notably, Maren Bennewitz, Björn Bringmann, Kristian Kersting, Tapani Raiko, Cyrill Stachniss, Moritz Tacke, Sau Dan Lee, and Albrecht Zimmermann. In 2005, I skipped the cold winter in Munich and headed for a research visit to the University of Waikato in New Zealand. I am sorry for making some people envious and I am very grateful to my host Bernhard Pfahringer as well as Eibe Frank, Geoffrey Holmes and Peter Reutemann for the support, the discussions and the great time I had with them. Peter Bartlett kindly invited me to UC Berkeley. I am indebted to him and Ambuj Tewari for the interesting discussions and their valuable comments on my work. Special thanks go to my colleagues and co-authors. Lothar Richter provided his immense biological knowledge.

Caroline Friedel and Elisabeth Georgii did impressive work and were a joy to work with. Marianne Mueller shared a room with me and supported me on my teaching duties. Together with Markus Erlacher and Jörg Wicker they were responsible for the stimulating environment at TU München. Finally, I need to thank Eibe Frank, Johannes Fürnkranz, Nada Lavrač and Bernhard Pfahringer for commenting on drafts of the paper I wrote with Luc.

Some people contributed tremendously to this work, even though they would probably deny any involvement. This includes my parents, my brother, my friends, Iris Hochstatter, and, in particular, Verena Lauer, who had to suffer the most from my obsession with unfinished proofs for clearly wrong theorems.

Chapter 1

Introduction

This dissertation investigates rule learning as a statistical classification problem. In the introductory chapter, we describe our motivation for investigating statistical rule learning. We start with a typical rule learning scenario and motivate statistical rule learning from two perspectives: from a theoretical perspective it is interesting to gain more insights into the trade-off between comprehensibility and predictivity of models and to reason analytically about different approaches to rule learning. From a practical point of view, there are a range of interesting applications that can benefit from statistical rule learning. We close with an outline of the thesis.

1.1 Motivation

One of the most popular learning schemes in machine learning is rule induction. There is plenty of research on rule induction, some of it dating back as far as the 1960s (Michalski, 1969). However, unlike neural networks, SVMs and some decision trees learners, many established rule learning systems are hard to analyze from a statistical point of view. Before we give the motivation for the work in this dissertation, we describe a short example application, where the results of this thesis could be put to use.

1.1.1 A Typical Learning Scenario

Imagine a research lab in the quest for a drug against cancer. A typical approach to this endeavour is to apply thousands or millions of agents to cancerous cells and automatically test the tumor growth rate of the cells after the treatment. Typically, some agents will slow down tumor growth, some may actually boost tumor growth and many agents will have no effect. With those results the researchers have information on which *existing* agents are successful in which particular settings. However, given such a table of millions of results, it is a tremendous undertaking to extract knowledge that may help creating *new* effective drugs. One particular way to gather such knowledge is to let a computer program induce a *predictive model* from the experimental data, which relates the structure of the compounds to their efficiency. Once learned, such a model can predict the tumor growth inhibition rates of new potential drugs. By evaluating the model, the researchers may get new insights into which factors influence growth inhibition. Ideally, the researchers would like to find a model whose predictions are as accurate as possible and which can be easily understood and analyzed by humans. The problem of finding such a model is known as *regression*, if the quantity to predict is continuous, and as *classification*, if the task is to predict whether an instance falls into two or more distinct categories.

Designing computer programs that are able to output helpful models in scenarios like the illustrated one is part of machine learning. Of course, the described scenario gives rise to some obvious questions: First of all, we do not give any information on what exactly a model constitues and how it is represented. Researchers have come up with various ways to represent models and each representation has its own advantages and disadvantages. Second, it is not clear how one could ever find models that are good at predicting new, previously unseen instances. After all, we might be unlucky and the new instances might be very different from the ones we have seen in the training data. Statistical learning theory gives some insight into this and similar questions. Third, there are also computational issues involved. Automated methods can easily generate data sets with millions or billions of records, so it may take a considerable amount of time to process the data. It is therefore an important goal of machine learning to design fast algorithms whose runtimes scale well with the data set sizes.

1.1.2 The Trade-Off Between Comprehensibility and Predictive Accuracy

One particular way to represent a classification model is *sets of rules*. For instance, in the scenario above, the computer program could output the following set of if-then rules:

```
If the molecule contains an aromatic ring,
    predict class carcinogen.
If the molecule contains a bromide ion,
    predict class carcinogen.
If the molecule contains a chlorine ion connected to a carbon atom,
    predict class carcinogen.
Otherwise
    predict class non_carcinogen.
```

The interpretation of a set of rules is straightforward: the individual rules specify explicitly, which conditions an instance must fulfill in order to be classified into a specific class. As long as the rule set is not too large and the rules are not too complicated, it is easy to get a quick overview on how and why a rule set predicts in a certain way. For that reason, rule sets are considered to be among the best comprehensible and interpretable model representations in machine learning. To foster the comprehensibility of rule sets, rule learning has traditionally aimed at finding simple, that is, small rule sets. Since rule learning has a long tradition in machine learning, there are many efficient and fast algorithms for deriving small sets of rules. For this reason, rule learning is popular among practitioners, often as a quick way to gather initial insights into the data at hand.

On the other side, statistical machine learning has come up with many methods focusing mainly on *predictive accuracy* of models rather than comprehensibility. For example, there are algorithms for inducing *neural networks* (Haykin, 1999): in this representation, a number of artificial *neurons* are connected with weighted connections, often in a hierarchical structure. Usually, the structure and the neuron's activation functions are predefined by the user, and the weights are determined by the learning algorithm. Another popular approach are *support vector machines* (Cristianini and Shawe-Taylor, 2000). In this case, the classifier attaches weights to the training instances. For classification, a kernel is used to project the instances into a high-dimensional feature space, where the instance weights induce a hyperplane, which separates instances of two distinct classes.

Both, neural networks and SVMs are typically hard to interpret by humans, mainly because the models contain many weights and it is not obvious, how those weights interoperate to determine a prediction. However, in many practical applications, neural networks and SVMs have been shown to feature excellent predictive accuracy, often significantly outperforming rule learning approaches. So, if one aims for accurate prediction without interpretation, approaches from statistical machine learning appear to be better suited. If comprehensibility is more important than predictive accuracy, rule learning seems to be the better choice. This trade-off between *comprehensibility* and *predictive accuracy* can be seen in many cases. Sometimes, statistically motivated methods can be applied to improve a classifier's predictive accuracy at the expense of its comprehensibility.

For example, in Section 4.4, the predictive accuracy of the rule learner RIPPER (Cohen, 1995) was investigated on 35 datasets. RIPPER is known to induce small and comprehensible rule sets with comparably good predictive accuracy (Rückert and De Raedt, 2007). However, applying RIPPER as a base learner in a bagging ensemble (Breiman, 1996) improves the predictive accuracy in all but one cases. Sometimes, the gain is huge, such as 11.2% on the vehicle dataset. On the downside, bagged RIPPER derives models that consist of twenty different rule sets, hardly an easily interpretable representation. By framing rule learning as a statistical classification problem, we hope to shed some light on the nature of this trade-off. The insights gained by doing so might then be used to design novel rule learning systems and to allow the user to find a good compromise between comprehensibility and predictive accuracy for her specific application.

1.1.3 Statistical Analysis of Rule Learning

Early research on rule learning was constrained by the performance of the hardware that was available at that time. Consequently, researchers were concerned mainly with finding fast algorithms. As the problem of inducing rule sets from examples is at its core an NP-hard combinatorial optimization problem (Pitt and Valiant, 1988), most researchers used heuristics and pragmatic approximation approaches to derive rule sets in a reasonable time frame. In particular, the *sequential covering* or *separate-and-conquer* algorithm (Fürnkranz, 1999) proved to be a fast approximation scheme. Many traditional rule learning systems are based on this approach and differ mainly in the choice of heuristics and intermediate or postprocessing steps to avoid overfitting. Each iteration of the sequential covering algorithm depends on the preceding iterations and the applied heuristics. This, and the combinatorial nature of the underlying problem make it extremely hard to analyze most established rule learning algorithms theoretically. While one strategy is to focus on empirical investigations and visualization tools (see for instance Fürnkranz and Flach, 2005), it is comparably hard to argue about the advantages or disadvantages of certain approaches to rule learning. For example, the survey by Fürnkranz (1999) lists fourteen heuristics used in rule learning algorithms, but there are no systematic theoretical results on which heuristic excels under which circumstances. Instead of selecting a rule learner that fits to a particular problem, the user is obliged to test many different variants for suitability on her data.

This is in contrast to the situation in statistical machine learning, where many methods are derived from basic statistical principles and optimizing system performance is performed afterwards. Instead of heuristics, statistical methods often start with a common *induction principle* and differ mainly in the assumptions and techniques they employ while following the principle. This often allows a more principled way to discuss strengths and weaknesses of different methods. For example, one can show that *logistic regression* derives the best possible classifier, if the positive and negative instances are normally distributed (Hastie *et al.*, 2001). Also, recent theoretical results relate the performance of a SVM with the ability of its kernel to quantify the level of similarity between the instances (Balcan and Blum, 2006). This gives an a-priori criterion for choosing a kernel which is suited for a given learning problem. We hope that the theoretical results in this thesis allow for similar insights that enable a sound and convincing reasoning about the utility of some design decisions in rule learning.

1.2 Applications

Of course, theoretical results are of limited immediate value, if they can not be translated into practically useful methods. A main goal of this dissertation is thus the design and practical evaluation of statistically motivated rule learning systems. In the following chapters we will present various systems, such as SL^2 in Section 4.3, the rule ensemble in Section 4.4.2 and RUMBLE in Section 5.4. We evaluate those systems empirically on typical learning problems. We distinguish between *propositional* and *first-order* data representations. Propositional rule learning deals with data that can be stored in a single table, whose rows constitute training examples and whose columns represent attributes. The algorithms presented in the Chapters 4 and 5 deal with such propositional data. To allow the comparison of our results with other results in the literature, we evaluate the systems mainly on data sets taken from the UCI repository (Asuncion and Newman, 2007). The data sets in this repository cover a variety of different applications. Generally, we try to use all UCI data sets, which have been used in comparable empirical studies and which meet the requirements of the specific learning system (for example, if a system only supports two-class problems, etc.).

In Chapter 6 we deal with learning settings, where the instances are not represented as a single row in a table. Learning classifiers for such data is known as *multi-relational learning* or *first-order learning* in the more general case of a representation in first-order logic (Nienhuys-Cheng and Wolf, 1997; Lavrač and Džeroski, 1994; Lloyd, 2003). Since a theory in definite clausal first-order logic is essentially a conjunction of first-order rules, rule learning is particularly well-suited for this induction setting. For the empirical evaluation in this chapter we resort to learning problems similar to the one outlined in Section 1.1.1 above. The learning task is as follows: given data about the molecular structure of some compounds and the compounds' effectiveness (for instance as a drug), learn a model that predicts the effectiveness of new compounds. Learning tasks of this form are usually known as *structure activity relationship* (SAR). Obviously, SAR learning is highly relevant in modern medicinal chemistry and drug design, where automated experiments, for instance from combinatorial chemistry, have led to vast amounts of data that is almost impossible to analyze without the help of computers. As the molecular structure is essentially defined by the atoms and the bonds between the atoms, SAR is clearly a multi-relational learning task. For our evaluation, we use a couple of popular SAR data sets that have also been employed in the literature to evaluate other multi-relational rule learners, for instance kFOIL (Landwehr *et al.*, 2006).

1.3 Outline of the Thesis

The main aim of this thesis is to analyze rule learning from a statistical perspective. Before doing so, we recall the basic concepts of rule learning in Chapter 2 and describe the general framework of statistical machine learning we will apply later on in Chapter 3. Readers familiar with these topics might consider skipping the two chapters. Rule learning is introduced in Chapter 2: Section 2.2 gives an overview of the representation languages in rule learning, while Section 2.4 deals with the combinatorial and practical difficulties that are involved in finding good rule sets. In particular, we identify two main challenges: empirical risk minimization, that is, the task of finding rule sets that explain the given training data as well as possible, and *capacity control*, that is, the problem of avoiding under- and overfitting. In Chapter 3, we introduce the statistical approach to classification. We start with the basic framework in Section 3.2. Based on this, we sketch relevant parts of statistical learning theory, in particular the theory on empirical and structural risk minimization (Sections 3.4.1 and 3.4.2), ensemble theory (Section 3.5) and the Bayesian approach to classification (Section 3.6).

With these prerequisites, we can tackle the actual main task. As described in Section 2.2.1, there are different ways to combine rules into rule sets. One particular way is to build the disjunction of rules, so that a rule set is equivalent to a propositional formula in disjunctive normal form (DNF). In Chapter 4, we investigate learning DNF formulae from a statistical point of view. In Section 4.2 we dig into the problem of finding a preferably small DNF formula that misclassifies as few training instances as possible. We propose a randomized algorithm for the noise-free setting in Section 4.2.1, and an algorithm based on stochastic local search for the noisy setting in Section 4.2.2. Sections 4.3 and 4.4 deal with capacity control methods for overfitting avoidance. We test two approaches. The first one is based on structural risk minimization and a bias towards simple and small rule sets. An implementation of this method is evaluated empirically in Section 4.3.2. The second approach uses ensembles for capacity control. It is analyzed analytically (Section 4.4.3) and empirically (Section 4.4.4).

Representing rule sets as DNF introduces a certain combinatorial complexity into rule learning. A different way to combine rules to rule sets is by applying weights to the individual rules and predicting the class labels with a weighted voting procedure. This introduces the notion of a *margin* into rule learning. In Chapter 5 we discuss rule learning with margins. Again, we treat the two main challenges, empirical risk minimization and capacity control, in distinct sections. Section 5.2 compares three optimization criteria for empirical risk minimization: the 1-norm support vector machine, empirical margin, and margin minus variance (MMV). While the first criterion has been investigated extensively in the literature on support vector machines, it is unclear how to perform efficient capacity control for empirical margin and MMV. In Section 5.3 we take three analytical approaches to estimate the structural risk in rule learning settings. The first one is based on Rademacher penalties, the second one on the PAC-Bayesian theorem, and the third one describes a novel bound based on rule repository sizes. Finally, in Section 5.4, we describe Rumble, an implementation of the optimization criteria with capacity control, and evaluate it empirically in Section 5.5.

So far we have focused on propositional rule sets, where each instance assigns values to the attributes in a predefined set. In Chapter 6 we extend the previously covered approaches to the first-order case, where an instance can be described by predicates in first-order logic. After an initial motivation, we extend Rumble towards the multi-relational case in Section 6.2. Rule learning for first-order data poses the additional challenge of how to build rules that extract meaningful information from the available training data. To tackle this problem formally, we describe a generic framework in Section 6.3, which rates and categorizes multi-relational learning systems according to the type of information that is used during classifier construction. Based on this framework, we propose a new dispersion-based rule generation procedure in Section 6.4, which aims at small, but diverse and informative rule sets. Finally, in Section 6.5, we give an empirical evaluation of the multi-relational version of Rumble (Section 6.5.1), the framework (Section 6.5.2) and the dispersion-based rule generation procedure (Section 6.5.3).

Finally, in Chapter 7, we summarize the contributions of the dissertation in Section 7.1 and describe some promising directions for further research and open problems in Section 7.2.

Chapter 2

Rule Learning

In this chapter we recall basic concepts of rule learning. We start with a description of three different ways to combine single rules to rule sets: DNF formulae, decision lists and weighted rule sets. Then, we address the question on how to form (and represent) single rules. After these syntactic and semantic considerations, we deal with the problem of inducing good rule sets. We identify two main challenges in rule learning, namely the combinatorial complexity of finding rule sets with small training error, and the task of overfitting avoidance. Finally, we give a short survey of the algorithmic approaches that have been taken in the literature on rule learning.

2.1 Introduction

Learning sets of rules has a long history within machine learning. Some of the theoretical and algorithmic work on rule learning even dates back to the late 1960s, more than a decade before the term "machine learning" was made popular at the first machine learning workshop in 1980. For example, Michalski's AQ system (Michalski, 1969) applied *separate-and-conquer* strategies to rule induction in 1969 and Plotkin introduced *relative least general generalization* (Plotkin, 1971), providing the theoretical foundation for the ILP system GOLEM (Muggleton and Feng, 1992). Separate-and-conquer based approaches to rule learning became popular in the early 1990s. Especially in *Inductive Logic Programming* (ILP), where theories in first-order logic are derived, the representation with rules is much more natural than the use of competing representations such as decision trees. Since the late 1990s, research on inductive rule learning has declined, but never vanished. Quite the contrary, there appears to be an increase of interest in recent years, as advances in other fields of machine learning shed a new light on certain aspects of rule learning. For example, Friedman and Popescu's RuleFit system (Friedman and Popescu, 2005) is based on research in ensemble methods. Also, in 2004, a workshop on inductive rule learning (Fürnkranz, 2004) exposed a broad field of recent research on this topic.

Even though rule learning has a long tradition, there is no precise and general criterion of what constitutes a "rule learner", and the border to some other approaches such as ensemble methods or perceptron learning is blurry at best. As a general rule of thumb, a learning algorithm which outputs hypotheses containing rules of the form "if *condition* then *target=class*" can be considered to be a rule learning system. There are no established standards on what kind of conditions are valid for the antecedent of a rule. Often, the antecedent is a conjunction of "attribute=value" or "attribute is less (or more) than a threshold" tests, but some learning systems use more powerful tests, such as whole DNF formulae in LRI (Weiss and Indurkhya, 2000). More importantly, if an instance matches the conditions in two or more distinct rules, there are various methods to combine the rules to a consistent prediction. In the following section we describe some popular ways to combine single rules into consistent hypotheses.

2.2 Rule Set Representations

The established rule learning algorithms differ not only in the way they derive the rule sets, but also in the way they interpret the sets in order to make a prediction. In the following we present three popular ways to interpret rule sets. For this section, we deal mainly with propositional rule learners and we assume that the rule antecedents are conjunctions of literals.

2.2.1 DNF Rule Sets

For two-class problems, there is a natural and conceptually intriguing way to combine the rules so that the resulting hypothesis resembles a Boolean formula in disjunctive normal form (DNF). The main idea is to generate only rules with the (previously selected) positive class label in the consequent. Thus, an instance is classified as positive, whenever it matches the condition of at least one rule and negative otherwise. From a logical point of view, a rule set is thus a disjunction of rules. Usually, the rules' antecedents contain only conjunctions, as a rule with a disjunctive antecedent could easily be split into two rules. This means that a rule set is essentially a disjunction of conjunctions, that is, a DNF formula. There are a couple of advantages of this representation: first of all, each rule is valid and meaningful independently of the other rules in the set. This makes DNF rule sets easy to interpret and comprehend when compared to other hypothesis languages. For example, in linear classifiers it is often difficult to rate how each feature influences the prediction of an instance. Because of this compositional nature, DNF like rule sets are also often used by humans to define concepts. An entry in a dictionary "A key is either a notched and grooved metal implement to open locks, or a vital crucial element, or a button that is depressed to operate a machine" resembles the form of a DNF formula: $(IsNotched \wedge IsGrooved \wedge OpensLock) \vee (IsVital \wedge IsCrucial) \vee (IsButton \wedge OperatesMachine)$. Another advantage is that DNF rule sets extend naturally to the first-order case, if one uses first-order predicates in antecedents and consequents. Thus, many of the results on DNF rule sets are also valid for first-order rule sets. This is especially interesting in the case of large or infinite domains, as first-order theories can contain recursive definitions to express complex regularities. A broad range of theoretical results and practical applications on first-order rule learning has been originated in the field of Inductive Logic Programming (ILP). Strictly speaking, ILP systems usually output formulae in Horn clausal normal form. However, this is equivalent to DNF, because the usual Horn clausal representation "$[pos \leftarrow (a_{11} \wedge \ldots \wedge a_{1l})] \wedge \ldots \wedge [pos \leftarrow (a_{k1} \wedge \ldots \wedge a_{kl})]$" can be easily transformed into the DNF representation "$[pos \leftarrow [(a_{11} \wedge \ldots \wedge a_{1l}) \vee \ldots \vee (a_{k1} \wedge \ldots \wedge a_{kl})]$".

On the downside, it is difficult to extend DNF to the multi-class learning setting. The main problem is that an instance can meet the conditions of more than one rule. In this case it is not clear which rule should be used to make an prediction. In principle, one could generate rule sets with non-overlapping rule conditions (see for instance Segal and Etzioni, 1994), but this often leads to large and incomprehensible rule sets. Most practical rule learners resort to one of the rule set representations below or define an order on the target classes and split a multi-class learning problem in a number of one-vs.-rest learning problems (Cohen, 1995; Rifkin and Klautau, 2004). Another disadvantage of DNF rule sets is the fact that it is sometimes difficult to represent models that are "robust" in the sense that a small change in an instance does not lead to a completely different prediction. For example, it is easy to formulate a single rule that predicts the positive class for one specific instance x and the negative class for all others. If, however, one wants to have a robust model that also assigns the positive class to all "neighboring" instances that differ from x in at most one attribute, then the model needs to be way larger; it must contain at least one rule per attribute. Therefore, DNF rule sets are often not very well suited for noisy data. DNF rule sets are induced for example by PRISM (Cendrowska, 1987) and RIPPER (Cohen, 1995).

2.2.2 Decision Lists

A popular extension of DNF rule sets to the multi-class setting are *decision lists* (Rivest, 1987). The main idea is to order the rules in the set and to apply the rules according to this order during prediction. This means that one can safely mix rules with varying consequents as only the first matching rule is used for prediction. Decision lists require a *default rule*, that is, a rule that assigns a class if an instance does not meet any of the conditions in the other rules. Decision lists match very well with separate-and-conquer learning systems, because those systems induce more general rules first, and the majority of the prediction errors tend to happen with the small disjuncts (Weiss and Hirsh, 2000; Holte *et al.*, 1989), that is, the later rules which cover comparably few instances. Like DNF rule sets, decision lists generalize well to the first-order case, especially for recursive rules, where the base case is often required to precede the recursive case. Since decision rules can contain rules with arbitrary consequents, they can encode disjunctive concepts more elegantly than DNF rules with an inappropriately selected positive class. For example, consider Boolean attributes and a rule set that classifies an instance as positive whenever at least one of the attributes is true. A DNF rule set would require one rule per attribute, but a decision list can simply state a negative rule "if all attributes are false then assign the negative class" and have the default rule classify all other instances as positive. On the other hand this notational elegance is bought at the expense of having interdependent rules: the exact meaning of a rule depends on all preceding rules. Thus, unlike with DNF rule sets, one can not analyze the individual rules independently of the rest of the rule set. CN2 (Clark and Niblett, 1989) and PART (Frank and Witten, 1998) induce decision lists.

2.2.3 Weighted Rule Sets

Another approach to extending DNF rule sets towards the multi-class setting is assigning a weight to each rule. To predict a class for a new instance, the system evaluates all rules in the set and collects the weights of rules whose antecedent is met. Finally, it sums up the weights for each class and predicts the class with the largest sum of weights. This voting procedure resembles a model where each rule quantifies the influence of its antecedent to the target. Such a representation is well suited for many real-world phenomena, because there are often a multitude of factors that influence the target variable. Weighted rule sets avoid the "all-or-nothing" nature of decision lists and DNF rule sets; the prediction of the class is based on the contribution of all supporting rules, not only on one or the first matching rule. Thus, if a small perturbation in an instance leads to the erroneous violation or assertion of a rules' condition, the remaining rules can still overrule the incorrect vote and ensure a correct prediction. This makes weighted rule sets well suited for noisy domains. Also, weighted rule sets introduce the concept of a *margin* into rule learning. The margin of a prediction is the difference between the sum of weights for the winning class and the runner-up class. It can be seen as an estimate of the certainty of the prediction. If all rules vote for a particular class, the classifier appears to be much more certain about the prediction than if one class has only a slightly better vote than the remaining classes.

Even though the introduction of weights appears to be a minor extension of the previous settings, it introduces a wealth of new aspects. First of all, it is a strict generalization of DNF rule sets and decision lists.[1] A DNF rule set can be emulated by setting a constant weight to the positive rules and adding a negative default rule with a smaller weight. Likewise, if one assigns weights that decrease exponentially with the rule number, the weighted rule set is equivalent to a decision list (see Anthony *et al.*, 1995, for a proof). If, on the other hand, one allows arbitrary weights, a weighted rule set can be seen as a linear combination of binary valued features, so that many classical results from linear algebra apply. This can avoid some of the combinatorial peculiarities of DNF and decision lists. Even better, the research on learning linear classifiers immediately applies to weighted rule sets. Linear classifiers are one of the best investigated hypothesis classes in machine learning. They form the basis of perceptrons, support vector machines and ensembles methods. From a Bayesian perspective linear classifiers correspond to Naive Bayes (Jäger, 2003).

It is not obvious how to select good weights for the rules in a rule set. Early rule learners such as C4.5rules (Quinlan, 1993) and versions compute weights from the coverage of the rules. Later versions are usually based on ensemble theory. For instance, SLIPPER (Cohen and Singer, 1999) and LRI (Weiss and Indurkhya, 2000) are inspired by boosting, whereas RuleFit (Friedman and Popescu, 2005) relates to bagging. A major disadvantage of weighted rule sets is the fact that it is not very clear how and to which degree the rules interact during the prediction of new examples. Consequently, a large part of the RuleFit system deals with extracting meaningful characteristics from the rule set.

2.3 Rule Condition Representations

In the preceding section we discussed combining rules into sets and ignored questions concerning the construction of the rule conditions. We assumed that the rule antecedents contain conjunctions of literals or negated literals. Indeed, this is the case for the overwhelming majority of rule learning systems. This choice is obvious for DNF rule sets: since a rule set is just a disjunction of rules, it does not make sense to use another disjunction in the rule precedent. Conjunctions, however, form a nice counterpart, because they limit rather than extend the scope of a rule. This allows for an easy separate-and-conquer approach to fine-tune the coverage of the rule sets. Adding literals to the conjunctions in the rule conditions limits the coverage, while adding a new rule extends the coverage. Since every Boolean

[1]Of course, this is only true, if one applies the same restrictions on the number and types of literals that are used in the rule antecedents. If one allows one literal per rule condition, the class of Boolean functions that can be represented by decision lists is exactly the class of read-once weighted rule sets, see Eiter *et al.* (2002).

function can be expressed as a DNF formula, it is not necessary to allow any other operation in the rule antecedents. Similarly, decision lists with conjunctions as rule conditions can encode all possible Boolean functions.

Since weighted rule sets can choose the consequent of each rule individually, there is no difference in using disjunctions or conjunctions in the two-class setting. This is a simple consequence of the De Morgan duality $a \wedge b = \neg(\neg a \vee \neg b)$. For example, the rule `if` $a_1 \wedge a_2 \wedge a_3$ `then class=pos` is logically equivalent to the rule `if` $\neg a_1 \vee \neg a_2 \vee \neg a_3$ `then class=neg`. Thus, one can transform any two-class weighted rule set with conjunctive rule conditions into a version of the same size and form with disjunctive conditions. This is not the case in multi-class settings. However, it is possible to express every possible (multi-class) classifier as a decision list or weighted rule set with conjunctive rule conditions. Because of this and the elegance of the separate-and-conquer approach to rule learning, most rule learners use conjunctions in the rule antecedents.

The expressiveness of conjunctive or disjunctive rule conditions depends also on the form of used literals. For example, many rule learners allow only literals of the form *attribute=value*. This is not a restriction with two-valued attributes, because the two logical alternatives a and $\neg a$ can be expressed by *a=true* and *a=false*. It is a restriction, however, for attributes with three or more values, because one can not express "attribute takes the first or third value" in a conjunctive rule condition.[2] Theoretically, this restriction can be overcome by inducing several rules with slightly different antecedents, but in practice, the expressive power of a rule set depends very much on the way the information is encoded in the original data table and the predicates which are available for forming rule conditions.

This is true in particular in the case of first order rules, where the literals express relationships between objects. Furthermore, first order rules can also contain variables, so that the evaluation must take into account that each occurrence of a variable in the literals references the same object. This makes the evaluation of a rule much more difficult. Additionally, there is a vast amount of possible ways to combine literals, variables and constants into rule conditions, so that it is hopeless to enumerate and test all possible conditions during the learning process. Consequently, ILP systems often require some user-defined *language bias*, which specifies how to generate and refine rule conditions. Typically, the language bias specifies which literals can be used with which type of objects, and what kind of literals can be added to a condition that already references objects of certain types. For example, consider a database containing small molecules encoded with the two relations atom and bond. The language bias could specify the object types *atom*, *molecule* and *element*. Then a refinement operator could specify that, whenever a condition contains a variable A referencing an object of

[2]Such conditions are called *internal disjunctions* by Michalski (2004).

type *atom*, it can be extended by the two literals *bond(A, B), atom(B, El)*, where B is another atom and *El* is an element. The new literals demand that the atom A has a bond to a new atom B which is an element of type *El*. If this refinement operator is applied repeatedly, the resulting conditions describe substructures that may or may not occur in the database molecules.

Additionally to those fundamental considerations, there are also a set of pragmatic points to keep in mind. First of all, propositional and first-order rules might use attributes that contain continuous values, such as the temperature of an object. Since it is very unlikely that two instances feature the exact same values for this attribute, it is reasonable to use literals which test whether a value exceeds some threshold or if it lies in an interval. Finding reasonable thresholds or intervals can be a hard problem. Also, real world data often contains *missing values*. For example, a physician might sometimes skip a particular medical test during an examination, so that the corresponding field in the database does not contain any data. Modern rule learning algorithms provide ways to evaluate a rule in a reasonable way, even if the rule antecedent references the missing parts of an instance.

2.4 Challenges in Rule Learning

Before we can give a description of the algorithmic approaches to rule learning in Section 2.5, we describe the challenges and problems that need to be addressed in order to find good rule sets. As we will see, there are two main issues. First, one would like to find a small rule set that explains the training data well. Then, the induced rule set might be too specific to generalize well beyond the given training data, a phenomenon known as *overfitting*. We will see in Section 3.4 that these two points have their counterparts in statistical learning theory.

2.4.1 Combinatorial Complexity

In general, rule learning is concerned with inducing a rule set from a database of instances, that has a low misclassification error on unseen test data. This is essentially a statistical problem and we cover it as such in Section 3.4. For the scope of this section, we just need the following observation, and refer the reader to Section 3.4 for a more formal and principled discussion. Assume a learning algorithm is given a training set containing the same number of positive and negative instances. It is then asked for a rule set that explains the training data correctly. It turns out that this task is actually quite easy. For example, the system could just build a rule set where each rule covers exactly one positive training instance and all remaining instances are assigned to the negative class. Such a rule set has two disadvantages. First of all, it is very skewed, in that it assigns the positive class only to the positive instances it has already seen. In a sense it is nothing more than a rote learner. This is not a very good learning bias in most practical settings, because it does not generalize the regularities that appear in the training data. For example, it might be the case in the given training data that if a certain attribute has a certain value, the instance is very often positive. A rote learner would ignore such a pattern rather than taking advantage of it. Second, the size of such a rule set grows with the size of the training set, even though the regularities and patterns in the training set would allow for a much more compact representation. For example, if the training set contains the pattern described above, the rule set could classify a large number of positive instances correctly with just one rule, leading to a more *comprehensible* and compact representation. Intuitively, it seems to be likely that such a compact rule set generalizes better to new unseen instances.[3]

Thus, practical approaches aim at finding small rule sets, or—similarly—search for rule sets in a restricted hypothesis class, which contains only rule sets of a certain limited size. This is motivated by the considerations above, but also by a philosophical principle called *Occam's razor.* This principle basically states that, if two descriptions explain a phenomenon equally well, one should choose the simpler one (Domingos, 1999; Webb, 1996). For rule sets, "simple" is generally associated with "small", because smaller rule sets are easier to comprehend. So, the main problem we are faced with in rule learning is a combinatorial optimization task: given a set of instances, find the rule set in a restricted hypothesis class (containing only small rule sets) which assigns the correct target label to as many training instances as possible. Typical hypothesis classes in the literature on computational learning theory include

- k-DNF formulae: formulae in DNF with at most k literals per term (Valiant, 1984)
- k-term DNF formulae: DNF formulae with at most k terms (Pitt and Valiant, 1988)
- k-decision lists: decision lists with at most k literals per term (Rivest, 1987)
- k-dimensional perceptrons: weighted rule sets with at most k rules (Höffgen *et al.*, 1995)

Unfortunately, the problem of finding such a rule set can be extremely hard in the worst case. In particular, for the classes that restrict the number of rules in a rule set (that is, k-term DNF and k-dimensional perceptrons), the problem of finding a consistent rule set can be shown to be NP-hard

[3]From a statistical point of view, this is just an assumption, see Section 3.4 for a more principled discussion.

(Pitt and Valiant, 1988; Höffgen *et al.*, 1995). Even worse, a recent result shows that it is NP-hard to approximate the maximal agreement of a rule set with a training set to within some fixed constant (Ben-David *et al.*, 2003). Therefore, most rule learning algorithms resort to heuristics and greedy approximation strategies such as the separate-and-conquer approach (compare Section 2.5).

The fact that these algorithms tend to do quite well in practice indicates that the underlying learning problems are often not as hard as the worst-case complexity results suggest. This is certainly due to the fact that users of machine learning techniques generally aim at providing training sets that are representative and informative about the concept to be learned. But, even if the training sets are chosen at random, the average complexity of finding the best rule set is often way lower than the worst-case complexity. A recent study has investigated k-term DNF learning in the phase transition framework (Rückert, 2002; Rückert *et al.*, 2002). It appears that finding the best rule set with at most k rules for a random training set can indeed be extremely hard for certain settings of k and training set sizes, but if one increases the k by a comparably small amount, the resulting combinatorial optimization problems are often way easier. If the main goal is to find rule sets with high predictive accuracy, it is often not necessary to find the smallest possible rule set. Instead, it makes sense to find reasonably small rule sets while optimizing for other criteria that are known to affect predictive accuracy.

For instance, it has been shown that a comparably large number of prediction errors are made on *small disjuncts*, that is, rules that cover only a small number of examples (Weiss and Hirsh, 2000; Holte *et al.*, 1989). As these rules depend only on a small number of training examples, they are especially susceptible to noise and random fluctuations in the data. Small disjuncts are a problem in particular for separate-and-conquer algorithms, which generate large disjuncts first and are left with only a small number of training examples to generate new rules later on. If one uses weighted rule sets, optimizing for a large margin has also been shown to be effective to gain predictive rule sets. This criterion is motivated by the fact that weighted rule sets can be seen as hyperplanes that separate the instance space into two half-spaces, where one half-space contains the positive instances and the other half-space the negative ones. In this case it is reasonable to have a large distance from the instances to the hyperplane, that is, a large margin. Such a large margin classification is robust against a random perturbation of the data that moves an instance only to a small extent, because the margin is often large enough to accommodate for the effect. Large margin classification has been made popular by support vector machines, see for instance the book by Cristianini and Shawe-Taylor (2000).

Another criterion can be used, if a large number of unlabeled instances are available during the training stage, the so-called *semi-supervised* setting.

In such a case, one can direct the search for good rule sets in a direction that agrees well with the unlabeled data. For instance, it is sometimes reasonable to assume that the decision boundary separating positive from negative instances should not cross through an area which contains a lot of (unlabeled) instances, because it would split a naturally occurring cluster into two distinct areas. Semi-supervised learning is a field of its own. We refer to the survey by Zhu (2005) and the book by Chapelle *et al.* (2006) for more details. Of course, there are many possible heuristics that could be used to tailor the rule induction process to certain settings. These heuristics influence the *inductive bias* of a rule learning algorithm, that is, the preference of a learning algorithm towards certain hypotheses. In general, a learning algorithm performs well whenever its bias matches well with the problem setting at hand.

2.4.2 Overfitting Avoidance

One particularly important part of every rule learner's bias is *overfitting avoidance*. Generally speaking, overfitting describes the phenomenon that the training accuracy increases when the classifier gets larger and more complex during the learning process, but its predictive accuracy (as measured on a test set) decreases. Overfitting happens, whenever the induced model is expressive enough to incorporate noisy or random patterns which appear in the training data, but are not present in the underlying data generation process. In order to avoid overfitting one has to restrict the expressive power of a classifier, or, equivalently, the richness of the hypothesis class of which the classifier is taken from. On the other hand, if one uses only a very restricted hypothesis class, so that the induced classifier is very simple, it may not be able to catch valid structure in the underlying data and exhibit bad training and test accuracies. This is known as *underfitting*. Unfortunately, it is impossible to identify the "correct" classifier complexity only from the training data, because overfitting harms only the test error, not the training error.

Hence, many approaches to overfitting avoidance lay aside a certain fraction of the training data as a *validation set*. The actual classifier is then induced only on the remaining training instances and the independent validation set is used to estimate the level of overfitting. If the induced rule set turns out to overfit, then it can be easily simplified by erasing literals or whole rules until its predictive performance on the validation set is satisfactory. Such a process of cutting back a rule set in order to avoid overfitting is called *pruning*, and pruning with a validation set as described above is a *post-pruning* strategy, because the pruning is done as a separate step *after* the rule set induction. This is in contrast to *pre-pruning*, where the induction of the rule set is stopped early, often without consulting a validation set. In practice, there is a certain trade-off involved: Ideally, one would like to use as many instances as possible in the training step to ensure that the induced rule sets catches all important regularities. However, if one does not use a validation set and resorts to general heuristics instead, there is no accurate assessment of the degree of overfitting, and the induced classifier might feature bad predictive accuracy.

Different rule learning systems have tackled this problem in different ways. Early systems applied pre-pruning heuristics based on statistical tests, such as χ^2 in CN2 (Clark and Niblett, 1989), or on the minimum description length (MDL) principle (Rissanen, 1978) as in FOIL (Quinlan and Cameron-Jones, 1995). Post-pruning methods are usually based on a validation set. For example, *reduced error pruning* (REP) (Brunk and Pazzani, 1991) uses two thirds of the data for the *growing set* and one third for the *pruning set*. Later approaches intermix pre- and post-pruning. For instance, *incremental reduced error pruning* (IREP) (Fürnkranz and Widmer, 1994) prunes each rule individually, right after it has been induced. Since different growing and pruning sets are used for each rule, IREP can (potentially) make use of all instances for learning and pruning. Further refinements led to complicated systems such as RIPPER (Cohen, 1995), which combines MDL heuristics with a loop that iterates over an IREP-like induction step and a sophisticated pruning step. We refer to the survey by Fürnkranz (1997) for further details on pruning strategies.

It must be emphasized that the ability to overfit is a property of the induction algorithm rather than the induced classifier or the underlying hypothesis class. Even if the hypothesis class is very expressive and even if the induced classifier is very complex, it might be the case that the classifier was constructed not to optimize training set accuracy directly, but in a different way that is not susceptible to overfitting. This is true in particular for some ensemble methods. For example, in *bagging* (Breiman, 1996), the final classifier contains a large set of possibly complex base classifiers, which individually might very well overfit. However, since it uses a voting procedure for prediction, the resulting ensemble classifier in a sense averages over the different base classifiers so that the random fluctuations that lead to overfitting are canceling each other out.[4] This effect is neither due to the hypothesis class, which clearly contains classifiers that overfit, nor the voting procedure, as demonstrated by the fact that boosting (Schapire, 2003) is able to overfit even though it applies a similar voting procedure. The main difference is the fact that the induction algorithm for bagging averages over the base learners, while boosting is designed to optimize the margin and therewith the training accuracy. Both ensemble methods can be extended to generating weighted rule sets and they are actively employed, for example in the rule learning systems SLIPPER (Cohen and Singer, 1999) and

[4]See Section 3.5 for a more detailed and principled explanation on why bagging does not overfit.

RuleFit (Friedman and Popescu, 2005).

2.5 Algorithmic Approaches to Rule Learning

The preceding sections dealt with rule set representations and the main algorithmic challenges that are at the core of rule learning problems. In this section we are presenting possible strategies to address those challenges. The first part is devoted to the separate-and-conquer approach as it is the first and by far most popular strategy applied in rule learning. In the second part we present a small survey over different approaches that have been taken in recent work on rule learning.

2.5.1 Separate-and-Conquer

The main idea of *separate-and-conquer* or *set covering* for rule learning is straightforward: Assume one has found a rule that classifies a (preferably large) number of positive instances correctly. If we add such a rule to the rule set, the positive instances that are covered by this rule are explained correctly, and we do not need to care about them during the induction of further rules. Thus, it is safe to simply remove them from the training set and induce the new rules based only on the remaining, still unexplained instances. This procedure can be repeated until no positive instances are left. Similar considerations lead to an iterative procedure for generating a rule. In this case the goal is to exclude instances with a class label that is different from the one in the rules' consequent. If one extends a rule with a literal that excludes a (preferably large) number of those instances, further literals in the rule antecedent do not have to exclude the same instances. Consequently, we can discard those instances and iteratively generate new literals based on the remaining instances until the rule covers only instances whose class label agrees with the rule consequent.

Algorithm 1 gives the pseudo-code for the most basic incarnation of separate-and-conquer on a two-class problem. The algorithm contains two loops, one in Separate-And-Conquer for the iterative generation of rules and one in NewRule for the repeated extension of an initially empty rule antecedent with new literals. The actual implementation of the procedure NewLiteral depends on the rule learning system. Usually, it applies some heuristic to return the literal that discriminates best between positive and negative examples. The extension towards multiple classes is straightforward. The version of separate-and-conquer presented in Algorithm 1 generates DNF rule sets that are fully consistent with the training data. Of course, such a rule set will overfit hopelessly on most realistic data sets. Practical rule learners therefore often have less restrictive conditions in the two while loops. This leaves three places where the plain version of separate-and-conquer can be extended with heuristics to improve its practical fitness:

Algorithm 1 The Separate-And-Conquer algorithm. Given a set P of positive examples and a set N of negative examples (and assuming $P \cap N = \emptyset$), it returns a DNF rule set that classifies all instances correctly.

```
procedure Separate-And-Conquer($P$, $N$)
    $R \leftarrow \emptyset$
    while $P \neq \emptyset$ do
        $r \leftarrow$ NewRule($P$, $N$)
        $R \leftarrow R \cup \{r\}$
        $P \leftarrow P \setminus \{x \in P \mid x$ covered by $r\}$
    end while
    return $\bigvee_{r \in R} r$
end procedure

procedure NewRule($P$, $N$)
    $L \leftarrow \emptyset$
    while $N \neq \emptyset$ do
        $l \leftarrow$ NewLiteral($L$, $P$, $N$)
        $L \leftarrow L \cup \{l\}$
        $N \leftarrow N \setminus \{x \in N \mid x$ violates $l\}$
    end while
    return rule "if $\bigwedge_{l \in L} l$ then assign the positive class"
end procedure
```

- the heuristic in NewLiteral to evaluate new literals.
- the condition in the while loop in NewRule. This criterion determines how many negative instances a rule may label incorrectly.
- the condition in the while loop in Separate-And-Conquer. This criterion determines the number of positive training instances, which are not covered by the final rule set and are thus classified incorrectly.

For the first heuristic it is a natural choice to measure the difference between the number of positive instances and the number of negative instances covered by a rule. The literal whose addition to a rule maximizes this *accuracy* measure, is selected in NewLiteral. Accuracy has been shown to have some deficiencies, though, and various other heuristics have been proposed to overcome its limitations. The survey by Fürnkranz (1999) lists purity, information content, entropy, cross entropy, the laplace estimate, the m-estimate, the ls-content and correlation as measures that have been used in separate-and-conquer learning systems and research on new heuristics is still ongoing, see for example the work on weighted reduced accuracy (Todorovski *et al.*, 2000). The second and third heuristic influence the complexity of the rule set and can thus be seen as some form of pre-pruning.

We refer the reader to the survey of Fürnkranz (1999) for a list of heuristics used for complexity control and a classification of over forty different separate-and-conquer rule learning systems based on various criteria.

From a theoretical point of view, the separate-and-conquer algorithm can be seen as an *approximation algorithm* to the problem of finding the smallest rule set which is consistent with a given training set (Haussler, 1988). While separate-and-conquer is in general not able to find the set with the smallest number of rules, there is a guarantee that it does not perform "too bad" when compared to the optimal solution: If separate-and-conquer evaluates all possible rules in the inner loop, the problem of finding the smallest number of rules can be seen as a set covering problem and the outer loop of the separate-and-conquer algorithm is essentially the greedy approximation algorithm for solving this problem. It is well known that the greedy algorithm is a $1 + \ln n$ approximation algorithm, that is, if the smallest consistent rule set contains s rules, then it generates at most $s \sum_{i=1}^{n} \frac{1}{n} \leq s(1 + \ln n)$ rules, where n is the number of positive instances (Vazirani, 2003; Haussler, 1988). Of course, practical systems never evaluate all possible rules, so this guarantee does not apply in practice. Indeed, the empirical results presented in Section 4.3.2 indicate that some established separate-and-conquer rule learner induce unnecessarily large rule sets on some commonly used data sets. When it comes to predictive accuracy, the number of rules is not a critical measure. Separate-and-conquer's disregard of examples that are already covered by a previously generated rule, though, appears to be harmful for predictive accuracy, in particular on small disjuncts.

Separate-and-conquer approaches are widely employed in propositional and especially in first-order rule learning. The survey by Fürnkranz (1999) lists over forty different separate-and-conquer systems. Among the most popular propositional separate-and-conquer rule learning systems are CN2 (Clark and Niblett, 1989), RIPPER (Cohen, 1995), PART (Frank and Witten, 1998), and PRISM (Cendrowska, 1987).

2.5.2 Other Approaches

Of course, separate-and-conquer is by no means the only way to derive rule sets. In the following we give a list of some other approaches that have been taken in the machine learning literature. One of the earliest alternative strategies is to build a decision tree first, then pool the branches to all leaves of the tree as a set of rules. Of course, it is not sensible to keep all branches, so a post-processing step is required to select a subset of good rules and to determine their order. This is the strategy taken by C4.5rules (Quinlan, 1995), an adaption of the well-known C4.5 decision tree learner (Quinlan, 1993). Instead of post-processing an existing decision tree it is also conceivable to modify the *divide-and-conquer* strategy used in decision tree learning to output rules rather than trees. This has been investigated by Boström (Boström, 1995) for the first-order setting.

Both, separate-and-conquer and divide-and-conquer induce rule sets in a top-down fashion. Domingos' RISE system (Domingos, 1994, 1996) follows the opposite direction: it starts with a rote learning rule set where each rule covers exactly one instance. Then, it iteratively merges two similar rules into one more general rule until an accuracy-based stopping criterion is met. This method can be seen as a bottom-up rule learner, but also as an instance-based learning scheme, as it generates a "compressed" version of the instances in the training set. The bottom-up approach also avoids the problem of small disjuncts in separate-and-conquer rule learning.

As we have seen in Section 2.4, inducing a small set of rules which is consistent with a training set is essentially a combinatorial optimization problem. As such, it is amenable to all possible discrete optimization algorithms. One particular approach are *genetic algorithms* or, more generally, *evolutionary algorithms.* Those algorithms start with a random population of solution candidates. Then, they iteratively select candidates based on their fitness, that is, their performance on the task to be solved, and recombine or mutate the selected candidates to generate a new population. This strategy is motivated by the ability of evolutionary processes in nature to originate complex creatures that adapt well to their environments. Research on genetic algorithms for rule learning dates back to the 1980s (Smith, 1980; Holland, 1986). More recent research includes G-Net (Anglano *et al.*, 1998) and ECL (Divina and Marchiori, 2002). A survey by Divina (2006) gives an overview over recent work on evolutionary approaches to first-order rule learning.

Another combinatorial optimization strategy that has had remarkable success on NP-hard optimization problems is *stochastic local search* (SLS). Like genetic algorithms, SLS starts with a random solution candidate. It then iteratively evaluates the fitness of the neighboring candidates according to a predefined neighborhood relation and selects the best neighbor as a candidate for the next iteration. This choice is from time to time replaced by a random neighbor to escape local optima. SLS algorithms can be implemented very efficiently and they are among the best algorithms available to solve hard satisfiability problems, see for instance Hirsch and Kojevnikov (2005). SLS has been applied to rule learning by LERILS (Chisholm and Tadepalli, 2002). It is also featured in Chapter 4, where it is used in an investigation to find optimally small rule sets.

If run times are not a big issue, one can also apply brute-force methods and simply search through the hypothesis space for good rule sets. As the hypothesis space of all possible rule sets is too large to be searched exhaustively, those systems often look only for rule sets of some specific restricted subclass. Of course, there are also heuristics involved to assess predictive accuracy and to speed up the search. In the literature on machine learning, exhaustive enumeration algorithms have been explored for Brute

(Riddle *et al.*, 1994), BruteDL (Segal and Etzioni, 1994) and PVM (Weiss *et al.*, 1990). Brute uses a depth-bounded search for the best individual rules and combines the 50 best in a rule set. Its successor BruteDL searches for the best set of homogeneous (that is, non-overlapping) rules. PVM can be seen as a heuristic approximation to finding the best single rule. In ILP, Bratko's HYPER (Bratko, 1999) applies a best-first search strategy.

Exhaustive rule enumeration algorithms have also been investigated in the data mining community, where a huge amount of research has been undertaken on *association rule mining*. Researchers have begun to extend this work towards the setting of predictive rule learning. Conceptually, association rules are not concerned with the prediction of an unknown class label, but with the extraction of statistically significant regularities in large databases. A typical application of association rule mining is in market basket analysis, where algorithms such as *Apriori* (Agrawal and Srikant, 1994) can be applied to enumerate all rules of the form "if a customer buys item A and item B, it is likely she also buys items C and D", which meet certain significance criteria. It is obvious that a subset of those rules could be used as a basis for a classifier that predicts only one specific class. Work in this direction has been spawned by CBA (Liu *et al.*, 1998) and subsequent systems include CMAR (Li *et al.*, 2001) and CPAR (Yin and Han, 2003).

Another style of rule learning algorithms has been influenced by recent developments in ensemble methods. There are two main approaches to learning ensembles of rules. The first one is conceptually based on *bagging* (Breiman, 1996), the second on *boosting* (Schapire, 2003). In its original form, bagging is a technique to reduce the variance portion of the prediction error. To this end, a whole set of classifiers is induced on different bootstrap samples of the original data set. These classifiers are then combined in a simple voting scheme. In a sense, the resulting ensemble averages over the randomness, which is due to the instance selection in the bootstrap sample. Hence, it is more stable and less sensitive to variations that depend on the particular choice of the training set. Similar ideas can be adopted for rule learning, in particular, if the rule generation process is non-deterministic. An adaptation of random forests (Breiman, 2001) towards rule sets can be found in Section 4.4, also published in a recent paper (Rückert and Kramer, 2004b). A related approach based on *importance sampled learning ensemble* (ISLE) (Friedman and Popescu, 2003) is taken in RuleFit (Friedman and Popescu, 2005). Finally, Pfahringer *et al.* (2004) and Anderson and Pfahringer (2006) propose a system to build ensembles containing a large number of randomly generated rules.

Boosting stems from computational learning theory as a method to combine *weak learners* with a predictive accuracy slightly better than chance into ensembles of high predictive accuracy. The main idea is to iteratively induce new classifiers, where each classifier focuses on the training instances that have been misclassified by the preceding classifiers. Most boosting approaches assign a weight to each classifier quantifying its performance on the training set and perform prediction with a weighted voting procedure. As such, boosting appears to be a natural way to induce weighted rule sets and has been adopted quite early for rule learning. While SLIPPER (Cohen and Singer, 1999) uses a single rule learning procedure as a base classifier in a boosting algorithm, Lightweight Rule Induction (LRI) (Weiss and Indurkhya, 2000) generates ensembles of whole rule sets without weights.

Chapter 3

Statistical Machine Learning

This chapter gives a short overview over the parts of statistical machine learning that are relevant for the work in the following chapters. We describe how classification is generally framed in statistical learning theory and state basic results: the test set bound to rate the prediction accuracy measured on previously unseen data, the Bayes classifier as the theoretically best possible classifier, the question of consistency of a learning system in the limit, the no free lunch theorems revealing the theoretical limits of inductive learning, and the separation of the prediction error into an approximation and an estimation part. We describe why pure empirical risk minimization lacks some form of capacity control and we present structural risk minimization and regularization as remedies. Finally, we touch upon ensemble methods and Bayesian approaches to classification.

3.1 Introduction

The tasks that are adressed in typical machine learning applications deal very often with imperfect, noisy data and uncertain settings, where the available information is generally too sparse to draw justified deductive conclusions. Hence, many machine learning systems rely on statistical and probabilistic methods, which can express and assess the risks, randomness and probabilities of the involved events and decisions. This is especially true when it comes to insights on fundamental questions, such as: What kind of concepts can or can not be learned under certain circumstances? How many instances need to be observed to induce a classifier with a certain prediction performance? Consequently, modern theory on machine learning in general, and on classification in particular depend to a large degree on probabilistic and statistical modeling and reasoning.

Research in statistics has always been relevant for machine learning. Many of the early statistical methods for classification have been adopted and extended in machine learning. For example, Fisher's linear discriminant (Fisher, 1936), logistic regression (Hosmer and Lemeshow, 2000) and nearest-neighbor methods (Dasarathy, 1991) are now integral parts of machine learning. Early theoretical results on modern learning theory are rooted in the work on parametric estimation starting in the 1920s and non-parametric estimation in the 1950s and 1960s. On the computer science side, Gold introduced his *identification in the limit* paradigm (Gold, 1967) for learning languages. On the statistical side, most results on the prediction performance of classifiers were restricted to certain data-generation distributions, until Vapnik and Chervonenkis (1971) gave distribution-free guarantees for empirical risk minimization and Stone (1977) did so for instance-based classification rules. These results were concerned with finding classifiers that are provably predictive for large training sets, but often ignored the computational issues of deriving such classifiers. This changed with Valiant's *probably approximately correct learning* (PAC) framework (Valiant, 1984), which also considers the time complexity of computing classifier representations. Modern statistical learning theory has extended these results into various directions and numerous modern learning schemes are directly based on the underlying theoretical considerations.

A relatively recent development in learning theory is the area of ensemble methods. As ensemble methods essentially derive sets of weighted base classifiers, they are immediately applicable to rule learning with weighted rule sets. Historically, ensembles emerged roughly at the same time from two different backgrounds. The first one, boosting, has its roots in PAC learning. It is motivated by the goal to prove that strong PAC learning is equivalent to weak learning, that is, learning classifiers which feature a predictive accuracy only slightly larger than chance. To this end, Schapire (1990) came up with a first method to boost weak learners into strong ensembles. This methods were subsequently refined to overcome certain practical drawbacks and the research ultimately led to the highly popular AdaBoost algorithm (Freund and Schapire, 1997). The second prominent ensemble method, bagging, was introduced by Breiman in a seminal paper (Breiman, 1996). It is motivated by the goal of reducing the variance part of the prediction error of unstable predictors. We elaborate on both methods in more detail in Section 3.5.

A different approach to statistical inference has been followed by Bayesian statistics, where the focus is on distributions of classifiers rather than selecting a single best classifier. The main idea is to use Bayes' rule to calculate a posterior distribution on classifiers from a prior distribution and the training set. In machine learning, Bayesian methods have traditionally been more popular for unsupervised learning than for classification. We therefore give a short introduction in Section 3.6, but we will not apply Bayesian methods to rule learning in the subsequent chapters. However, it is sometimes very instrumental to view classification methods from a Bayesian perspective and we will do so, when applicable, in the later chapters.

3.2 A Statistical Framework for Classification

Before we can go into the details of statistical learning theory, we need to specify some terminology, frequently used definitions and assumptions that are shared by most approaches. We start with a typical single-relational learning scenario: A user observes some natural phenomenon, such as the occurrence of a plant disease at some agricultural site. He measures certain aspects of the phenomenon, for instance nutritional conditions, average solar irradiation, humidity, size of the plants, and whether or not a plant is diseased. From this data, the user wishes to learn how the measured aspects influence the phenomenon; in the plant example, how environmental factors and treatments affect the presence of the disease in a plant. In particular, the user wishes to make justified predictions regarding the phenomenon in future cases.

In statistical machine learning terminology, this problem can be framed as follows. First of all, to describe the phenomenon, we need a set of *attributes*, sometimes called *features* $\mathcal{A} := \{a_1, \ldots, a_m\}$. Each of the m attributes describes a distinct aspect of the observations. The *domain* of an attribute a_i, denoted by $dom(a_i)$, states which values can be observed for the attribute. Typical choices for domains are $\mathbb{R}$, $[0;1]$, $\mathbb{N}$, $\{\text{true}, \text{false}\}$, or some set of nominal values such as $\{\text{red}, \text{green}, \text{blue}\}$. Second, the individual observations are called *examples* or *instances.* The space of all possible instances is the *instance space* or *input space* $\mathcal{X} := dom(a_1) \times \ldots \times dom(a_m)$. For notational convenience, we denote an instance $x_i = (x_{i1}, \ldots, x_{im}) \in \mathcal{X}$ as an m-dimensional row vector, rather than a column vector. This enables us to write a set of n instances $\{x_1, \ldots, x_n\}$ as an $n \times m$ matrix X, where x_{ij} denotes the value of the jth attribute in the ith instance.

Furthermore we need to formalize the part of the observation we would like to make predictions about. This is called the *class label* or *target label*, or shorter *class* and *target* respectively. The target label for instance x_i is denoted by y_i. Its domain is the *class label space* or *output space*, denoted by $\mathcal{Y}$. The structure of $\mathcal{Y}$ determines the kind of statistical problem one has: if $\mathcal{Y}$ contains exactly two labels, it is *binary classification*, if it contains a finite number of nominal values, it is *multi-valued classification*, if $\mathcal{Y}$ is ordered, the learning problem is called *ordinal regression*, and if $\mathcal{Y}$ is $\mathbb{R}$ or a subset thereof, the setting is called *regression.*

With these definitions we have the tools to model the learning scenario as a statistical framework. First of all, we assume a fixed, but unknown distribution $D \sim (\mathcal{X}, \mathcal{Y})$ over labeled instances. This distribution models the probabilistic nature of the underlying phenomenon to be explored. The user draws a sample of n labeled instances $(X, Y) = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ according to D^n. This sample constitutes the training data that is given to a machine learning system to induce a predictive model. More formally, a *classifier* or *predictor* is a function $c : \mathcal{X} \to \mathcal{Y}$ from the instance space to the class labels. The space of all classifiers is denoted by $\mathcal{C}$, but usually, we will only deal with some subset of $\mathcal{C}$, which can actually be handled by a learning algorithm. Such a space of potential classifiers is typically called *hypothesis space* and denoted by $\mathcal{H}$. Finally, we may want to assess the quality of a classifier. To this end, we define a *loss function* or *cost function* $l : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$. This function quantifies the regret or the costs that we experience whenever an instance of true class y_i is classified as y'_i by a classifier. For most practical loss function, $l(y, y') = 0$ whenever $y = y'$ and $l(y, y') > 0$ otherwise. For (binary) classification problems, the most important and popular loss function is the *zero-one loss* $l_z(y, y') = \mathbf{I}[y \neq y']$, where $\mathbf{I}$[condition] denotes the indicator function, which is one whenever the condition is satisfied and zero otherwise. In the following, we assume the zero-one loss whenever no other loss function is explicitly specified.

We can use the loss to assess the quality of a classifier c: The *risk* or *error* is the expectation of $l(c(x), y)$ over some distribution of (x, y). There are three distributions that are of particular interest. The first one is to take the expectation over the unknown underlying distribution D: $\varepsilon_c := \mathbf{E}_{(x,y)\sim D}\, l(c(x), y)$. This quantity is called the *true error* (*true risk*), because it specifies the average error of c we can expect on new unseen data. If we are given a training set (X, Y), the second quantity is the *empirical error* (*empirical risk*) $\hat{\varepsilon}_c := \frac{1}{n}\sum_{i=1}^{n} l(c(x_i), y_i)$, which computes the accuracy of c on the training set. The empirical error is a random variable depending on D. Finally, if the learning algorithm outputs a classifier c, it is sometimes possible to draw another n' instances (X', Y') (usually from D) for an independent test set. The accuracy on a classifier on this test set is the *test error* (*test risk*), denoted by $\varepsilon_c^T := \frac{1}{n'}\sum_{i=1}^{n'} l(c(x'_i), y'_i)$.

From a statistical perspective, the main problem in classification is as follows. We would like to find a classifier c that predicts well on unseen data, that is, we would like to have ε_c^T minimized by c. Unfortunately, at learning time we have neither access to the test data, nor to the underlying distribution D, so we can not optimize ε_c^T or ε_c directly. The only information we have is the training set (X, Y). So, the obvious questions in this setting are: How does optimizing for $\hat{\varepsilon}_c$ relate to optimizing the true and test risks? In particular, can we still find a classifier that performs well on the test data? If no, which additional assumptions are necessary to enable successful prediction? If yes, can we give performance guarantees, for example depending on the size of the training set? Are those guarantees valid, if we restrict the c to a specific hypothesis class, such as certain types of rule sets?

Before we give principled answers to those questions in the next sections, there are two obvious points we need to address. First, it is obviously impossible to learn well if the training data is not related to the test data, for example, because the test set it drawn from a completely different distribution than the training data. In the following we therefore assume that training, test and validation data is drawn from the same distribution D.

This assumption is sometimes violated and learning in such a setting is called *concept drift* (Widmer and Kubat, 1996). In principle, one could extend most results to the case where the test data is drawn from a slightly different distribution, but in practice there is rarely enough information to quantify the extent to which the test distribution differs from the training distribution, hence the stronger and more practical assumption.

Second, even if training and test data are drawn from the same distribution, learning can be impossible, if we allow dependencies between the individual instances. For example, if the drawing of an instance depends on the preceding instances, one could have the effect that the sampling process gradually shifts from one part of the instance space to another. Then, in the worst case, the training data may come from a completely different part of the instance space than the test data. To avoid this pitfall, we assume in the following that all (training and test) instances are drawn independently from each other. Again, this assumption is too strong in some applications, and the requirement of complete independence can be somewhat relaxed without losing most of the important results, see for instance Ryabko (2006). These two assumptions are well known as the assumption of *independently and identically distributed* (i.i.d.) data.

3.3 Basic Results

Let us start to address the questions posed in the preceding section. First of all, we are interested in determining a classifier c from the training data that has low test error ε_c^T. Assume we already found a good classifier c. Because of the i.i.d. assumption the test set is drawn from D as well, so the expected test error is $\mathbf{E}\,\varepsilon_c^T = \varepsilon_c$. Since the examples in the test set are drawn i.i.d. as well, the question of wether or not a particular test instance is classified correctly is equivalent to the toss of a biased coin where the probability of observing head is given by the true error ε_c. Thus, the probability of observing exactly k errors among the n' test instances is distributed binomially:

$$\mathop{\mathbf{Pr}}_{(X',Y')\sim D^{n'}}\left[\sum_{i=1}^{n'} l(c(x_i'), y_i') = k\right] = \binom{n'}{k}\varepsilon_c^k(1-\varepsilon_c)^{n'-k}$$

This has an important implication: The test error is concentrated around its mean, the true error ε_c. We have only two ways to influence the distribution of the test error. We can increase the number of test examples to decrease the distribution's variance, so that the test error is tighter concentrated around its mean. Or, we can choose a different classifier c that gives rise to a smaller true error ε_c, which in turn leads to a smaller test error. Thus, we can reformulate the original problem as follows. Instead of aiming for a classifier with low test error, we are interested in determining a classifier c

from the training data that has low true error ε_c. Then, the error on any new test set is automatically distributed around ε_c and the deviation of the test error from the true error just depends on the test set size.

3.3.1 The Test Set Bound

In practice, we can never know the true error, because we do not know the underlying distribution D. However, we can estimate it from a (preferably large) test set. Assume we have a test set of n' instances and we have observed $k = n'\varepsilon_c^T$ misclassifications among those instances. Since we know that the distribution of the test error is binomial, we can calculate the largest true error that gives rise to the observed test error. To do so, we just need to upper-bound the tail of the binomial distribution.

Definition 3.3.1.

$$\overline{Bin}\big(n', k, \delta\big) := \sup_{\varepsilon} \left\{ \varepsilon \,\middle|\, \sum_{i=0}^{k} \binom{n'}{i} \varepsilon^i (1-\varepsilon)^{n'-i} \geq \delta \right\} \tag{3.1}$$

Thus, $\overline{Bin}(n', k, \delta)$ denotes the largest true error ε_c so that the probability of having at most k misclassifications among the n' test instances is greater than δ. The following bound on the test set follows immediately:

Theorem 3.3.2 (Test Set Bound (Langford, 2005))**.** *For all $\delta > 0$ and all classifiers c with test error ε_c^T on a test set of n' instances, it is the case that*

$$\mathop{\mathbf{Pr}}_{(X',Y')\sim D^{n'}} \left[\varepsilon_c \leq \overline{Bin}\big(n', n'\varepsilon_c^T, \delta\big)\right] \geq 1 - \delta \tag{3.2}$$

Thus, if we observe a test error ε_c^T on a test set of size n', then, with error probability δ over the choice of the test set, the true error is smaller than the tail inversion $\overline{Bin}(n', k, \delta)$. This tail inversion can not be calculated analytically, because one needs to take the supremum in (3.1). Often it is more convenient to use the following easily computable upper bound in closed form:

$$\overline{Bin}\big(n', n'\varepsilon_c^T, \delta\big) \leq \varepsilon_c^T + \sqrt{\frac{\varepsilon_c^T \ln\frac{1}{\delta}}{n'}} + \frac{\ln\frac{1}{\delta}}{n'} \tag{3.3}$$

This inequality demonstrates the approximate deviation of the test error from the true error one has to expect on test sets: If the actually observed test error is near zero, the second term on the right hand side becomes negligible, so the deviation is essentially determined by the third term $O(1/n')$. Otherwise, the difference between test and true error is dominated by the second term and thus $O(1/\sqrt{n'})$.

3.3.2 Bayes Classifier and Consistency

In the preceding section we have seen that in order to gain a small error on new data, we need to find a classifier that minimizes the true error. Unfortunately, the true error cannot be computed, because we do not know the underlying distribution D. The only quantity we have at our disposal is the training set (X, Y). Therefore, the main questions in learning theory deal with how to make use of this information in order to find classifiers featuring a low true error. Before we can consider specific settings and hypothesis classes, it is natural to ask about the best classifier one could possibly find. We demand that a classifier is a deterministic function $c : \mathcal{X} \to \mathcal{Y}$, but the underlying distribution might be non-deterministic in the sense that it could assign different class labels with various probabilities to one and the same instance x. Thus, even the best classifier will have a non-zero true error, if the conditional probability $\mathbf{Pr}[Y = y|X = x] \notin \{0, 1\}$ for some class y and some instance x with $\mathbf{Pr}[X = x] > 0$. More formally, assume a two-class classification setting with $\mathcal{Y} = \{0, 1\}$ and define

$$\eta(x) := \mathbf{Pr}[Y = 1|X = x]$$

If we know η, we can construct a classifier that always predicts the most likely class label for each x:

Definition 3.3.3 (Bayes Classifier)**.**

$$c^*(x) := \begin{cases} 1 & \textit{if } \eta(x) \geq 1/2 \\ 0 & \textit{otherwise} \end{cases}$$

The expected error of this Bayes classifier is the *Bayes error*.

Definition 3.3.4 (Bayes Error)**.**

$$\varepsilon^* := \mathop{\mathbf{E}}_{x \sim D} \left[\min \left\{ \eta(x), 1 - \eta(x) \right\} \right] = \varepsilon_{c^*}$$

It is fairly easy to show that the Bayes classifier is the classifier with the smallest possible true error:

Theorem 3.3.5 (Optimality of the Bayes Classifier, (Devroye *et al.*, 1996))**.**

$$\varepsilon^* = \inf_{c:\mathcal{X}\to\mathcal{Y}} \mathop{\mathbf{E}}_{(x,y)\sim D} \left[l(c(x), y) \right] = \inf_{c:\mathcal{X}\to\mathcal{Y}} \varepsilon_c$$

Again, there is no way to compute ε^*, because it depends on the unknown distribution D, but it serves us as a theoretical lower bound when comparing the risks of classifiers induced by practical learning algorithms. The worst possible Bayes error is 0.5, which is achieved for the distribution, where $\eta(x) = 0.5$ for all possible x. In this case, the target label is independent of the instances and doing better than chance is not possible. If the Bayes error is zero, the class label is a deterministic function $f : \mathcal{X} \to \mathcal{Y}$ of the instance, so that $y = f(x)$ for all (x, y). In this "noise free" setting a classifier must be consistent with every single training instance in order to be considered for prediction. Finding such a classifier is considerably easier than inducing a good classifier in the general case, where $\varepsilon^* > 0$. We will demonstrate this later in Section 4.2 for DNF rule sets, where learning in the noise free setting (Section 4.2.1) is more efficient than in the general setting (Section 4.2.2).

It is clear that there is no learning algorithm that can identify the Bayes classifier from a finite training set; after all, one might be unlucky and draw a training set that is not representative of the underlying D. In this case any learning algorithm will have insufficient data to induce the Bayes classifier exactly. However, as the training set gets larger and larger, the empirical distribution of the training set converges to the true distribution. This might raise the hope that one can construct a learning algorithm whose classifiers reach the Bayes accuracy *in the limit*, that is, when the training set size approaches infinity. More formally, let $A : (\mathcal{X} \times \mathcal{Y})^n \to \mathcal{C}$ denote a learning algorithm, which outputs a classifier $c \in \mathcal{C}$ when given a training set of size n. Now consider the case where we start with a single training example and draw new examples step by step. For each training set size $n = 1, 2, 3, \ldots$, the algorithm outputs a new classifier $c_1, c_2, c_3, \ldots$. The following definition makes the notion of "reaching the Bayes risk in the limit" precise:

Definition 3.3.6 (Consistency). *An algorithm A is* consistent*, if*

$$\lim_{n\to\infty} \varepsilon_{c_n} = \varepsilon^*$$

It is universally consistent*, if it is consistent for any arbitrary underlying distribution D.*

Can we come up with a consistent or even universally consistent algorithm? Many practical learning algorithms are clearly not universally consistent, because they induce only classifiers in a restricted hypothesis class $\mathcal{H} \subset \mathcal{C}$. Thus, they are consistent, if $\mathcal{H}$ contains the Bayes classifier, but they are generally not, if one selects a distribution D so that $\eta(x) = c(x)$ for some $c \in \mathcal{C} \setminus \mathcal{H}$. Learning algorithms, which select hypotheses from a restricted hypothesis space $\mathcal{H}$ are generally only expected to be consistent in the sense that they select the best hypothesis in $\mathcal{H}$, when given an infinite amount of raining data. We refer to Section 3.4.1 for a discussion of this setting. On the other hand, one could design an algorithm which chooses a classifier from a nested sequence of hypothesis classes $\mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3, \ldots$ of increasing size. If $\lim_{i\to\infty} \mathcal{H}_i = \mathcal{C}$ and if the algorithm chooses a classifier from $\mathcal{H}_n$ for training sets of size n, it would be possible that such an algorithm is universally consistent.

The question on whether there exists a universally consistent learning algorithm was answered affirmatively by Stone (1977), who showed that the k-nearest neighbor algorithm is universally consistent for training sets of increasing size $n \to \infty$, if $k \to \infty$ and $k/n \to 0$. Indeed, Stone's original result was more general and proves the universal consistency of related learning algorithms such as instance-based histogram or kernel approaches, too. We do not state the result, because it is of little relevance for rule learning, but refer to Devroye *et al.* (1996) for a detailed treatment. In a sense, consistency is a minimal requirement one would demand from a learning algorithm, because it ensures that adding new training instances improves the predictive accuracy. However, we will see in the next section that there is generally no guarantee about *how much* predictive accuracy benefits from a certain number of instances. Therefore, it is of little relevance in practice, where only finite amounts of training data are available.

Are rule learning algorithms consistent? Most practical rule learning systems are certainly not consistent; for instance, in separate-and-conquer approaches, typical overfitting avoidance heuristics impose a fairly strong bias towards rule sets that prefer large disjuncts over small disjuncts. However, if we assume that all attributes of the instance space are binary valued, then rule sets can represent all possible classifiers and results from Vapnik-Chervonenkis in Section 3.4.1 indicate that the original separate-and-conquer approach is consistent, because it minimizes the empirical risk. If one assumes that the instance space is an m-dimensional vector space, such as $\mathbb{R}^m$, the consistency of a rule learning algorithm depends crucially on the algorithm's way of deriving the rule conditions on the numerical attributes. Again, a rule learning algorithm can be designed to be consistent. For instance, the histogram learning algorithm can be seen as a rule learning algorithm with intervals of constant size in the rule antecedents. As it meets the conditions of Stone's theorem directly, it is universally consistent. Also, many results on tree classifiers (Devroye *et al.*, 1996) extend naturally to rule sets. However, many practically used heuristics introduce biases that make learning algorithms non-consistent.

3.3.3 No Free Lunch

In practical applications there is only a limited amount of training and test data. Therefore, one is generally more interested in results about the predictive accuracy of classifiers on finite data sets rather than asymptotic results in the limit. In particular, it would be helpful if one could estimate an upper bound on the true error of a classifier depending on the training data and the learning algorithm. Such a bound would enable a user to derive classifiers with a guaranteed predictive performance and possibly lead to valuable insights into the characteristics of the used learning approach. Ideally, we are interested in bounding $\varepsilon_c - \varepsilon^*$, the difference between the true error of a classifier c and the Bayes error. Unfortunately, one can show that there is no way to bound this quantity based on finite amounts of training data without making any assumptions about the underlying distribution D (Devroye *et al.*, 1996; Wolpert, 1996). This rather disappointing negative result can be formulated in different ways and is generally known as the *no free lunch theorem.* In the following we will present it in two different flavors; first as a result on the slow rate of convergence of the empirical risk to the true risk, and then from a Bayesian perspective as the equivalence of all possible learning algorithms under conservative assumptions on the selection of learning problems.

The first result is due to Devroye (1982) and states that for each training set size n and learning algorithm A there is a distribution that gives rise to a true error arbitrarily close to pure chance.

Theorem 3.3.7 (Bad predictivity on finite samples (Devroye *et al.*, 1996))**.** *Let $\epsilon > 0$ be an arbitrarily small number. For any training set size n and learning algorithm A, there exists a distribution on (X, Y) with Bayes risk $\varepsilon^* = 0$ and true error ε_c of the classifier c output by A such that*

$$\varepsilon_c \geq \frac{1}{2} - \epsilon$$

This theorem states that the *finite sample performance* of an algorithm can be extremely bad for some distributions even though the algorithm achieves the Bayes risk asymptotically. Thus, there is no guarantee that a certain finite training set size, regardless of how large it may be, leads to classifiers with reasonably low true error. The Bayes risk is achieved only in the limit. The theorem is based on a fixed algorithm A and a fixed sample size n. It does not exclude the possibility that some algorithms feature a better universal rate of convergence than other algorithms. The following stronger result indicates that this hope is not justified; there are distributions leading to arbitrarily slow convergence for any algorithm.

Theorem 3.3.8 (Slow rate of convergence (Devroye *et al.*, 1996))**.** *Let $\{a_n\}$ be a sequence of positive numbers converging to zero with $\frac{1}{16} \geq a_1 \geq a_2 \geq \ldots$. For every learning algorithm A there exists a distribution on (X, Y) with Bayes risk $\varepsilon^* = 0$ so that the algorithm A induces a sequence of classifiers $c_1, c_2, \ldots$ with true error $\varepsilon_1, \varepsilon_2, \ldots$ and for all $i > 1$*

$$\varepsilon_i \geq a_i$$

So, all possible learning algorithms are equal in the sense that for any algorithm, there is a specific hard learning problem which gives rise to arbitrarily low convergence of the prediction risk. This means there is no universal distribution-free guarantee. To show that a specific algorithm works well, one needs to make some assumptions on the distribution D of the learning problem. If the assumptions are met, the algorithm works well, if not, it can be arbitrarily bad. Of course, a different algorithm A_2 might work well on those learning problems on which algorithm A_1 fails. This is a significant insight, because it means that rather than looking for the "universally best" learning algorithm, one should search for an algorithm whose bias matches well with the learning problem at hand. This requires some a-priori knowledge about the learning problem one is faced with. This fact motivates the term "no free lunch". When designing a learning system, it is thus important to provide a flexible and easy-to-use mechanism for specifying and modifying the system's bias. We will deal with such considerations later, for instance in Sections 5.4 and 6.2.

Another very instructive way to look at this phenomenon is from a Bayesian perspective, a view that has been made popular by Wolpert (1994). In this framework, the importance of a good match between the learning system's bias and the underlying learning problem can be formalized quite nicely. There are three essential differences to the framework described so far. First, Wolpert assumes a noise free setting, so that the true class is a deterministic function of the instance. This function is denoted by $f : \mathcal{X} \to \mathcal{Y}$ so that for all $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y} : y_i = f(x_i)$. Assuming any arbitrary distribution D on unlabeled instances, we denote the distribution of instances drawn from D and labeled according to f by D_f. Second, rather than dealing with the true error of a classifier, one evaluates the *off-training set error*, that is, the expected cost of misclassification on instances outside the training set. Formally, for a finite instance space $\mathcal{X}$:

$$\varepsilon_c^{\mathrm{OTS}} := \frac{\sum_{x \in \mathcal{X} \setminus X} \mathbf{Pr}\left[X = x\right] l\big(c(x), f(x)\big)}{\sum_{x \in \mathcal{X} \setminus X} \mathbf{Pr}\left[X = x\right]}$$

The use of the off-training set error is a natural choice in the noise-free setting, because one already knows the true label of a training instance, anyway. Instead, one is interested in predicting the labels of the new, unseen instances outside the training set. Third, the space of all possible learning problems is the set of all possible functions from $\mathcal{X}$ to $\mathcal{Y}$, that is, the space of all classifiers $\mathcal{C}$. Since $\mathcal{X}$ is finite, $\mathcal{C}$ is finite as well. This allows us to easily compute the off-training set error of a learning algorithm averaged over all possible learning settings. More realistically, we may assume that some learning problems are more likely than others and assume a prior probability distribution over $\mathcal{C}$. Since we do not know the correct prior in practice, we can compute the off-training set error averaged over all possible prior distributions. The following theorem does so for two arbitrary algorithms A_1 and A_2.

Theorem 3.3.9 (No Free Lunch, (Wolpert, 1996)). *Assume a fixed training set size n and two algorithms A_1 and A_2. Let $\mathcal{P} := \{P : \mathcal{C} \to [0,1] | \sum_{f \in \mathcal{C}} P(f) =$*

$1\}$ be the space of all probability distributions on $\mathcal{C}$. Denote the expected off-training set error of the classifier induced by algorithm A_i on the training set X drawn from the learning problem $f \in \mathcal{C}$ by $\varepsilon_i^{\mathrm{OTS}}(f, X)$. Then, for any distribution D on unlabeled instances:

$$\sum_{f\in\mathcal{C}} \mathop{\mathbf{E}}_{X\sim D_f^n} \varepsilon_1^{\mathrm{OTS}}(f,X) = \sum_{f\in\mathcal{C}} \mathop{\mathbf{E}}_{X\sim D_f^n} \varepsilon_2^{\mathrm{OTS}}(f,X)$$

$$\int_{P\in\mathcal{P}} \mathop{\mathbf{E}}_{f\sim P} \mathop{\mathbf{E}}_{X\sim D_f^n} \varepsilon_1^{\mathrm{OTS}}(f,X)dP = \int_{P\in\mathcal{P}} \mathop{\mathbf{E}}_{f\sim P} \mathop{\mathbf{E}}_{X\sim D_f^n} \varepsilon_2^{\mathrm{OTS}}(f,X)dP$$

Also, for a fixed training set X of size n:

$$\sum_{f\in\mathcal{C}} \varepsilon_1^{\mathrm{OTS}}(f,X) = \sum_{f\in\mathcal{C}} \varepsilon_2^{\mathrm{OTS}}(f,X)$$

$$\int_{P\in\mathcal{P}} \mathop{\mathbf{E}}_{f\sim P} \varepsilon_1^{\mathrm{OTS}}(f,X)dP = \int_{P\in\mathcal{P}} \mathop{\mathbf{E}}_{f\sim P} \varepsilon_2^{\mathrm{OTS}}(f,X)dP$$

In other words, the theorem states that A_1 and A_2 lead to the same off-training set error when averaged over all possible learning problems or over all possible prior distributions on learning problems. This remains to be the case even if one restricts oneself to the case of a fixed known training set X. Thus, without any assumptions on the learning problems f or, equivalently, a prior distribution over $\mathcal{C}$, there is no justification for preferring one algorithm over the other. While the result paints a rather pessimistic picture of a learning algorithm's ability to generalize well, the situation is not so bad in practice. The no free lunch theorems depend crucially on the fact that for any learning algorithm one can construct distributions on which the algorithm performs arbitrarily bad. However, most of these distributions are rather awkward and occur rarely, if at all, in practice. For example, when $\mathcal{X} = \{0,1\}^m$ and $x(j)$ denotes the j component of x, the parity function $f : x \mapsto (\sum_{j=1}^m x(j)) \bmod 2$ is known to be hard to learn by typical rule learning algorithms. Devroye *et al.* (1996) give an awkward learning problem for $\mathcal{X} = \mathbb{R}$, where $f(x) = 1$ if x is rational and $f(x) = 0$ if x is irrational, so that it is impossible to check $f(x)$ for a given x. Without doubt, such learning problems occur rarely in practice. Even if they do, a well-informed user might choose to use a different representation and, for instance, add an attribute that is related to the parity or the rationality of an instance, so that the problem is easier to learn.

3.3.4 Approximation and Estimation Error

In a sense, the crux of classification in machine learning is to find algorithms whose biases match well with the learning problems that are frequently encountered in practice. This is often more an art than a science. In particular, it is difficult to adjust how strong the bias should be. Informally, if a system's bias is very strong, it performs extremely well on a small set of learning problems, but badly on other problems. If it has a weak and broad bias, it might perform well on average, but the induced classifier depends to a large degree on the training set so that its performance varies greatly with the random peculiarities of the training sets. This phenomenon can be described in various ways, depending on the learning setting and the way the bias is encoded.

For regression problems, the *bias-variance decomposition* is a natural way to formalize the concept. Since in regression we have $\mathcal{Y} = \mathbb{R}$, the zero-one loss $l_z(y, y') = \mathbf{I}[y \neq y']$ does not apply. The most popular loss function for regression is the *squared loss* $l_2(y, y') = (y - y')^2$. Let us assume the usual setting, where a classifier c is induced from a training set (X, Y) drawn from D. Then, assume a fixed labeled test instance (x_0, y_0). We denote the expected squared loss error of c on the test instance by $\varepsilon_c(x_0) := \mathbf{E}\, l_2(c(x_0), y_0)$. Since we assume the noisy setting, where y_0 is a random variable rather than a deterministic function of x_0, the random variable $\varepsilon_c(x_0)$ depends on the training set and the class label y_0. Now, $\varepsilon_c(x_0)$ can be written as:

$$\begin{aligned}
\varepsilon_c(x_0) &= \mathbf{E}\left[\big(y_0 - c(x_0)\big)^2\right] \\
&= \mathbf{E}\left[\big(y_0 - \mathbf{E}[y_0]\big)^2\right] + \Big(\mathbf{E}[y_0] - \mathbf{E}\left[c(x_0)\right]\Big)^2 + \\
&\quad \mathbf{E}\left[\Big(\mathbf{E}\left[c(x_0)\right] - c(x_0)\Big)^2\right] \qquad (3.4)\\
&= \text{Noise} + \text{Bias}^2 + \text{Variance}
\end{aligned}$$

In this decomposition, the first term is the expected deviation of the target value y_0 from its mean, that is, the unavoidable Bayes error, caused by noise in the underlying distribution. The second term is the squared difference between the expected target value and the expected prediction. This quantity depends neither on the (random) training set nor on the (random) target value, but only on the systematic error that is made on average by the learning system. It therefore quantifies the bias of the learning algorithm with regard to D and x_0. Finally, the third term denotes the expected deviation of $c(x_0)$ from its mean. It depends on how much the prediction $c(x_0)$ varies with fluctuations in the training set. Hence it represents the variance of the prediction.

While the noise depends only on the underlying distribution and is therefore independent of the learning algorithm, each learning system has to find a balance in the natural trade-off between bias and variance: If the selection of the classifier c is biased strongly towards a particular setting, a small perturbation in the training set will not change its decision very much, hence the variance part on the error is small. On the other hand, the bias part might be very large, if the underlying problem matches not very well with the system's bias. If the system has a small bias, it must base the choice of c mainly on the training set, so that a small change in (X, Y) might cause a significant change in c and $c(x_0)$. Therefore, the bias part of the error may be small, but the variance tends to be large. It must be emphasized that a large bias error does not automatically cause a low variance and vice versa. It is perfectly possible to design a learning algorithm whose error features high bias and variance ratios. The usual way to decrease the bias part of the error is to run the learning machine with different parameters to fine-tune its bias towards the problem or to use a different learning algorithm with a different bias altogether. The variance part can be reduced by stronger regularization (see Section 3.4.3) or ensemble techniques (see Section 3.5).

The bias-variance decomposition is also possible for binary classification with the zero-one loss (Domingos, 2000b), but in computational learning theory, the phenomenon is more often framed as a compromise between *approximation error* and *estimation error.* This decomposition approach applies, if the learning algorithm is designed to output only classifiers from a restricted hypothesis space $\mathcal{H} \subset \mathcal{C}$. The size of $\mathcal{H}$ can be seen as a measure of the strength of the system's bias: if $\mathcal{H}$ contains only a few classifiers, the system can induce only this restricted type of classifiers. If, on the other hand, $\mathcal{H}$ is very large, then the algorithm can learn a vast number of possible classifiers, but its actual choice depends to a larger degree on the training set. One can therefore distinguish between the error that is made on the selection of classifiers within the restricted hypothesis class and the error that is made on the selection of the class as a whole.

$$\begin{aligned}\varepsilon_c &= \varepsilon^* + \left[\inf_{h\in\mathcal{H}} \varepsilon_h - \varepsilon^*\right] + \left[\varepsilon_c - \inf_{h\in\mathcal{H}} \varepsilon_h\right] \\ &= \text{Bayes error} + \text{approximation error} + \text{estimation error}\end{aligned} \tag{3.5}$$

The first term on the right hand side is the Bayes error, that is, the error of the best classifier overall. No classifier can do better than the Bayes risk. However, the algorithm chooses its output from $\mathcal{H}$ rather than $\mathcal{C}$. The second term represents the *approximation error* that is caused by this restriction. It depends only on the hypothesis class and D and is zero if the Bayes classifier is contained in $\mathcal{H}$. Even if this is the case, the learning algorithm might fail to find and output the best classifier in $\mathcal{H}$, because it has only access to the training set and not the true distribution D. This *estimation error* depends mainly on how representative the training set is on average with regard to D.

Typically, learning systems aim at an approximation and estimation error in the same range, because this indicates a sufficiently broad, but not too general bias. To do so, it is crucial to control the complexity of the classifier to be induced: if there is only a limited amount of training data, there is insufficient information to justify the selection of a classifier from a large hypothesis class and the estimation error is large. This can often be seen in practice, when a learning system is *overfitting.* If there is plenty of training data, a small hypothesis class of simple classifiers might be too restricted to contain a good approximation to the Bayes classifier, hence the approximation error is large. This is known as *underfitting.* Later, we will present methods for under- and overfitting avoidance with rule sets, for instance, in Sections 4.3, 4.4, and 5.3.

3.4 Capacity Control

In the preceding section we have seen that a learning algorithm's bias must match well with the encountered learning problems, in particular with regard to the complexity of the induced classifier. This problem is often called *model selection* or, when the classifier's complexity is concerned, *capacity control.* In the following we will introduce three popular ways to formulate learning algorithms with well-defined and easy-to-analyze biases. In the first setting, the algorithm uses a fixed hypothesis class, so the problem of model selection is essentially left to the user. In the second setting, the learning algorithm chooses first a hypothesis class and then a classifier from within this class to control the classifier complexity automatically. Finally, a system can search for classifiers which optimize the training set accuracy plus some scoring function that penalizes overly complex classifiers. In this way, capacity control is seamlessly integrated in the classifier induction procedure.

3.4.1 Empirical Risk Minimization

In the easiest of the three settings, the learning system is restricted to output classifiers in a specific (often small) hypothesis class $\mathcal{H}$. Thus, the bias is essentially determined by the choice of $\mathcal{H}$ and model selection is left to the user. Since $\mathcal{H}$ is fixed, there is no way to influence the approximation error, so the learning systems in this category can only aim at a low estimation error. A natural way to do so is to search for those hypotheses $h \in \mathcal{H}$ that have the smallest empirical error $\hat{\varepsilon}_h$. This approach is known as *empirical risk minimization.* Obviously, empirical risk minimization is the only sensible strategy in the noise-free setting, because any classifier c with empirical error $\hat{\varepsilon}_c > 0$ is inconsistent with the data and can therefore be discarded. In the noisy setting, the estimation error of empirical risk minimization can be shown to converge to 0, where the rate of convergence depends on the size of $\mathcal{H}$. If $\mathcal{H}$ is finite (such as with DNF rule sets), the size of $\mathcal{H}$ can be easily measured by its cardinality $|\mathcal{H}|$. The following theorem gives a (rather loose) upper bound of the true error depending on the empirical error and the hypothesis space size.

Theorem 3.4.1 (Uniform Deviation (Devroye *et al.*, 1996))**.** *Let D be an arbitrary distribution and (X, Y) be a training set of size n drawn i.i.d. from*

D. Then, for all $\delta > 0$ and for every $h \in \mathcal{H}$:

$$\underset{(X,Y)\sim D^n}{\mathbf{Pr}}\left[\varepsilon_h \le \hat{\varepsilon}_h + \sqrt{\frac{\ln|\mathcal{H}| + \ln\frac{1}{\delta}}{2n}}\right] \ge 1 - \delta \tag{3.6}$$

Thus, with high probability it is the case that the true error of the hypothesis h output by the learning algorithm is smaller than the empirical error plus some penalty that depends essentially on the training set size n and the hypothesis class size $|\mathcal{H}|$. When $n \to \infty$, the penalty term $(\frac{1}{2n}(\ln|\mathcal{H}|+\ln\frac{1}{\delta}))^{0.5}$ converges to zero and $\hat{\varepsilon}_h$ converges to ε_h. Hence, empirical risk minimization is guaranteed to minimize the estimation error in the limit. When $n < \infty$ it is instructive to compare the penalty in (3.6) with the one of the test set bound (3.3). When we fix a hypothesis h before seeing the training set, then the training data essentially constitutes an independent test set and (3.3) tells us that we should expect a penalty of $O(1/\sqrt{n})$ in the worst case. If we choose h after seeing the training set, then the test set bound is too optimistic, but the inequality above states that we should expect a penalty of $O(\sqrt{\ln|\mathcal{H}|}/\sqrt{n})$. The difference is remarkably small, if $|\mathcal{H}|$ is not too large. Unfortunately, it is easy to construct $\mathcal{H}$ of significant size. For instance, it is easy to see that the hypothesis class of DNF rules with at most k terms and l literals per rule can contain 3^{kl} different hypotheses, so that the penalty in (3.6) depends exponentially on k and l.

For weighted rule sets, each classifier $h \in \mathcal{H}$ is given by a set of r rules and a weight vector $w \in \mathbb{R}^r$. Thus, $\mathcal{H}$ is uncountably infinite and measuring the size of $\mathcal{H}$ by its cardinality does not work anymore. The historically first and certainly most popular measure of $\mathcal{H}$'s discriminative power is the *VC dimension* due to Vapnik and Chervonenkis (1971):

Definition 3.4.2. *Let (X,Y) be a training set of size n and $\mathcal{H}$ a hypothesis class. Let $\mathcal{S}_X = \{(h(x_1),\ldots,h(x_n))|h \in \mathcal{H}\}$ be the set of all possible ways in which X can be divided into two classes. Then, the* VC dimension *of $\mathcal{H}$ is the largest n so that there exists a training set X of size n such that*

$$|\mathcal{S}_X| = 2^n$$

The following famous theorem uses this combinatorial property of $\mathcal{H}$ to bound the true error of empirical risk minimization.

Theorem 3.4.3 (Vapnik and Chervonenkis (Devroye *et al.*, 1996))**.** *Let D be an arbitrary distribution, (X,Y) be a training set of size n drawn i.i.d. from D and $\mathcal{H}$ a hypothesis space with VC dimension v. Then, for all $\delta > 0$ and for every $h \in \mathcal{H}$:*

$$\underset{(X,Y)\sim D^n}{\mathbf{Pr}}\left[\varepsilon_h \le \hat{\varepsilon}_h + 2\sqrt{2\frac{v\ln\frac{2en}{v} + \ln\frac{2}{\delta}}{n}}\right] \ge 1 - \delta \tag{3.7}$$

Using some more involved methods, one can remove the unnecessary $\log n$ factor from the right hand side of (3.7), so that the penalty is essentially $O(\sqrt{v}/\sqrt{n})$, only a $\sqrt{v}$ factor worse than the test set bound. Similar to the test set bound, the bound can also be improved, if $\varepsilon^* = 0$. Unfortunately, it is comparably hard to determine the VC dimension of hypothesis classes representing rule sets, that is, Boolean functions. There are some results for k-DNF (Ehrenfeucht *et al.*, 1988) and monotone functions (that is, Boolean functions that can be represented without negated literals) (Procaccia and Rosenschein, 2006), but no general result for k-term DNF or weighted rule sets.

Even though empirical risk minimization is attractive from a theoretical point of view, computing the hypothesis that minimizes the empirical error is often very hard in the worst case. In Section 2.4.1 we list some of the hypothesis classes that are relevant for rule learning. For the most interesting classes, empirical risk minimization is NP-hard and even NP-hard to approximate to within some fixed constant. In practice, these hardness results are often not very prohibitive and most practical systems do well with (sometimes simple) approximation schemes. This is mainly due to the fact that the predictive accuracy of a system depends much more on a suitable bias than on its ability to locate the hypothesis with the best empirical risk exactly. Quite the contrary, the hypothesis that minimizes $\hat{\varepsilon}$ is often rather volatile with regard to small changes in the training set, so that a pure empirical risk minimization approach is often more unstable than a robust approximation scheme and thus exhibits higher variance. We will deal with empirical risk minimization for rule learning in Section 4.2 and 5.2.

A particularly elegant way to sidestep the computational hardness of empirical risk minimization is to not optimize $\hat{\varepsilon}$ directly, but a related quantity. This approach has been made popular by the support vector machine, which aims at linear classifiers with optimal *hinge loss* rather than zero-one loss. Since the hinge loss is a convex function, its minimizer is unique and can be determined efficiently by a quadratic programming procedure. Also, the hinge loss aims at classifiers whose decision boundary has a large margin to the closest training instances. Optimizing for a large margin leads to more robust classifiers, because the classification of a noisy instance remains the same whenever the deviation, which is introduced by the noise, is less than the margin. For many practical learning settings, this bias towards robust classifiers is more effective than an unstable strategy that minimizes the zero-one loss (see Section 5.2 for margin-based empirical risk minimization for rule sets).

3.4.2 Structural Risk Minimization

One of the main problems of straight empirical risk minimization is the missing capacity control. Empirical risk minimization determines the best classifier in a fixed hypothesis class, regardless of the amount of available training data. If there is only a small amount of training data available, there might be not enough information to justify the selection of a particular classifier from the fixed hypothesis class, so the estimation error is large. On the other hand, if there is a wealth of training data, there might be enough information to identify classifiers that are more complex than the ones in the hypothesis class, so the approximation error is large. With empirical risk minimization the choice of the right hypothesis size is essentially left to the user, even though classical model selection methods or bounds like (3.7) could be applied to make this choice automatically.

Structural risk minimization tries to overcome this problem by introducing a sequence of nested hypothesis classes rather than a single class of fixed size. Assume a sequence of nested hypothesis classes $\mathcal{H}_1 \subset \mathcal{H}_2 \subset \ldots$ with finite, but increasing VC-dimension $h_1 < h_2 < \ldots$. Then, structural risk minimization determines the sequence $c_1, c_2, \ldots$ of classifiers that minimize the empirical risks $\hat{\varepsilon}_i$ for each hypothesis class $\mathcal{H}_i$. Finally, from this sequence of classifiers, it outputs the classifier c_i which minimizes the sum of $\hat{\varepsilon}_i + \mathrm{pen}(i,n)$, where the complexity penalty $\mathrm{pen}(i,n)$ penalizes hypothesis classes with large capacity.

$$c_{SRM} := \arg \min_{c \in \mathcal{H}_i, i \in \mathbb{N}} \big(\hat{\varepsilon}_c + \mathrm{pen}(i,n)\big)$$

In its original formulation by Vapnik and Chervonenkis (1974), the complexity penalty was set to

$$\mathrm{pen}(i,n) := \sqrt{\frac{32}{n} h_i \ln(en)} \tag{3.8}$$

so that minimizing $\hat{\varepsilon}_i + \mathrm{pen}(i,n)$ is essentially the same as minimizing the VC bound (3.7) over all hypothesis classes $\mathcal{H}_i$. This strategy avoids overfitting in the sense that it selects a hypothesis class whose capacity is not too large (up to a certain error probability) even for worst case distributions D. Unlike empirical risk minimization with a fixed hypothesis class, structural risk minimization can be made universally consistent, if the sequence $\mathcal{H}_1, \mathcal{H}_2, \ldots$ of hypothesis classes is selected suitably. The following theorem gives the formal conditions, see Chapter 18 of Devroye *et al.* (1996) for more details.

Theorem 3.4.4 (Consistency of SRM). *Let $\mathcal{H}_1, \mathcal{H}_2, \ldots$ be a sequence of hypothesis classes so that for any distribution on (X, Y)*

$$\lim_{i \to \infty} \inf_{c \in \mathcal{H}_i} \varepsilon_c = \varepsilon^*$$

and the VC dimensions $h_1 < h_2 < \ldots$ of the hypothesis classes are finite and satisfy

$$\Delta = \sum_{i=1}^{\infty} e^{-h_i} < \infty$$

Then, the structural risk minimization strategy over $\mathcal{H}_1, \mathcal{H}_2, \ldots$ is universally consistent.

The original penalty (3.8) is rather pessimistic as it assigns a worst-case penalty to each hypothesis class. This ensures consistency in the asymptotic case, but it leads to underfitting in most practical applications. Therefore, practitioners tend to use other penalty functions. If enough training data is available, one can simply set aside a *holdout set* and estimate the predictive accuracy of a classifiers c_i on this independent data. If training data is rare, *cross validation* offers a computationally more expensive alternative. Both methods are intriguing, because they measure the exact performance of the c_is rather than just the capacity of the $\mathcal{H}_i$, but—as apparent from the test set bound (3.2)—they can feature a comparably high variance. Traditional statistical model selection criteria include the *Akaike information criterion* (AIC) and the *Bayesian information criterion*, see chapter 7 of Hastie *et al.* (2001) for a survey.

Computational learning theory has come up with a range of different quantities that can be used to bound $\sup_{c \in \mathcal{H}}(\varepsilon_c - \hat{\varepsilon}_c)$, that is, assess the capacity of a hypothesis class. Among the more recent ones are the *VC entropy* (Vapnik, 1995; Opper, 1999), *covering numbers* (Haussler, 1992), *maximum discrepancy* (Bartlett *et al.*, 2002) and *Rademacher averages* (Bartlett and Mendelson, 2003; Koltchinskii, 2001). Unlike the VC dimension, these quantities depend on the training set, because non-data-dependent capacity estimates have to assume worst case distributions over (X, Y) and are therefore unnecessarily pessimistic. Calculating those quantities can be a non-trivial task, but at least minimum discrepancy and Rademacher averages can be computed by modified empirical risk minimization procedures. Bartlett *et al.* (2002) and Kearns *et al.* (1995) provide experimental and theoretical investigations of certain complexity estimates used in structural risk minimization and model selection in general. We will apply structural risk minimization for DNF rule sets in Section 4.3 and use Rademacher averages, among others, for capacity control in Section 5.3.

3.4.3 Regularization

A third approach to capacity control is *regularization.* Like structural risk minimization, regularization makes use of a penalty function to control the classifier complexity. However, the penalty function is included in the empirical risk minimization procedure directly, so that a regularized learning algorithm minimizes the sum of the empirical error and the penalty for classifiers from a single, large hypothesis class $\mathcal{H}$:

$$c_{reg} := \arg\min_{c \in \mathcal{H}} \left(\hat{\varepsilon}_c + \lambda \mathrm{pen}(c)\right)$$

Here, the *regularization parameter* λ can be fine-tuned by the user to control the extent of the regularization. Regularization can be seen as a smoothed version of structural risk minimization, because it penalizes the classifiers individually rather than through the stepwise nesting of hypothesis classes. If the penalty function is normalized so that $\int_{c\in\mathcal{H}} \exp(-\text{pen}(c))dc = 1$, $\pi(c) := \exp(-\text{pen}(c))$ can be regarded as a probability distribution over the classifiers. Then, regularized risk minimization can be seen as reformulation of maximum a-posteriori estimation in Bayesian inference:

$$\begin{aligned}
\arg\min_{c\in\mathcal{H}} \left(\hat{\varepsilon}_c + \lambda\text{pen}(c)\right) &= \arg\min_{c\in\mathcal{H}} \ln\left(e^{\frac{1}{\lambda}\hat{\varepsilon}_c} e^{\text{pen}(c)}\right) \\
&= \arg\max_{c\in\mathcal{H}} \frac{e^{-\frac{1}{\lambda}\hat{\varepsilon}_c}\pi(c)}{Z_\lambda} \\
&\approx \arg\max_{c\in\mathcal{H}} \frac{\text{likelihood}\cdot\text{prior}}{\text{evidence}}
\end{aligned}$$

where Z_λ is a normalization factor.

Regularization is especially popular with (generalized) linear classifiers (for instance in the SVM, see Cristianini and Shawe-Taylor, 2000), because a linear classifier in a d-dimensional instance space can be conveniently represented by a $(d+1)$-dimensional vector, so that any vector-valued function can be used as a regularization penalty. To allow for an efficient computation of c_{reg} one is interested in penalty functions that give rise to convex optimization problems, such as penalties based on norms $\text{pen}(c) := \|c\|_p$. The two most well-known regularization penalties for linear classifiers are the 1-norm (the *lasso penalty*) and the squared 2-norm (the *ridge penalty*). Both norms penalize vectors whose components feature high variance. This introduces a certain robustness, when some features are highly correlated and a large positive component for one feature can be canceled out by a large negative component for a correlated feature. Additionally, since the 1-norm is “peaked” at points where a component is set to zero, the lasso penalty favors vectors that have many components set to zero. In a sense it acts as a “feature selection” penalty that keeps only the most informative features and ignores the others.

It is remarkable that neither the ridge nor the lasso penalty depend on n or some form of capacity estimate. Thus, unlike the classical structural risk minimization approaches they do not perform capacity control. Instead, the user needs to adjust the parameter λ by hand, which controls the strength of regularization and therewith the effective capacity of the method. Usually, λ is determined by cross-validation or a holdout set, although a more efficient method based on an upper bound on the leave-one-out risk (Joachims, 2000) is available for SVMs. We will describe a regularization scheme based on Lp-norms for weighted rule sets in Section 6.2.

3.5 Ensemble Theory

The theoretical results so far were concerned with the problem of finding a single good classifier from a large space of hypotheses. Apart from considerations about bias and capacity control, this was framed mainly as a search or optimization problem. In the following we we will deal with methods where a classifier is not selected depending on training set and bias, but composed from smaller "sub-classifiers". This is motivated by the idea that a combination of classifiers might balance the various weak spots of each single classifier, so that the combination of classifiers outperforms each single classifier in terms of predictive accuracy. In Bayesian statistics, a similar phenomenon is known: one can show that the best single classifier, the maximum a posteriori classifier, performs worse on average than a prediction based on the posterior probability distribution over all possible classifiers.

The classifiers that originate from such a combination of existing classifiers are generally called *ensembles*, and the methods to generate them are known as *ensemble methods*. The distinction to other approaches is somewhat blurry. Rule sets, for example, can be seen as ensembles of single rules and ensembles based on a weighted voting strategy are essentially linear classifiers. Ensembles can be analyzed from different perspectives. In the following we present the theoretical foundations that were instrumental for bagging and boosting, two particularly popular types of ensembles.

3.5.1 Bagging

Bagging is short for "Bootstrap aggregating" and was introduced by Breiman (1996) as a technique for decreasing the estimation part of the prediction error for general unstable learning systems. The application of bagging to decision tree classifiers led (with some further refinements) to the development of the *Random Forest* (Breiman, 2001) learning system. In both cases, the ensemble method proceeds as follows: First, it draws a sequence $(X_1, Y_1), \ldots, (X_k, Y_k)$ of k different *bootstrap samples* from the original training set. A bootstrap sample of an instance set (X, Y) containing n instances is simply the result of randomly picking n instances with replacement from (X, Y). Then, it induces a base classifier c_i for each bootstrap sample (X_i, Y_i). The resulting ensemble of classifiers $(c_1, c_2, \ldots, c_k)$ is output as the final classifier. To predict the class of a new test instance x, the ensemble applies each of the base classifiers c_i to x and then predicts the class label which was output by the majority of base classifiers.

From a theoretical point of view, bagging essentially aims at reducing the estimation part of the prediction error by averaging over a large number of classifiers. This is especially effective for *unstable predictors*, that is, learning systems where a small change in the training data can lead to large changes in the predictions on a test set. This can be framed as follows: Assume we have a random number generator which draws k i.i.d. random vectors $\Theta_1, \ldots, \Theta_k$ according to the fixed distribution T. The learning system generates k different base classifiers $c_{\Theta_1}, \ldots, c_{\Theta_k}$, where each c_{Θ_i} depends on the training set (X, Y) and the random vector Θ_i. In the case of bagging, the Θ_is represent the random information used to build the bootstrap sample, but the following results also apply to any other randomized learning algorithm. Let $\varepsilon_\Theta(x, y)$ denote the error on test instance (x, y) of the classifier c_Θ. Then, the expected error of a base classifier averaged over all Θs on an instance (x, y) is $\varepsilon^b(x, y) := \mathbf{E}_\Theta\, \varepsilon_\Theta(x, y)$, and ε^e denotes the true error of the voting procedure of the ensemble $(c_1, \ldots, c_k)$. The following theorem (Breiman, 2001) gives an upper-bound for the expected error of such an ensemble (asymptotically for $k \to \infty$).

Theorem 3.5.1 (Ensemble Bound)**.** *Denote the* strength *of a base classifier by*

$$s = \mathop{\mathbf{E}}_{(x,y)\sim D} \left[1 - 2\varepsilon^b(x, y)\right]$$

and the ensemble correlation *by*

$$\bar{\rho} = \mathop{\mathbf{E}}_{\Theta\sim T} \mathop{\mathbf{E}}_{\Theta'\sim T} \rho\left[c_\Theta(x), c_{\Theta'}(x)\right]$$

where $\rho[f_1(x), f_2(x)]$ is the correlation of $f_1(x)$ and $f_2(x)$ with regard to $x \sim D$. Then, for $k \to \infty$, ε^e converges almost surely to $\varepsilon^e_\infty := \mathbf{E}_{(x,y)\sim D}\, \varepsilon^b(x, y)$. Additionally, if $s \geq 0$,

$$\varepsilon^e_\infty \leq \frac{\bar{\rho}(1 - s^2)}{s^2} \tag{3.9}$$

In other words, with an increasing number of base classifiers, the ensemble's true error can be asymptotically upper-bounded by a term depending on the strength of the base classifiers and the ensemble correlation. The theorem has two implications. First of all, there is no overfitting in the sense that the predictive performance deteriorates when adding many more base classifiers. Instead, the ensemble's predictive accuracy converges to a limiting value. This is in contrast to methods such as separate-and-conquer rule learning, where extending a classifier increases its complexity, which in turn leads to a larger estimation error.

Second, there is a certain trade-off between the parameters; in order to minimize the right side of (3.9) one would like to maximize the strength s, so that $(1 - s^2)/s^2$ is small. Unfortunately, strong classifiers assign the correct class to most instances. Hence, they agree on large parts of the training set and are highly correlated to each other, so that $\bar{\rho}$ is large. However, this is exactly the opposite of what we would like to have, because a large $\bar{\rho}$ in turn increases the right hand side of (3.9). Apparently it is necessary to find a compromise between the classifier strength and ensemble correlation. In practice, this ambivalence is not a problem, because the strength measures the *true accuracy*, which depends on D and is unknown. Hence, the best thing one can do is to build a number of classifiers with high empirical accuracy and combine them in an ensemble. If the randomized base learning algorithm is stable, it will output only a few highly correlated base classifiers and bagging will neither help nor harm very much. If, on the other hand, the learning algorithm is unstable and outputs base classifiers that differ significantly from each other, $\bar{\rho}$ will be small and the bound suggests that bagging might improve the predictive accuracy. Since learning algorithms with large hypothesis classes and broad unspecific biases are naturally unstable, bagging can also be seen as a regularization method, which is applied *after* the actual learning step.

While theoretical and empirical experiments confirm that bagging generally reduces the variance (or estimation error), its impact on the bias (or approximation error) is less clear. Most research focuses on regression with the squared loss, where one can show that bagging reduces the non-linear parts of the variance in a bias-variance decomposition (Friedman and Hall, 2000) or—for non-smooth predictors like decision trees—it reduces the variance by smoothing the discontinuities at "hard decisions" (Bühlmann and Yu, 2000). There is little research on the effect of bagging classifiers with the zero-one loss. Rao and Tibshirani (1996) frame bagging as a Bayesian averaging approach with a non-informative Dirichlet prior, while a recent empirical study indicates that bagging equalizes the influence of individual training instances and therefore makes more effective use of the available information (Grandvalet, 2004). We will describe an approach to capacity control for DNF rule sets similar to bagging in Section 4.4.

3.5.2 Boosting

Another popular ensemble method is *boosting*. Boosting has its roots in the theory on probably approximately correct learning (*PAC learning*). The earliest work on boosting was motivated by the question of whether the class of *weak learners*, that is, learning algorithms whose classifiers have predictive accuracy slightly larger than chance, differs from the class of *strong learners*, which can approximate the Bayes classifier when given reasonable amounts of time and data. To show that the two classes are essentially equivalent, Schapire (1990) devised an algorithm that is able to efficiently combine weak classifiers into a strong ensemble classifier. Further refinements led to the popular *AdaBoost* algorithm (Freund and Schapire, 1997), which was shown to work well in numerous empirical studies.

Boosting differs from bagging in two important regards: First of all, it assigns weights to the training instances and asks the base learner to focus on instances with large weights. Initially, the weights are set equally for

Algorithm 2 The AdaBoost algorithm. It is given a training set (X, Y) and the maximal ensemble size t.

```
procedure AdaBoost($X$, $Y$, $t$)
    $D_0 \leftarrow$ uniform distribution over the instances in $X$
    for $i = 1, 2, \ldots, t$ do
        $h_i \leftarrow$ base classifier induced on $(X, Y)$ with instance weights $D_{i-1}$
        $\epsilon_i \leftarrow \mathbf{Pr}_{j \sim D_{i-1}} \left[ h_i(x_j) \neq y_j \right]$
        $\alpha_i \leftarrow \frac{1}{2} \ln \left( \frac{1-\epsilon_i}{\epsilon_i} \right)$
        $Z_i = \sum_{j=1}^{n} D_{i-1}(j) e^{-\alpha_i y_j h_i(x_j)}$
        for $j = 1, 2, \ldots, n$ do
            $D_i(j) = \frac{1}{Z_i} D_{i-1}(j) e^{-\alpha_i y_j h_i(x_j)}$
        end for
    end for
    return $c : x \mapsto \text{sign}\Big( \sum_{i=1}^{t} \alpha_i h_i(x) \Big)$
end procedure
```

all instances, but during the construction of the ensemble, boosting assigns large weights to instances that have been misclassified by a large number of previous base classifiers. Thus, it actively aims at finding a diverse set of base classifiers complementing each other. This can be seen as a deterministic strategy to minimize the ensemble correlation $\bar{\rho}$ in (3.9). Second, AdaBoost also assigns weights to the individual classifiers in the ensemble, usually depending on their (weighted) training accuracy. In the final voting procedure, each classifier contributes according to its weight, so that accurate classifiers are valued more than inaccurate ones. The pseudo code for AdaBoost is given in Algorithm 2.

Since AdaBoost generates the base classifiers deterministically and since it specifically focuses on the shortcomings of the existing ensemble, the convergence guarantee of theorem 3.5.1 does not apply.[1] It is therefore perfectly possible to achieve overfitting by increasing the ensemble size t. Indeed, the following theorem (Schapire, 2003) upper-bounds the training error of AdaBoost depending on the accuracy of the base classifiers and the number of boosting rounds t.

Theorem 3.5.2 (AdaBoost Training Accuracy Bound)**.** *Let Z_i, ϵ_i and α_i be the values that AdaBoost computes in its main loop in Algorithm 2. Then, the training error $\hat{\varepsilon} = \frac{1}{n} \sum_{i=1}^{n} l(c(x_i), y_i)$ of the boosted ensemble classifier*

[1]Indeed, recent research indicates that AdaBoost can get trapped in periodic cycles, so it sometimes fluctuates rather than converges (Rudin *et al.*, 2004).

$c(x) = sign(\sum_{i=1}^{t} \alpha_i h_i(x))$ *is upper-bounded as follows:*

$$\hat{\varepsilon} \leq \frac{1}{m} \sum_{j=1}^{n} e^{-y_j \sum_{i=1}^{t} \alpha_i h_i(x_j)} = \prod_{i=1}^{t} Z_i \tag{3.10}$$

$$\leq e^{-2\sum_{i=1}^{t}(0.5-\epsilon_i)^2} \tag{3.11}$$

This results has two interesting implications. First, it is easy to see that AdaBoost aims at minimizing the Z_i by adding base classifiers that focus on the misclassified instances, that is, those instances, which contribute most to the Z_i. Thus, inequality (3.10) means that boosting is essentially a greedy method to minimize the *exponential loss* $l_{\exp}(x, y) := \exp(-yc(x))$ rather than the zero-one loss on the instances in the training set. In this regard it is very similar to *logistic regression*, because minimizing the empirical exponential loss is essentially equivalent to optimizing the log-likelihood of an additive logistic model (Friedman *et al.*, 2000).

Second, if all base classifiers are better than random by a fixed minimum margin ϵ_{min}, so that $\epsilon_j \leq 0.5 - \epsilon_{min}$ for all $j \leq t$, then the right hand side of (3.11) declines exponentially fast with the number of boosting rounds t. Thus, under this assumption, boosting ensembles converge to a consistent classifier ($\hat{\varepsilon} = 0$) for large ensemble sizes, a clear sign of overfitting, if the underlying distribution is noisy. Freund and Schapire (1997) showed that the true error of a boosted ensemble can be upper-bounded by a term that depends on the VC-dimension of the base classifier and t. If t is large, the bound is loose, another theoretical indication that boosting overfits. However, early practical experiments indicated that AdaBoost is surprisingly robust against overfitting, even for very large rounds of boosting. In some cases, the test error continues to decrease, even though the training error has already reached zero. This phenomenon has been subject of many discussions. While there are numerous investigations that shed light on the overfitting behavior of boosting, there is currently no single conclusive theory.

Instead, boosting has spawned a wealth of research, some of which led to new boosting strategies and connections to other learning systems. To paraphrase Bartlett *et al.* (2004), boosting can be seen as a strategy to optimize the margin, that is, the distance between the (linear) decision boundary and the training instances (Schapire *et al.*, 1998), as a game-theoretic strategy to play a zero-sum game between the boosting algorithm and the weak learner (Freund and Schapire, 1996), as a general convex optimization method (Rätsch *et al.*, 2000), as a gradient descent method for convex loss optimization (Mason *et al.*, 1999), and as a special case of optimizing Bregman-distances (Collins *et al.*, 2000). Some of these frameworks also apply to the ensemble based approach outlined in Section 4.4.

3.6 Bayesian Learning

In the preceding sections, we framed classification as a statistical estimation problem. While doing so, we made certain assumptions and abstractions about the underlying phenomenon (fixed, but unknown distribution), the available information (an i.i.d. sample), and the object we would like to learn (a classifier from a predefined hypothesis class that minimizes the test or true error). While these modelling decisions were overall rather conservative, one can certainly argue about their utility and appropriateness. In the following we will sketch classification from a different, Bayesian point of view. We will not treat the subject exhaustively, but rather give a short review of the typical Bayesian framework and then touch upon the issues that are relevant for the succeeding sections.

So far, we framed the data-generating phenomenon as a fixed, but unknown distribution. This is a rather pessimistic assumption, because the results in this framework must hold for all, even the most improbable and difficult distributions. For instance, in the discussion following theorem 3.3.9, we mentioned a few awkward data-generating distributions that are particularly hard to learn, but extremely unlikely to appear in practice. If one could quantify which distributions are likely to appear in practical situations, one could modify the learning algorithm to prefer classifiers that represent likely phenomena over unlikely ones. An easy way to model such "likeliness" information is by assigning a *prior probability* to each hypothesis in the hypothesis class before the actual learning takes place. Given such a prior distribution $R := \mathbf{Pr}[c]$ on the classifiers $c \in \mathcal{H}$ and a training set, the learning algorithm would then only need to find the hypothesis that is most likely according to R while explaining the training set well. More formally, the learning task can be framed as follows for the noise-free setting[2]. We assume a set of n fixed unlabeled training instances $(x_1, \ldots, x_n)$. First, the true underlying classifier (the underlying "concept") c_t is drawn according to the prior R, and the instances are labeled according to c_t so that $y_i = c_t(x_i)$. Then, the labeled instances $((x_1, y_1), \ldots, (x_n, y_n))$ and the prior R are given to the learning algorithm. It turns out that the knowledge of R allows for the design of a conceptually very simple, yet optimal learning algorithm. Let the random variable $C \sim R$ represent the randomly chosen concept and the random variable Y denote the resulting sequence of class labels assigned to the $(x_1, \ldots, x_n)$. Then, one can apply Bayes' law to compute the *posterior probability* $\mathbf{Pr}[C = c_t | Y = (y_1, \ldots, y_n)]$ that a particular concept c_T was chosen given the training set $(y_1, \ldots, y_n)$:

$$\mathbf{Pr}[C|Y] = \frac{\mathbf{Pr}[Y|C] \cdot \mathbf{Pr}[C]}{\mathbf{Pr}[Y]} \tag{3.12}$$

[2]The noisy setting can be formalized in the same way by setting $\mathcal{H}$ to the space of distributions over $\mathcal{X} \times \mathcal{Y}$, or, equivalently, to the space of non-deterministic classifiers.

This means the learning algorithm can determine the most probable classifier given the training set by computing the *maximum a posteriori classifier* $c_{\text{MAP}} := \arg\max_{c_t \in \mathcal{H}} \mathbf{Pr}[C = c_t | Y = (y_1, \ldots, y_n)]$. Even better, $\mathbf{Pr}[C|Y]$ induces a *posterior distribution* $S := \mathbf{Pr}[C|Y = (y_1, \ldots, y_n)]$ on the set of classifiers. Thus, when observing a new unlabeled test instance x', one can compute the probabilities that the corresponding class label Y' takes a specific value y' by summing up over all classifiers:

$$\mathbf{Pr}[Y' = y'|Y] = \sum_{c \in \mathcal{H}, c(x') = y'} S(c) \tag{3.13}$$

Thus, a Bayesian learning system simply calculates the posterior S from prior and training set and then outputs the class label y' which maximizes $\mathbf{Pr}[Y' = y'|Y]$. It is easy to verify that such a learning system is optimal, because it achieves on average the best possible accuracy when applied repeatedly on learning problems drawn according to the prior R.

The described Bayesian learning system is known as the *Bayesian averaging* algorithm or the *Bayes optimal classifier*. Unfortunately, there are two important problems with the application of such a Bayesian system in practice. First of all, its optimality depends crucially on the fact that the learning problems are drawn according to the prior R. Of course it is impossible to know which learning problems one will encounter in the future, so there is no way to compute or estimate a correct "universal prior" for generally applicable learning systems. Also, since one will rarely encounter one and the same learning problem twice, the usual interpretation of probabilities as relative frequencies makes little sense. To alleviate this problem, the prior and posterior probabilities in Bayesian analyses are usually interpreted to quantify the "degree of belief" one has in the validity of a classifier, that is, the uncertainty of the available background information, rather than relative frequencies. Thus, the Bayesian averaging algorithm is optimal in the sense that it makes the best possible use of the information in the prior, but not in the sense that it achieves the best possible prediction accuracy. Finding a practically good prior can be a hard problem. The second practical problem of Bayesian methods is the computational complexity of representing and processing the prior and posterior distributions. It is clearly not efficient to store the probability values for each classifier in $\mathcal{H}$ individually. Thus, Bayesian learning systems model prior and posterior often with fixed classes of parameterized distributions (for instance, Gaussians) or apply sophisticated inference and sampling techniques (for example, Markov Chain Monte Carlo). Constructing a meaningful prior that incorporates the available background knowledge efficiently while keeping the computational complexity at a reasonable level can be challenging.

Regardless of these practical issues, Bayesian learning has the distinctive advantage of modeling the bias of the learning system explicitly through the prior. This allows for a particularly elegant way to investigate, compare and analyze different biases (that is, priors) for different learning systems. Since every (Non-Bayesian) learning system can be represented by a function $A : (X, Y) \to \mathcal{H}$, one can always find a prior R and a likelihood distribution $\mathbf{Pr}[Y|C]$ which yield an equivalent Bayesian learning system. Thus, one can analyze non-Bayesian systems in a Bayesian framework and this often leads to interesting insights. For instance, we outlined in Section 3.4.3 how regularized risk minimization (and structural risk minimization as a special case) can be translated into a Bayesian form quite naturally.

As an easy example of a Bayesian analysis of a non-Bayesian system, consider weighted rule sets or—more generally—linear classifiers, as described in Section 2.2.3. A weighted rule set predicts the class $\mathrm{sign}(\sum_i w_i r_i(x) + b)$, where $r_i : \mathcal{X} \to \{-1, +1\}$ denotes the ith rule, w_i is the corresponding weight and b is the weight of the default rule. Thus, a weighted rule set predicts the positive class exactly when $\sum_i w_i r_i(x) + b \geq 0$, or, equivalently, when

$$e^{\sum_i w_i r_i(x)+b} = e^b \prod_i e^{w_i r_i(x)} \geq 1$$

Scaling the weight vector with an arbitrary factor does not change the classification of a linear classifier. It turns out that one can always find a scaling factor so that each w_i and b can be written as the logarithm of a quotient of two probabilities: $\exp(w_i r_i(x)) = \mathbf{Pr}[r_i(x)|y = +1]/\,\mathbf{Pr}[r_i(x)|y = -1]$ and $\exp(b) = \mathbf{Pr}[y = +1]/\,\mathbf{Pr}[y = -1]$ (see Jäger, 2003, for a proof). With this, the classifier outputs the positive class whenever:

$$\mathbf{Pr}[y = 1] \prod_i \mathbf{Pr}[r_i(x)|y = 1] \geq \mathbf{Pr}[y = -1] \prod_i \mathbf{Pr}[r_i(x)|y = -1]$$

This is the well known *Naive Bayes* decision rule. Naive Bayes is the maximum a posteriori classifier in the case when the rules are conditionally independent given the class so that $\mathbf{Pr}[r_1(x), \ldots, r_m(x)|y] = \prod_i \mathbf{Pr}[r_i(x)|y]$. Thus, using a linear classifier to combine the rules is essentially equivalent to the Naive Bayes assumption that the rules are conditionally independent given the class. Furthermore, if the rules are indeed conditionally independent and the weights are proportional to the log odds computed from the relative frequencies in the training set, then the resulting weighted rule set constitutes—from a Bayesian perspective—the optimal classifier with regard to the information extracted by the rules.

Another way to analyze weighted rule sets from a Bayesian perspective is to frame ensembles of weighted classifiers as an approximation to the Bayesian averaging algorithm. Recall from (3.13) that the Bayesian averaging algorithm predicts the positive class for test instance x', if

$$\sum_{c \in \mathcal{H}, c(x')=+1} \mathbf{Pr}[C = c|Y] \geq \sum_{c \in \mathcal{H}, c(x')=-1} \mathbf{Pr}[C = c|Y]$$

which is equivalent to

$$\sum_{c \in \mathcal{H}} c(x')\,\mathbf{Pr}[C = c|Y] \geq 0 \tag{3.14}$$

Since the hypothesis space $\mathcal{H}$ is usually very large, it is not efficient to compute all the summands in (3.14). Fortunately, most classifiers $c \in \mathcal{H}$ are clearly not consistent with the training set, so that $\mathbf{Pr}(C = c|Y)$ is negligibly small. It is therefore often enough to take the sum only over the "likely" classifiers, whose $\mathbf{Pr}[C = c|Y]$ is comparably large. This is a straightforward recipe for constructing an ensemble: First, use a (possibly randomized) base learner to identify the set $\{c_1, \ldots, c_m\}$ of classifiers which agree well with the training set. Then estimate a weight $w_i \propto \mathbf{Pr}[C = c|Y]$ for each classifier depending on its performance on the training set and some prior information. Finally predict according to $\text{sign}(\sum_i w_i c_i(x))$. We will follow this recipe in Section 4.4.2 to perform capacity control with DNF rule sets. The performance of such a system is optimal, if $\mathcal{H}$ is large enough and the prior and likelihood estimates are justified. In practice, it can be quite hard to find good estimates, see Domingos (2000a) for an example application. On the other hand, given a successful ensemble learning system, one can deduce priors and likelihood estimates that yield an equivalent Bayesian learning system. This allows for a better comparison of general ensemble methods. For instance, Rao and Tibshirani (1996) frame bagging as a Bayesian averaging procedure with a non-informative Dirichlet prior. The fact that Bayesian analysis associates a probability with the *margin* $\mu := |\sum_i w_i c_i(x)|$ can be used for *abstaining classifiers* (Friedel, 2005; Friedel *et al.*, 2006; Pietraszek, 2005). The main idea is to construct a classifier, which predicts a class label y' for test instance x' only if $\mathbf{Pr}[Y' = y'|Y]$ (or, equivalently, the ensemble margin μ) is larger than a threshold θ, and abstains from prediction otherwise. In this way, the classifier makes a prediction only on instances, which are likely to be classified correctly, thus improving the predictive accuracy. We investigate abstaining classifiers in a non-Bayesian framework in Section 4.4.3.

Bayesian statements about the predictive accuracy of a classifier always depend on the validity of the prior and should therefore be taken with a grain of salt. However, it is possible to investigate the Bayesian averaging procedure in a non-Bayesian, that is, frequentist, setting. When doing so, one can upper bound the true error of the Bayesian averaging classifier, even if the training set was not drawn according to the prior. To state this more formally, let us go back to the frequentist setting described in Section 3.2 and assume that the training set (X, Y) is drawn i.i.d. from a fixed, but unknown distribution D. Now, we consider a fixed hypothesis space $\mathcal{H}$ and ask the user to state a fixed prior probability distribution R on $\mathcal{H}$ before she has seen the training set. Then, we give R and (X, Y) to a learning system. The learning system outputs some distribution S over $\mathcal{H}$. It can do so by applying Bayes' law, but any other derivation of a "posterior" S works as well. With this, we can make use of S in two different ways to predict the class label of a test instance x'. The *Gibbs classifier* is a non-deterministic predictor. It draws a hypothesis c from $\mathcal{H}$ according to S and then outputs the class label $c(x')$. Thus, its true error is $\varepsilon_G := \mathbf{E}_{(x',y')\sim D}\, \mathbf{E}_{c\sim S}[l(c(x'), y')]$ and its expected empirical error is $\hat{\varepsilon}_G := \frac{1}{n}\sum_{i=1}^{n} \mathbf{E}_{c\sim S}[l(c(x_i), y_i)]$. The *voting classifier* is deterministic and follows the original Bayesian averaging strategy: $c_V(S, x') := \arg\max_{y'\in\{-1,+1\}} \mathbf{Pr}_{c\sim S}[c(x') = y']$. Hence, it outputs the positive class, whenever S assigns a larger probability mass to the hypotheses that vote for the positive class than to those that output the negative class (and vice versa). The true error of the voting classifier is thus $\varepsilon_V := \mathbf{E}_{(x',y')\sim D}\, l(c_V(x'), y')$.

The following *PAC-Bayesian* theorem can be used to bound the true error of Gibbs and voting classifiers. We denote the *Kullback-Leibler divergence* between two distributions R and S over a countable space $\mathcal{H}$ by $D(R\|S) := \sum_{c\in\mathcal{H}} R(c)\log(R(c)/S(c))$.

Theorem 3.6.1 (PAC Bayesian (McAllester, 1999; Herbrich, 2001))**.** *Let R be an arbitrary fixed distribution over $\mathcal{H}$, and S be a distribution over $\mathcal{H}$ depending on a training set (X, Y) of size n drawn i.i.d. from a fixed distribution D. Then, for any $\delta > 0$:*

$$\mathop{\mathbf{Pr}}_{(X,Y)\sim D^n}\left[\varepsilon_G \le \hat{\varepsilon}_G + \sqrt{\frac{D(S\|R) + \ln\frac{1}{\delta} + \ln n + 2}{2n-1}}\right] \ge 1-\delta$$

$$\mathop{\mathbf{Pr}}_{(X,Y)\sim D^n}\left[\varepsilon_V \le 2\hat{\varepsilon}_G + 2\sqrt{\frac{D(S\|R) + \ln\frac{1}{\delta} + \ln n + 2}{2n-1}}\right] \ge 1-\delta$$

That means that the bound for the error of the voting classifier is twice the bound for the Gibbs classifier. Comparing this bound with the test set bound (3.3), we see that there is essentially only an additional $O(\sqrt{\ln n})$ factor, if the Kullback-Leibler divergence $D(R\|S)$ between the "prior" and the "posterior" is sufficiently small. In a sense, the PAC-Bayesian bound is similar to the Bayesian analysis of an ensemble classifier in that it makes use of the quality of any "prior" background knowledge one may have about the learning problem. It differs, though, in that it bounds the true error *for any arbitrary underlying distribution D*. A slightly modified version of the PAC-Bayesian bound can be shown to be tight for some distributions (Langford, 2005). We will use PAC-Bayesian methods later, for example, in Section 4.4.3 and 5.3.2.

Chapter 4

DNF Rule Sets

In this chapter we treat learning DNF formulae from a statistical perspective. After recalling some of the trade-offs involved in learning DNF, we address the empirical risk minimization problem, that is, the problem of finding rule sets that agree with the training data. We present a probabilistic algorithm for doing so in the noise-free setting, and a stochastic local search algorithm for the noisy case. For capacity control, we follow two different approaches. For the SL^2 learner, we perform structural risk minimization with a "pure" bias towards small, comprehensible rule sets. An empirical comparison with established rule-learners shows, that SL^2 induces smaller and simpler rule sets while maintaining the same level of predictive accuracy, albeit at the cost of high time complexity. The second approach combines SLS with capacity control based on ensembles. We give a theoretical analysis of the system's prediction performance, including the case where the classifier can abstain from uncertain predictions.

4.1 Trade-Offs in Rule Learning

Before we will delve into technical details in the following sections, let us recall the basic rule learning setting and the goals one wishes to achieve. As a basic premise, we assume that the rule learner is given a set of labeled examples and has to induce a set of rules that correctly describes the underlying data-generating phenomenon. As outlined in Section 2.5, numerous rule learning algorithms and fielded systems are available for addressing this task. Many of them employ a separate-and-conquer approach to generate rules while applying some pruning heuristics to avoid overfitting and handle noise. To evaluate and compare different rule learning algorithms, many criteria, most notably predictive accuracy, simplicity and time complexity, have been employed. The most prominent one, predictive accuracy, aims at generating rule sets with a low misclassification error on unseen test data. *Simplicity* aims at finding rule sets that are as simple as possible. This can be measured through the number of rules and literals in the rule set.

Simplicity has served in the machine learning literature as the most prominent measure of human comprehensibility (compare, for example, Michalski, 1983) as it is generally agreed that the smaller the rule set, the easier it is to understand. Finally, one would like to have low time complexity, so that algorithms scale well on large data sets.

The above mentioned criteria often contradict each other. Indeed, the empty rule set is the smallest one that can be generated and requires the least amount of time to generate. Unfortunately, it typically has an extremely low predictive accuracy. On the other hand, finding small, but consistent rule sets is an NP-hard problem (Pitt and Valiant, 1988). As outlined in Section 2.5.1, the popular separate-and-conquer approach can be seen as an approximation scheme for DNF minimization. Thus, it forms a compromise between the goals of simplicity and low time complexity. In the following, we deliberately ignore time complexity considerations and focus explicitly on simplicity and predictive accuracy. This is for three reasons. First of all, it allows one to get valuable insight into the "true" trade-off between predictive accuracy and simplicity in rule learning, that is, without the influence of heuristics that are usually applied to improve time complexity. Second, as computers get faster, time complexity is less of an issue. Third, if desirable, one could incorporate techniques to speed up processing times later on. By comparing such a modified system to the original algorithm, one could then quantify the loss of simplicity or predictive accuracy that is caused by the incorporation of performance tuning techniques.

So, how can one induce small, but predictive rule sets? As described in Section 3.3.3, there is no easy way to guarantee a certain level of predictive accuracy in one *particular* setting, because the individual setting might not match with a learner's bias. Instead, one can only apply the statistical methodology presented in the preceding chapter to ensure good predictive performance under *broad* conditions. In our case it is sensible to apply the structural risk minimization principle as explained in Section 3.4.2: we start with a sequence of nested hypothesis classes of increasing size. Then, for each hypothesis class, we identify the hypothesis that minimizes the empirical error on the training set and among those hypotheses we choose the one which forms the best trade-off between under- and overfitting. With this approach we have to tackle two challenges: the first one is to find the hypothesis with the smallest empirical error in a hypothesis class of restricted size. We propose two algorithms to solve such empirical risk minimization problems in Section 4.2. The second problem is to assess the right hypothesis class size to avoid under- or overfitting. We describe a system for doing so in Section 4.3. Finally, we can put the considerations into practical use and implement the SL^2 (Stochastic Local Search Learner) system that performs structural risk minimization with a "pure" bias towards small rule sets. While this system is too slow to be used in most practical settings, it is a perfect benchmark tool to experimentally compare to state-of-the-art rule learners. In this way, one can assess to which degree the existing rule learners succeed in finding predictive and small rule sets. We present such an empirical study in Section 4.3.2.

A different approach to capacity control for DNF rule sets is through ensemble methods, as described in Section 3.5. Ensembles are particularly well suited for rule learning, because typical rule learners are unstable classifiers. In Section 4.4.2 we present a rule learning system that uses ensembles for capacity control in a similar way as Random Forests (Breiman, 2001) do for decision trees.

4.2 Empirical Risk Minimization for DNF Rule Sets

This section deals with the algorithmic aspects of finding a DNF rule set that agrees well with a given training set. Before we delve into the details, let us recall the basic setting and assumptions. First of all, as explained in Section 2.2.1, we want to induce rule sets that are logic formulae in disjunctive normal form. In other words, each rule set is a logical disjunction of *terms* $t_1 \vee t_2 \vee \ldots \vee t_k$, where each term t_i is a conjunction of literals: $t_i = (l_{i1} \wedge l_{i2} \wedge \ldots \wedge l_{ik_i})$. $\mathcal{F}$ denotes the space of all formulae. For the scope of this chapter the terms *formula* and *rule set* and the terms *rule* and *term* are used with the same meaning. For now, we are not concerned with the formation of the literals and the conditions they represent. Instead we simply assume that all information about the training and test instances is encoded through a fixed set of m distinct Boolean variables $\{v_1, \ldots, v_m\}$. With this, a literal l in a rule set is either an unnegated or a negated variable, that is, it is of the form v_i or $\neg v_i$. Thus, we can represent a training instance as a vector of truth values $x \in \{\text{true}, \text{false}\}^m$, or, in a more compact representation, as a bit vector $x \in \{0,1\}^m$, where 0 represents false and 1 indicates true.[1] Given an instance x, we write $x(i)$ to denote the ith component of x. We say a term t *covers* an instance x (alternatively: *x satisfies t*), if all unnegated variables in t are set to 1 by x and all negated variables in t are set to 0. Likewise, formula f covers an instance, if at least one of its terms covers x. A formula f or a term t is said to *contradict* an instance x (alternatively: *x violates f*), if it does not cover it. We will write $f(x)$ to denote the application of a formula f on an instance x, more precisely:

$$f(x) := \begin{cases} 1 & \text{if} f \text{ covers } x \\ 0 & \text{if} f \text{ contradicts } x \end{cases}$$

Given a training set of labeled instances $\{(x_1, y_1), \ldots, (x_n, y_n)\} \subset \{0,1\}^{m+1}$, one can count the fraction of instances whose class label disagrees with a

[1]The use of 0 and 1 to represent false and true is chosen for notational compactness only. In later sections, it will be more convenient to represent false by -1 rather than 0.

rule set's f prediction $f(x)$. This is the well-known empirical error for the zero-one loss $\hat{\varepsilon}_f := \frac{1}{n}\sum_{i=1}^{n}|f(x_i) - y_i|$, as explained in Section 3.2.

With this, the problem of empirical risk minimization can be stated as follows: Given a restricted set of formulae $\mathcal{H} \subset \mathcal{F}$ (the hypothesis space) and a set (X, Y) of labeled training instances, find a formula $f \in \mathcal{H}$ that minimizes $\hat{\varepsilon}$. This is in general a combinatorial optimization problem. The hardness of this problem depends essentially on the hypothesis space $\mathcal{H}$. It is trivial to solve if $\mathcal{H}$ is the set of all formulae $\mathcal{F}$, because one can just build a rule set where each rule covers exactly one positive example. Typically, rule learning aims at formulae with a small number of rules and not too many literals per rule. As explained above, this is motivated by considerations on how humans tend to describe and communicate concepts (Michalski, 1983). Limiting only the number of literals per term (that is, learning k-DNF) has been investigated in computational learning theory (Valiant, 1984), but it is of little practical relevance. This is because a k-DNF formulae can contain up to $(2m)^k$ rules, way too many to comprehend by humans for realistic settings of k. In order to gain easily interpretable rule sets, it is necessary to limit the number of terms. The class of rule sets containing at most k rules is known as the class of *k-term DNF formulae* in learning theory.

We will deal with two versions of the problem of learning k-term DNF rule sets. The first version is concerned only with the question of whether there is a k-term DNF rule set f that is consistent with a given training set (X, Y), that is, where $f(x_i) = y_i$ for all $1 \leq i \leq n$. This can be seen as an application of empirical risk minimization in the noise-free setting, where it makes only sense to induce rule sets that agree exactly with the training set. This problem is usually called the *k-term DNF consistency* or *learning* problem (Kearns and Vazirani, 1994; Rückert, 2002). It can be trivially solved for $k = 1$, but it is NP-hard for $k \geq 2$ (Pitt and Valiant, 1988). The second and practically more relevant version is the actual empirical risk minimization problem in the noisy case: given a training set (X, Y) and a constant k, find the k-term DNF rule set f that agrees with the largest possible number of training instances, that is, find $\arg\min_{f\in\mathcal{H}} \hat{\varepsilon}_f$. This problem is known as the *maximum agreement problem for k-term DNF*. It can be shown to be NP-hard for $k \geq 1$ (Kearns and Li, 1993) and even for single monotone terms, that is, terms containing only non-negated literals (Angluin and Laird, 1988). Recent results indicate that even the problem of finding approximate solutions, which are within some constant factor of the optimal solution's accuracy, is NP-hard (Ben-David *et al.*, 2003). In the following two sections we describe algorithmic approaches to solving the k-term DNF learning and maximum agreement problems.

4.2.1 A Randomized Algorithm for k-term DNF Consistency

The k-term consistency problem can be stated as follows: Given an integer $k > 1$ and a training set $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ of n labeled instances, where each instance (x_i, y_i) is composed of an m-digit bit vector $x_i \in \{0,1\}^m$ and a binary class label $y_i \in \{0,1\}$, find a DNF formula f with at most k terms so that for all $1 \leq i \leq n : f(x_i) = y_i$. A naive algorithm for solving such a problem could simply enumerate all possible DNF formula with k terms over m literals. Consider a fixed term t in f: for each variable v_i, the term can either contain the literal v_i, the negated literal $\neg v_i$ or neither of the both. Thus, there are 3^m possible terms. If f contains k terms, the naive algorithm would have to generate 3^{mk} candidate formulae and check them on the n instances. Consequently, the algorithm's worst case complexity is $O(nm3^{mk}) = O(\exp[1.099mk + \ln n + \ln m])$. This is exponential in the number of variables, but only linear in the number of instances.

This analysis ignores the fact that the hardness of a k-term consistency problem depends crucially on the distribution of positive and negative instances. If the training set contains only one positive instance x_p, a trivial solution is given by the term $\text{lg}(\{x_p\})$, the *least generalization* of a set of instances, defined as follows:

$$\text{lg}(\{x_1, \ldots, x_k\}) := \bigwedge_{j=1}^{m} \sigma_j(\{x_1, \ldots, x_k\}). \tag{4.1}$$

Here, the $\sigma_j(\{x_1, \ldots, x_k\})$ is either an unnegated, a negated, or the empty literal, depending on which of of the three choices is consistent with all instances $\{x_1, \ldots, x_k\}$:

$$\sigma_i(\{x_1, \ldots, x_k\}) := \begin{cases} v_i & \text{if } x_j(i) = 1 \text{ for all } 1 \leq j \leq k, \\ \neg v_i & \text{if } x_j(i) = 0 \text{ for all } 1 \leq j \leq k, \\ 1 & \text{otherwise.} \end{cases} \tag{4.2}$$

It is easy to see that the term $\text{lg}(A)$ covers all instances in A, but as few other instances as possible. In the case of a single instance x_p, $\text{lg}(\{x_p\})$ covers only x_p and no other instances. It is therefore a trivial solution to consistency problems with a single positive instance. On the other hand, if there is only a single negative instance, then the problem can be solved by finding k literals that violate the negative instance but whose disjunction covers all positive instances. Since the naive algorithm described above ignores the fraction of positive and negative instances in the training set, it is not optimal in many cases. For many practical data sets, the number of variables is of the same order of magnitude as the number of instances. Even worse, Rückert *et al.* (2002) showed that for the hardest randomly generated problems in the *phase transition* region (see Section 4.2.2 and Kirkpatrick and Selman (1994) for further details), m is an exponential function of the number of positive instances. The exponential dependency on m in the naive algorithm's time complexity is therefore not optimal.

Fortunately, there are more sophisticated algorithms that depend only linearly on the number of variables. For example, the algorithm described by Rückert *et al.* (2002) has worst case complexity $O(n_n m k^{n_p})$, where n_p is the number of positive instances in the training set, and n_n is the number of negative instances. The main idea is to generate all partitionings of the set of positive instances into k pairwise disjoint partitions $X_1, \ldots, X_k$. Given a particular partitioning $\chi = (X_1, \ldots, X_k)$, the corresponding formula $f_\chi := \lg(X_1) \vee \ldots \vee \lg(X_k)$ is a k-term DNF formula. One can show that whenever a consistency problem has a solution, then there exists at least one partitioning χ so that f_χ is also a solution. Thus, enumerating all partitionings is guaranteed to find a solution, if there exists one. Since the number of possible partitionings of n_p instances into k partitions is the Stirling number of the second kind $S(n_p, k) := \frac{1}{k!} \sum_{i=0}^{k-1} (-1)^i \binom{k}{i} (k-i)^{n_p}$, the complexity of enumerating all partitionings and checking the coverage of negative instances for each partitioning is $O(n_n m k^{n_p}) = O(\exp[n_p \ln k + \ln m + \ln n_n])$.

The two previously described algorithms are simple "enumeration and test" schemes. They do not make any use of the structure in the training set and are therefore unnecessarily inefficient. In the following, we modify the partitioning-based algorithm towards a probabilistic algorithm, which applies pruning techniques to speed up the search. This allows for a worst-case analysis that also depends on structural properties of the training data, rather than simple summary statistics. The pseudo code for the algorithm is given in Algorithm 3. The idea is rather straightforward: FindFormula is called a predefined number r_{max} of times in the main loop of RandomSearch. The loop in FindFormula tries to find a partitioning $(F_1, \ldots, F_k)$ of the positive instances, which corresponds to a consistent formula. To do so, it iteratively adds new positive instances to an initially empty partitioning. As with the partitioning enumeration algorithm above, each partitioning corresponds to a formula, whose terms are the least generalizations of the partitions.

There are two important points to the algorithm's design: First of all, the two random vectors ϕ and π are generated before the loop in FindFormula is executed. The random permutation π determines the order, in which the positive instances are inserted into the partitioning. The vector ϕ contains one uniformly distributed random number between 0 and 1 for each of the n_p positive instances. The ith component of ϕ determines into which partition the ith positive instance is inserted. Secondly, the loop inserts instances only into those partitions, where the insertion does not cause the coverage of a negative example. Thus, while the first few positive examples can be

Algorithm 3 A randomized algorithm for the k-term DNF consistency problem with training set (X, Y).

```
1:  procedure RandomSearch($(X, Y)$, $k$, $r_{max}$)
2:      $P \leftarrow$ the positive instances from $X$
3:      $N \leftarrow$ the negative instances from $X$
4:      for $i = 1$ to $r_{max}$ do
5:          $\pi \leftarrow$ a random permutation of $\{1, \ldots, |P|\}$
6:          $\phi \leftarrow$ a random vector drawn uniformly from $[0,1]^{n_p}$
7:          $f \leftarrow$ FindFormula$(P, N, k, \pi, \phi)$
8:          if $f \neq$ false then
9:              return $f$
10:         end if
11:     end for
12:     return "no solution found"
13: end procedure
14:
15: procedure FindFormula$(P, N, k, \pi, \phi)$
16:     $(F_1, \ldots, F_k) \leftarrow$ the empty partitioning $(\emptyset, \ldots, \emptyset)$
17:     for $j = 1$ to $|P|$ do
18:         $p \leftarrow$ the $\pi_j$th instance in $P$
19:         $B \leftarrow \{b \in \mathbb{N} | 1 \leq b \leq k \wedge \lg(F_b \cup \{p\})$ does not cover any $x \in N\}$
20:         if $B = \emptyset$ then
21:             return false
22:         end if
23:         $b \leftarrow \lceil \phi_j \cdot |B| \rceil$th element from $B$
24:         $(F_1, \ldots, F_k) \leftarrow (F_1, \ldots, F_b \cup \{p\}, \ldots, F_k)$
25:     end for
26:     return $\lg(F_1) \wedge \ldots \wedge \lg(F_k)$
27: end procedure
```

inserted into k different partitions, later instances can only be inserted into $k-1$, $k-2$ or even less partitions, because the insertion into some other partitions F_i would cause the corresponding term $\lg(F_i)$ to cover one or more negative instances. Thus, the algorithm does not test all partitions, but only those that are still able to induce consistent formulae, given the distribution of positive instances processed so far. This is the main tool to bound the worst case running time in the analysis below. Before we delve into the details, we need to introduce some formal concepts.

First of all, we will use P to denote the set of positive instances, and N to denote the set of negative instances. We assume that $P \cap N = \emptyset$, otherwise there is no solution. Furthermore, we assume without loss of generality, that for each variable v_i there are two positive instance $p^0, p^1 \in P$ and two negative instance $n^0, n^1 \in N$ with $p^0(i) = n^0(i) = 0$ and $p^1(i) = n^1(i) = 1$.

If this is not the case, that is, if either all positive or all negative instances assign the same value to some variable v_i, one can create a new k-term decision problem, which has v_i removed and contains less instances than the original problem. The solution to this simpler problem can than be trivially transformed into a solution to the original problem. Now, let $x, x' \in \{0,1\}^m$ be two instances, that is, two m-dimensional bit-vectors. The following definitions make use of the *Kronecker delta* δ_{ij}, which is one, if $i = j$ and zero otherwise. First or all, we will need the well-known *Hamming distance*.

$$d_H(x, x') := \sum_{i=1}^{m} (1 - \delta_{x(i)x'(i)})$$

Then, if $x_1, x_2, x_3 \in P \cup N$ are instances, we define the *3-way similarity* as

$$d_3(x_1, x_2, x_3) := \sum_{i=1}^{m} \delta_{x_1(i)x_2(i)} \delta_{x_2(i)x_3(i)}$$

Finally, consider a subset $B \subseteq P$ of positive instances. If for every proper subset $B' \subset B$, the corresponding term $\lg(B')$ covers no negative instances, but $\lg(B)$ covers at least one negative instance, we say that B is a *block*. Blocks are important for the analysis of the randomized algorithm, because whenever a partition contains all of a block's instances but one, adding the last instance makes the corresponding term cover some negative instance. We are mainly interested in the size $|B|$ of the blocks that are induced by a training set (P, N). Let $\mathcal{B}$ be the set of all blocks of (P, N). Then, $b_s := \max_{B \in \mathcal{B}} |B|$ denotes the maximal block size achievable in $\mathcal{B}$. The following lemma gives an upper bound for b_s:

Lemma 4.2.1. *Let (P, N) be a training set and let $\mathcal{B}$ be the set of all blocks of (P, N). Then*

$$b_s \leq \max_{\substack{p,p' \in P \\ n \in N}} \Big(m + 2 - d_H(p, p') - d_3(p, p', n) \Big)$$

Proof. Let $B \in \mathcal{B}$ be an arbitrary block, let $t = \lg(B)$ be the corresponding term and let n denote the negative instance covered by t. Since $P \cap N = \emptyset$, B contains at least two positive instances. Let p and p' denote the two positive instances in B with the largest Hamming distance $d_H(p, p')$. We partition the set of variables $V = \{v_1, \ldots, v_m\}$ into three disjoint sets. The first set is $V_1 := \{v_i | p(i) \neq p'(i)\}$, the second is $V_2 := \{v_i | p(i) = p'(i) \wedge p(i) = n(i)\}$, and the third is $V_3 := \{v_i | p(i) = p'(i) \wedge p(i) \neq n(i)\}$. The term $\lg(p, p')$ does not contain any variables in V_1 and it agrees with n on all variables in V_2. Therefore, in order for t to cover n, B must contain one instance $p_i \in B \setminus \{p, p'\}$ for each $v_i \in V_3$, where p_i assigns a different value to v_i than p and p'. Thus, $|B| \leq 2 + |V_3|$. Since $|V_1| + |V_2| + |V_3| = m$, $|V_1| = d_H(p, p')$, and $|V_2| = d_3(p, p', n)$, $|B| \leq m + 2 - d_H(p, p') - d_3(p, p', n)$. Taking the maximum over all $p, p' \in P$ and $n \in N$ yields the result. □

With this, we have an easy way to compute an upper bound for b_s from the training set. We can now compute the algorithm's worst case complexity depending on n_p, b_s and k. If there is no solution for a k-term DNF consistency problem, RandomSearch will correctly output "no solution found". If there are one or more solution formulae, let τ denote the probability that a single call to FindFormula will find one of them. The probability that a solution is not found after r independent calls to FindFormula is $(1-\tau)^r \leq e^{-\tau r}$. We want to ensure that a solution is found with probability $1-\delta$ for some small error probability $\delta > 0$. This can be achieved by performing $r = \ln \frac{1}{\delta}/\tau$ independent calls to FindFormula. We will give a lower bound $\tau(n_p, b_s, k) \leq \tau$ in the theorem below. Thus, calling RandomSearch with $r_{max} = \ln \frac{1}{\delta}/\tau(n_p, b_s, k)$ will return the correct result with one-sided error probability at most δ, and it will do so in $O(r_{max} n_p n_n m)$ steps.

Theorem 4.2.2. *Given a soluble k-term consistency problem with training set (P, N) and let τ be the probability that one call to* FindFormula *finds a solution. Then:*

$$\tau \geq \exp\left[-n_D\Big(\ln(k) - \sum_{j=1}^{k-1} \frac{\ln(\frac{k}{k-j})}{(k-1)b_s + 1}\Big)\right].$$

Proof. The procedure FindFormula adds n_p positive instances to an initially empty partitioning in the order $p_{\pi_1}, \dots, p_{\pi_{n_p}}$ induced by the permutation π. Assume a fixed permutation π. The partition, into which an instance is inserted, is chosen uniformly and independently for each instance. Thus, the probability that FindFormula finds one specific solution partitioning $\mathcal{F} = (F_1, \dots, F_k)$ is $\Psi_{\pi,\mathcal{F}} := \prod_{i=1}^{n_p} \psi_{\pi,\mathcal{F}}(i)$, where $\psi_{\pi,\mathcal{F}}(i)$ is the probability that it chooses the correct partition for instance p_{π_i}. The individual $\psi_{\pi,\mathcal{F}}(i)$ depend on the number of partitions the π_ith instance can be added to. If an instance can be added to any of the k partitions without causing the coverage of a negative instance, the corresponding $\psi_{\pi,\mathcal{F}}(i)$ is $\frac{1}{k}$. If there are only $k-1$ partitions to choose from, $\psi_{\pi,\mathcal{F}}(i) = \frac{1}{k-1}$ and so on. Usually, there are many different solution partitionings $\mathcal{F}^{(1)}, \dots, \mathcal{F}^{(t)}$. Thus, given a fixed π, the probability τ_π that FindFormula finds one of the t partitionings is $\tau_\pi := \sum_{i=1}^{t} \Psi_{\pi,\mathcal{F}^{(i)}}$. We would like to lower-bound τ_π, but we do not know t. To avoid the rather messy analysis of τ_π, we analyze a related quantity θ_π instead and show later that $\tau_\pi \geq \theta_\pi$ for all permutations π.

For the definition of θ_π, we consider a modified version FindFormula$_2$ of FindFormula. There are two changes. First of all, instead of starting with an empty partitioning in line 16, we start with a partitioning, that contains exactly one unique artificial instance per partition. The new line is:

$$(F_1, \dots, F_k) \leftarrow (\{\tilde{p}_1\}, \dots, \{\tilde{p}_k\}) \tag{4.3}$$

Here, the $\tilde{p}_1, \dots, \tilde{p}_k$ are artificial instances, which do not appear in P and are used later on to block certain branches of execution. They do not have any further meaning and can be removed from the final partitioning to yield a solution. The second modification is in line 19. Let $\mathcal{B}^e_\pi$ be some set of blocks. The new line is:

$$B \leftarrow \{b \in \mathbb{N} | 1 \leq b \leq k \land \forall B \in \mathcal{B}^e_\pi : (F_b \cup \{p\}) \neq B\} \qquad (4.4)$$

In the original line, the set B contains the indices of those partitions, where the addition of the positive instance p does not cause the coverage of a negative instance. In other words, B contains an index i only if there is no subset $F'_i \subseteq F_i$ of the ith partition so that $F'_i \cup \{p\}$ is a block. Thus, if $\mathcal{B}^e_\pi$ is set to $\mathcal{B}$, the set of all blocks in the training set, the new line is equivalent to the original line.

In our case, though, we will use an extended version of $\mathcal{B}^e_\pi$, which contains all blocks in $\mathcal{B}$, and a few additional blocks, which ensure that FindFormula$_2$ can only find one specific solution partitioning. Let $\Psi^{min}_\pi := \min_{1 \leq i \leq t} \Psi_{\pi, \mathcal{F}^{(i)}}$ denote the smallest summand in the definition of τ_π, and let $\mathcal{F}^{min}$ denote the solution partitioning that achieves this minimum. We will now describe the additional blocks in $\mathcal{B}^e_\pi$, which ensure that FindFormula$_2$ can only find $\mathcal{F}^{min}$ and no other solution partitioning. To do so, consider the point where FindFormula$_2$ has added the first i positive instances and is now evaluating ϕ_{i+1} to determine which partition the π_{i+1}th instance should be assigned to. We assume that the first i instances have been assigned to the "correct" partitions agreeing with $\mathcal{F}^{min}$, so that the procedure would find F^{min}, if the remaining assignments are correct, too. Now, consider the case where FindFormula$_2$ selects the "wrong" partition F_j, so that FindFormula will not find $\mathcal{F}^{min}$. There are two cases. Either the selection of F_j for p can still lead to a different solution partitioning $\mathcal{F} \neq \mathcal{F}^{min}$, or the remaining positive instances can not be distributed among the partitions in a way that yields a solution partitioning. We would like to prevent the first case, but not the second. Thus, we add the new artificial block $\{\tilde{p}_j, p_{\pi_{i+1}}\}$ to $\mathcal{B}^e_\pi$. If we do this for every "wrong" choice and every iteration i, FindFormula$_2$ can only find $\mathcal{F}^{min}$ and no other solution partitioning. With this, θ_π is simply the probability that FindFormula$_2$ finds the only solution $\mathcal{F}^{min}$, when called with the parameter π and when using the extended block set $\mathcal{B}^e_\pi$ described above.

We will now show that the probability τ_π that FindFormula finds a solution is larger than the probability θ_π, that the modified FindFormula$_2$ finds a solution. The only relevant difference between the two procedures is the fact that FindFormula$_2$ can not add an instance to a partition F_j that would lead to a solution different from $\mathcal{F}^{min}$. We compare FindFormula and FindFormula$_2$ in the case where the first i instances are distributed "correctly" according to partitioning $\mathcal{F}^{min}$, and the $i + 1$th instance $p_{\pi_{i+1}}$

needs to be assigned to a partition. In the case of FindFormula we can have $a \leq k$ different choices. Let b denote the number of choices, which could possibly lead to another solution different from $\mathcal{F}^{min}$, and let $\bar{\psi}_1, \ldots, \bar{\psi}_b$ denote the corresponding probabilities that FindFormula will find such a solution, after choosing a "wrong" partition. The probability that FindFormula finds a solution at this point is therefore $\frac{1}{a}(\bar{\psi}_{min} + \sum_{j=1}^{b} \bar{\psi}_j)$, where $\bar{\psi}_{min}$ denotes the probability that FindFormula finds $\mathcal{F}^{min}$ after choosing the "correct" partitioning. Since $\mathcal{F}^{min}$ is the solution with minimal probability, $\bar{\psi}_j \geq \bar{\psi}_{min}$ for all j. This means that the probability that FindFormula finds a solution at this point is larger than $\frac{b+1}{a}\bar{\psi}_{min}$. On the other hand, FindFormula$_2$ can only choose between $a - b$ different partitions, and only one of them leads to a solution. The probability is therefore $\frac{1}{a-b}\bar{\psi}_{min}$. Since $\frac{b+1}{a} \geq \frac{1}{a-b}$ for $a \geq b+1$, the probability that FindFormula finds a solution is larger than the probability that FindFormula$_2$ does. Applying this inequality recursively in all iterations, where FindFormula differs from FindFormula$_2$ leads to the claim that $\tau_\pi \geq \theta_\pi$. This is true for every possible permutation π, and therefore also for the expectations $\mathbf{E}_\pi \, \tau_\pi \geq \mathbf{E}_\pi \, \theta_\pi$.

As a final step, we give a probabilistic analysis of FindFormula$_2$ to lower bound $\mathbf{E}_\pi \, \theta_\pi$, the expected probability that one call to FindFormula$_2$ will find a solution. Because of the discussion above, this is also a lower bound of $\tau = \mathbf{E}_\pi \, \tau_\pi$. Recall that for a specific π, $\theta_\pi = \prod_{i=1}^{n_p} \psi_{\pi,\mathcal{F}}(i)$. The individual $\psi_{\pi,\mathcal{F}}(i)$ are either $\frac{1}{k}$ or $\frac{1}{k-1}$ or ... or 1, depending on how many partitions are blocked at iteration i. Let $nbl(j,\pi)$ denote the number of iterations, where j among the k possible partitions are blocked, or equivalently, the number of factors of the form $\frac{1}{k-j}$ in the product above. With this we can merge the factors as follows:

$$
\begin{aligned}
\mathop{\mathbf{E}}_{\pi}\left[\theta_\pi\right] &= \mathop{\mathbf{E}}_{\pi}\left[\prod_{j=0}^{k-1}\Big(\frac{1}{k-j}\Big)^{nbl(j,\pi)}\right] \\
&= \mathop{\mathbf{E}}_{\pi}\left[\exp\Big[-\sum_{j=0}^{k-1} nbl(j,\pi)\ln(k-j)\Big]\right] \\
&\geq \exp\left[-\mathop{\mathbf{E}}_{\pi}\Big[\sum_{j=0}^{k-1} nbl(j,\pi)\ln(k-j)\Big]\right] && (4.5) \\
&= \exp\left[-\sum_{j=0}^{k-1}\ln(k-j)\mathop{\mathbf{E}}_{\pi}\big[nbl(j,\pi)\big]\right], && (4.6)
\end{aligned}
$$

where (4.5) is a consequence of the convexity of the exponential function. We will now compute the $\mathbf{E}_\pi \, nbl(j,\pi)$ for arbitrary j. Observe, that for each permutation π and each positive instance p_i, there is exactly one partition F_j, which gives rise to a solution. If one moves one particular positive instance p from F_j to a different partition F_l with $l \neq j$, the corresponding formula will not be a solution. That means that there must be a block $B_l \in \mathcal{B}^e_\pi$, which contains p and some other instances in F_l. The maximum size of B_l is b_s, because this is the maximum size for the blocks in $\mathcal{B}$ and the additional blocks added for FINDFORMULA$_2$ are of size $2 \leq b_s$. Let $x_1, \ldots, x_{b_s}, p$ denote the examples in the block B_l.

Let us assume for a moment that the blocks have maximum size b_s and that they are pairwise disjoint, i.e. $B_l \cap B_k = \emptyset$ for $k \neq l$. Then, the probability that all partitions are blocked for p_{π_i} is equal to the probability that p_{π_i} is last among $\bigcup_{l \neq j} B_l$. Since all permutations are equally likely, this probability is $\frac{1}{(k-1)b_s+1}$. Moreover, this is also a lower bound for the probabilities that $k-2, k-3, \ldots$ or no options are blocked, because for every permutation, which has p_{π_i} last, there is a permutation that sorts p_{π_i} before other instances from the blocks. Define $P^{nbl}_j := \mathbf{Pr}[$"instance p_{π_i} has j partitions blocked"$]$. The distribution of P^{nbl}_j that minimizes (4.6) subject to the constraint that $P^{nbl}_j \geq \frac{1}{(k-1)b_s+1}$ for all $j \geq 0$ is given by $P^{nbl}_0 = 1 - \frac{k-1}{(k-1)b_s+1}$ and $P^{nbl}_j = \frac{1}{(k-1)b_s+1}$ for all $1 \leq j \leq k-1$. Thus, we have the following lower bound:

$$\begin{aligned}
\tau &\geq \exp\left[-\ln(k)\Big(n_p - \frac{n_p(k-1)}{(k-1)b_s+1}\Big) - \sum_{j=1}^{k-1}\frac{\ln(k-j)n_p}{(k-1)b_s+1}\right] \\
&= \exp\left[-n_p\Big(\ln(k) - \sum_{j=1}^{k-1}\frac{\ln(\frac{k}{k-j})}{(k-1)b_s+1}\Big)\right]
\end{aligned}$$

The case where some active blocks are not pairwise disjoint or where some blocks do not have maximal size b_s corresponds to selecting a smaller b_s in the denominator of the fraction in the inequality above. Since doing so would improve the bound, so the result is valid for those cases, too. □

It is interesting to note that the number of negative instances is not used directly in the theorem, but only through the upper bound on b_s provided by lemma 4.2.1. This is a much more natural measure for the hardness of a k-term DNF consistency problem, because it measures the strength of the constraints imposed by the negative instances on the solution candidates. As an immediate consequence of the theorem and the considerations above, we can state that RANDOMSEARCH $\in$ RTIME$(\exp[\ln n_p + \ln n_n + \ln m + \tau])$.

4.2.2 Stochastic Local Search for k-term DNF Maximum Agreement

The k-term DNF maximum agreement problem is considerably harder to solve than the k-term DNF consistency problem. This is due to two reasons. First of all, it is a combinatorial optimization problem rather than a decision problem. The goal is to determine the DNF formula with the best possible error rate rather than deciding whether there exists a consistent formula. The problem is that there is no easy way to check whether a current candidate solution is indeed optimal. In order to prove that a DNF formula is optimal, an algorithm needs to ensure that there is no DNF formula with a smaller error. This is much more costly, because it needs to check many candidates that are "almost solutions". Of course, one can use branch-and-bound methods to convert any optimization problem into a sequence of decision problems with varying thresholds. In our case, one would iteratively solve the *decision versions* of the problem, which answer the question "Is there a k-term DNF formula with empirical error less than a predefined threshold t?". If the t are selected in a binary search fashion, one needs to solve at most $\log_2 n$ such decision problems to identify the DNF formula with optimal error.

Unfortunately, the decision version of this optimization problem is still harder than the k-term DNF consistency problem. Whereas the consistency problem demands that *all* training instances are classified correctly, the decision version of the optimization problem requires only a fraction of instances to be consistent, but it does not specify which ones need to be consistent. If it is known which instances need to be consistent, one can simply apply the randomized algorithm in the preceding section to those training instances to yield an optimal solution. Unfortunately, it is hard to obtain this information, and in the worst case, an algorithm would need to check $\binom{n}{t}$ possible combinations, thus increasing the runtime by an additional $O((\frac{ne}{t})^t(\frac{n}{n-t})^{n-t})$ factor. While one could certainly apply techniques similar to the ones in the preceding section to somewhat alleviate this penalty, there is little hope to solve even medium-sized problem instances of the k-term DNF maximum agreement problem within reasonable time frames.

In order to still gain good approximate solutions to k-term DNF empirical risk minimization problems, we need to resort to other, more heuristic approaches. In recent years, *stochastic local search* (SLS) algorithms have been shown to be remarkably successful on similar NP-hard combinatorial problems. In particular, they are among the best algorithms available to solve hard satisfiability problems, see for example Hirsch and Kojevnikov (2005). SLS algorithms differ from other approaches in that they perform a local, randomized-walk search. In its basic incarnation, an SLS algorithm starts with a randomly generated solution candidate. It then iterates in a two-step loop. In the first step it examines a fixed set of "neighboring" candidates according to some predefined neighborhood relation. Each neighbor is evaluated according to some global scoring function. In the second step the SLS algorithm selects the neighbor with the highest score as next candidate. Such a greedy hill climbing approach is obviously susceptible of getting caught in local optima. Most SLS algorithms therefore select with

Algorithm 4 A generic stochastic local search algorithm.

```
procedure SLSearch($p$)
    $c \leftarrow$ a randomly generated candidate
    while $score(c) \neq \max$ do
        $n \leftarrow$ the set of all neighbors of $c$
        with probability $p$ do
            $c \leftarrow$ a random neighbor in $n$
        otherwise
            $s \leftarrow$ the scores of the neighbors in $n$
            $c \leftarrow$ the neighbor in $n$ with the highest score
        end with
    end while
    return $c$ with the best score so far
end procedure
```

a fixed probability p (the so-called *noise probability*) a random neighbor instead of the neighbor with the highest score. In this way they can escape local optima through random steps. Algorithm 4 sketches the main concept.

Technically, SLS algorithms are *Las Vegas algorithms*, that is, nondeterministic algorithms that output a correct solution, if they terminate. Because of the non-determinism, the runtime of a Las Vegas algorithm is a random variable. Since in practice one is not willing to wait forever, SLS algorithms are usually implemented to stop after a certain maximum runtime (the so called *cutoff time*) and output "no solution found". This yields a *Monte Carlo algorithm*, that is, a non-deterministic algorithm which might with a certain probability output a wrong result. Sometimes the average runtime of an SLS algorithm on a specific set of problems can be improved by selecting a comparably low cutoff time and using frequent restarts (Gomes *et al.*, 1998; Hoos, 1998).

When designing an SLS algorithm for k-term DNF maximum agreement, one has to decide about a suitable candidate space first. Using the space of k-partitions, just as with the randomized search algorithm described in Section 4.2.1 seems to be an obvious choice. Unfortunately, calculating the neighboring formulae for a given candidate in k-partition space is a relatively time consuming task, because it requires the computation of two *lgg*s per neighbor. From our experiments (Rückert, 2002) it seems to be more time-efficient to use k-term DNF formulae as candidates and to add or remove literals to generate the neighboring candidates. The obvious choice for the global scoring function is the empirical error $score_P(f) := \hat{\varepsilon}_f$. We are therefore aiming for minimal, not maximal score.

Another important design issue for SLS algorithms is the choice of the neighborhood relation and the decision rule. The decision rule specifies which of the neighbors is chosen as the next candidate. A straightforward

Algorithm 5 An SLS algorithm for k-term DNF learning.

```
procedure SLSearch($k$, $maxSteps$, $p_{g1}$, $p_{g2}$, $p_s$)
    $H \leftarrow$ a randomly generated $k$-term DNF formula
    $steps \leftarrow 0$
    while $score_L(H) \neq 0$ and $steps < maxSteps$ do
        $steps \leftarrow steps + 1$
        $ex \leftarrow$ a random example that is misclassified by $H$
        if $ex$ is a positive example then
            $t \leftarrow \begin{cases} \text{with probability } p_{g1}\text{: a random term in } H \\ \text{otherwise: the term in } H \text{ that differs} \\ \text{in the smallest number of literals from } ex \end{cases}$
            $l \leftarrow \begin{cases} \text{with probability } p_{g2}\text{: a random literal in } t \\ \text{otherwise: the literal in } t \text{ whose removal} \\ \text{decreases } score_L(H) \text{ most} \end{cases}$
            $H \leftarrow H$ with $l$ removed from $t$
        else if $ex$ is a negative example then
            $t \leftarrow$ a (random) term in $H$ that covers $ex$
            $l \leftarrow \begin{cases} \text{with probability } p_s\text{: a random literal } m \\ \text{so that } t \wedge m \text{ does not cover } ex \\ \text{otherwise: a random literal whose} \\ \text{addition to } t \text{ decreases } score_L(H) \text{ most} \end{cases}$
            $H \leftarrow H$ with $l$ added to $t$
        end if
    end while
    return $H$
end procedure
```

decision rule for k-term DNF learning SLS algorithms could evaluate all formulae that differ from the current candidate by one literal and choose the neighboring formula with the lowest score. As it turns out, this approach is not very effective (Rückert, 2002). In the following we present a more target-driven decision process to boost the search.

The main idea is to concentrate on those changes that will correct the misclassification of at least one misclassified example. Assume p is an uncovered positive example. Thus, the current candidate formula c is obviously too specific and we have to remove at least one literal in order to satisfy p. Of course it does not make sense to modify a random term in c, because one might then affect a large number of currently correctly classified examples. A more sensible strategy would generalize the term t in c that differs in the smallest number of literals from p. One can then evaluate the formulae that differ in one literal in t as neighbors and choose that neighbor with the lowest score.

Similar considerations can be made for adding literals: assume n is a

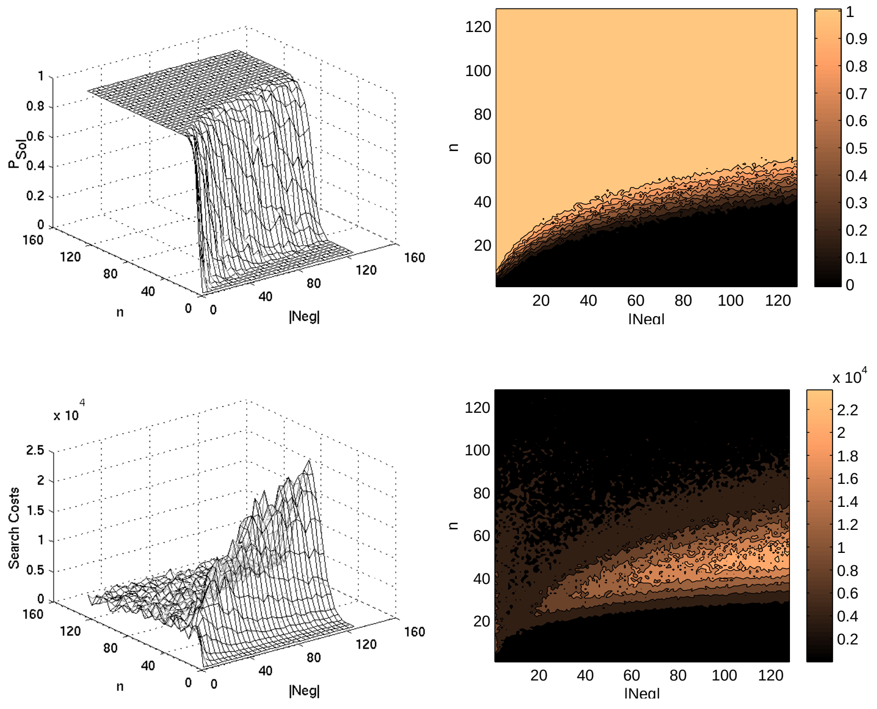

Figure 4.1: P_{Sol} (above) and search costs (below) plotted as 3D graph (left) and contour plot (right) for the problem settings with $k = 3$, $|Pos| = 15$, $1 \leq |Neg| \leq 128$, and $1 \leq n \leq 128$

covered negative example. Then the current formula is obviously too general. Let t be the term that covers n. We have to add a literal to t in order to make n uncovered. Again we can generate a set of neighbors by adding one literal to t and then choose that neighbor whose score is lowest. This consideration leads to a decision rule and a neighborhood relation for the final algorithm, which use as much information as possible to guide the search: The decision rule first selects a random misclassified example e. If e is an uncovered positive example, the algorithm performs a generalization step as explained above; if e is a covered negative example, it performs a specialization step. Of course, this algorithm can get stuck in a local optimum quite quickly. It makes sense to replace each decision step with a random choice from time to time to escape those local optima. There are two decisions to be made during a generalization step and one decision for the specialization step. Thus, we have three different places to perform a random instead of an informed decision with certain probabilities. Algorithm 5 sketches the idea.

How well does the SLS algorithm perform on hard k-term DNF maximum agreement problems? Unfortunately, it is not trivial to assess the hardness of such a combinatorial optimization problem. Contrary to intuition, the number of training instances or attributes are not very useful to assess a problem instance's hardness. For example, even though the test sets used in Kamath *et al.* (1992) have up to one thousand examples, they can be solved by a randomized version of an exhaustive search algorithm in less than a second (Rückert, 2002). On the other hand, some problem instances with only sixty examples can take weeks to solve.

One way to assess problem hardness is the phase transition framework. Consider, for instance, maximum agreement problem instances where m, the number of positive examples n_p and k are fixed and the number of negative examples n_n is varied. One would expect to have—on average—low search costs for very low or very high n_n. With only a few negative examples, almost any formula covering the positive instances should be a solution, hence the search should terminate soon. For very large n_n, we can rarely generate formulae covering even a small subset of the positive instances without also covering one of the many negative examples. Consequently, we can prune the search early and search costs should be low, too. Only in the region between obviously soluble and obviously insoluble problem instances, the average search costs should be high. Similar considerations can be made about m, n_p, and k. This transition between obviously soluble and obviously insoluble problem settings resembles the phase transition in physical systems. Recent research has shown that the hardest random problems are located in this "phase transition area" (Rückert *et al.*, 2002): Figure 4.1 shows $P_{Sol} := \mathbf{Pr}[\text{"instance soluble"}]$ and average search costs for randomly generated problem instances with $k = 3$, $n_p = 15$, $1 \leq n_n \leq 128$, and $1 \leq n \leq 128$. As can be seen, the problem instances whose average probability of being soluble is approximately 0.5 require the largest average search costs. A recent study (Rückert and Kramer, 2003) compares the performance of various SLS algorithms on hard problems taken from this phase transition region. As it turns out, the system presented above outperforms the other established systems such as WalkSAT (Selman *et al.*, 1996) and Novelty (McAllester *et al.*, 1997). These results give some confidence in the algorithm's ability to solve or at least find good approximative solutions to hard maximum agreement problems. In the next section we will extend it with methods for capacity control and investigate it empirically.

4.3 Structural Risk Minimization for DNF Rule Sets

So far we have dealt with "pure" empirical risk minimization for DNF rule sets of a certain restricted size k. Let us now address the question of how to choose a suitable k. This is essentially a capacity control problem. One popular approach to solve this problem is *structural risk minimization*, introduced in Section 3.4.2. Recall from Inequality (3.6) that the expected error of a classifier can be upper-bounded as follows:

$$\mathbf{Pr}\left[\varepsilon_h \leq \hat{\varepsilon}_h + \sqrt{\frac{\ln|\mathcal{H}| + \ln\frac{1}{\delta}}{2n}}\right] \geq 1 - \delta$$

As usual, ε_h is the true risk of hypothesis h, $\hat{\varepsilon}_h$ is its training set error, n the number of instances in the training set, $\mathcal{H}$ the hypothesis class and δ a small mistake probability. Thus, in order to aim for high predictive accuracy or, equivalently, low expected error, one would want to select a hypothesis with a small training set error $\hat{\varepsilon}_h$ from a small hypothesis class $\mathcal{H}$, so that both summands on the right side of the inequality are small. Unfortunately, if the hypothesis class is too small, it is unlikely that it contains a hypothesis with small training set error, so $\hat{\varepsilon}_h$ is large and the bound is loose. On the other hand, if $\mathcal{H}$ is very large, then it will be easy to find a hypothesis with small training error, but $\ln|\mathcal{H}|$ is large and the bound is loose again. In order to avoid under- and overfitting, one should select the best hypothesis from a hypothesis class whose size is just in between. In our case, the size of the hypothesis class is essentially determined by k, the maximum size of the rule sets in the class. In principle, one could compute the right hand side of the inequality above to assess the structural risk for a hypothesis class with a given maximum number of rules k. Unfortunately, the bound holds for all possible data-generating distributions D and is thus a worst-case rather than an average-case estimate. Choosing a good estimation procedure can be a challenging task. However, as put by Bartlett *et al.* (2002), it is clear that "good error estimation procedures provide good model selection methods". We therefore choose ten-fold cross validation as a popular and established error estimation technique to perform capacity control.

4.3.1 The SL^2 Rule Learner

With this, we can formulate an easy structural risk minimization scheme to learning small, but predictive rule sets. The algorithm starts with $k = 1$, that is, the class of all rule sets of size one. It uses the SLSearch algorithm as explained in Section 4.2.2 in an internal tenfold cross validation to estimate the predictive accuracy of the rule set with the best training accuracy in this class. Then, it iteratively repeats the same procedure for $k = 2$, $k = 3$, etc. as long as the accuracy estimate increases. As soon as the accuracy estimate for one size k is lower than that for the preceding size $k - 1$, the algorithm selects $k - 1$ as the optimal hypothesis class size. It then applies SLSearch on the whole learning set to identify a rule set r_k with low empirical error within this class. We call this rule learning procedure SL^2 (Stochastic Local Search Learner).

So far, the system has concentrated on minimizing the number of rules, but not the number of literals. The following postprocessing step aims at

Algorithm 6 A modified SLS algorithm for finding a set of k rules with at most l_{max} literals that misclassifies as few examples as possible.

```
procedure SLSEARCH($k$, $maxSteps$, $p_{g1}$, $p_{g2}$, $p_s$, $l_{max}$)
    $H \leftarrow$ a randomly generated set of $k$ rules
    for $j \leftarrow 1$ to $maxSteps$ do
        if $H$ has more than $l_{max}$ literals then
            $ex \leftarrow$ a random positive example that is misclassified by $H$
        else
            $ex \leftarrow$ a random example that is misclassified by $H$
        end if
        if $ex$ is a positive example then
            $t \leftarrow \begin{cases} \text{with prob. } p_{g1}\text{:} & \text{a random rule in } H \\ \text{otherwise:} & \text{the rule in } H \text{ that differs in the smallest number of literals from } ex \end{cases}$
            $l \leftarrow \begin{cases} \text{with prob. } p_{g2}\text{:} & \text{a random literal in } t \\ \text{otherwise:} & \text{the literal in } t \text{ whose removal decreases } score_L(H) \text{ most} \end{cases}$
            $H \leftarrow H$ with $l$ removed from $t$
        else if $ex$ is a negative example then
            $t \leftarrow$ a (random) rule in $H$ that covers $ex$
            $l \leftarrow \begin{cases} \text{with prob. } p_s\text{:} & \text{a random literal } m \text{ so that } t \wedge m \text{ does not cover } ex \\ \text{otherwise:} & \text{a random literal whose addition to } t \text{ decreases } score_L(H) \text{ most} \end{cases}$
            $H \leftarrow H$ with $l$ added to $t$
        end if
    end for
    return the $H$ with the lowest score found so far
end procedure
```

minimizing the number of literals in r_k without compromising its training set accuracy. To do so, we augment SLSearch with an additional parameter l_{max} that states an upper limit on the number of *literals* in the rule set. The stochastic local search then considers only formulae with less than the specified number of literals. The modified SLSearch algorithm is given in Algorithm 6. The postprocessing loop calls SLSearch repeatedly with decreasing values of the l_{max} parameter. It stops as soon as it fails to find a rule set that has the same or better training set accuracy than the unrestricted rule set r_k. The rule set with the smallest number of literals is then output as the final result. In other words, SL^2 outputs the rule set with the smallest number of literals among the rule sets with the best training set accuracy in the hypothesis class of sets of size k.

In a sense, SL^2 is designed to feature a clean and well-motivated bias:

it aims at rule sets with the smallest possible number of rules, which still exhibit good predictive accuracy. If there are many rule sets with the same (training) error, the post-processing step selects the one with the least number of literals. As such, SL^2 is a "pure" approach to find the best compromise between predictive accuracy and simplicity. However, there are two disclaimers to be considered. First of all, as already mentioned, we can not expect that SL^2 features high predictive accuracy in all possible cases. For some learning problems, a different bias (for instance, towards some *large* rule sets of a certain form) might work better. The system is designed to work well on a broad range of problems as it is designed from basic statistical induction principles. In particular it is guaranteed to minimize the estimation error in the limit. The approximation part of the error, though, depends on the bias and it is not clear whether a bias towards small rule sets works well in all cases. The study in the next section addresses this question empirically. The second disclaimer deals with the fact that one could also use the number of literals as the main indicator of simplicity. It is clear that one can modify SL^2 to optimize the number of literals directly in the cross-validation loop and then minimize the number of rules later on. However, the change of predictive accuracy caused by removing a literal from a DNF formula is usually minor when compared to the natural random fluctuation of the tenfold cross-validation estimate. We chose the current setup because it is more coarse-grained and therefore more efficient and stable to use in combination with cross-validation. Also, as Michalski (1983), we feel that the number of rules is a more natural indicator of simplicity than the number of literals when comparing different rule sets.

4.3.2 Experiments

The goal of the experiments in this section is to investigate how SL^2 compares to state-of-the-art concept-learners in terms of predictive accuracy and simplicity. In particular, we aim to answer the following three questions.

1. How does the predictive accuracy of SL^2 (as an algorithm that is designed to optimize for small rule sets) compare to the accuracy of state-of-the-art concept-learning algorithms? This gives us some feedback on whether or not the bias for small rule sets performs worse than other established biases.

2. How does the size of the rule sets that were induced by SL^2 compare to the size of those learned by state-of-the-art rule learning algorithms?

3. How does the number of literals that were induced by SL^2 compare to the number of those learned by state-of-the-art rule learning algorithms?

For our investigation, we choose all two-class concept learning data sets used by Frank and Witten (1998). All of the data sets are available from the UCI repository and have been used in many other empirical investigations (for example, Frank and Witten, 1998; Holte, 1993; Chisholm and Tadepalli, 2002; Cohen, 1995; Lavrač *et al.*, 2004). Since the basic SL^2 algorithm handles only Boolean attributes without missing values, we transformed the data sets for use in our experiments. The goal of this transformation was to keep as much information of the original data sets as possible. For nominal attributes, we introduced one Boolean attribute per attribute value. For continuous attributes, we used simple frequency based discretization to replace the attribute with ten new Boolean attributes[2]. Some data sets also contained missing values. To keep the information "value is missing" we added a new Boolean variable for attributes with missing values. This Boolean attribute was set to 1, if the value was missing and 0 otherwise. We used the default parameters $p_{g1} = 0.2$, $p_{g2} = 0.1$, $p_S = 0.1$ and $maxSteps = 100,000$. These parameters have been found to deliver good performance on hard rule minimization problems by Rückert and Kramer (2003).

We employed the following rule learners in our investigation:

- The benchmark system SL^2 as described above.
- CN2 (Clark and Niblett, 1989) as an example of a pure covering algorithm using beam search and an entropy based heuristic.
- CN2-WRACC (Todorovski *et al.*, 2000) is a modification of the CN2 algorithm that uses the weighted reduced accuracy heuristic instead of the entropy based one.
- PART (Frank and Witten, 1998) uses the standard covering algorithm to generate a decision list, but avoids over pruning by obtaining rules from partial decision trees.
- RIPPER (Cohen, 1995) employs the covering algorithm twice: first, it generates an initial rule set using the set covering algorithm with incremental reduced error pruning. Then, in the optimization step, it replaces each rule with a modified one if the modified rule is better according to an MDL heuristic. Finally, it applies the covering algorithm a second time to cover any remaining positive examples. The stopping criterion of the covering algorithm is based on the total description length of the rule set and the examples. RIPPER thus explicitly aims at finding small rule sets. We tested RIPPER both with and without the global optimization step.

[2]Using a larger number of discretization intervals would have increased the number of attributes to a large extent, possibly exceeding the limits imposed by some learners.

System	Win	Loss	Draw	Prob
CN2Orig	5	5	8	0.472
RIPPERwOpt	5	3	10	0.435
C50	5	5	8	0.327
RIPPERwoOpt	4	5	9	0.879
CN2WRACC	5	6	7	0.586
PART	5	8	5	0.381
FOIL	1	13	4	<0.001
DecStump	2	7	9	0.094

Table 4.1: Results: the number of wins/losses/draws of the tested algorithms compared to SL^2 together with the p-value of a signed Wilcoxon rank test on the outcome.

- C5.0 (J, 2003) is an improved version of the classic C4.5 decision tree learning algorithm. As such it employs a divide-and-conquer approach rather than a separate-and-conquer one. After inducing the trees, there is a post-processing step that produces the rules.

- FOIL[3] (Quinlan, 1990) employs the covering algorithm using a two-stage pruning approach and an MDL-based stopping criterion to induce rules expressed in first order logic. To be able to run FOIL on our data sets, we used a straightforward conversion of our Boolean descriptions into a relational format.

Unless noted otherwise we used the default values provided by the implementations for the algorithms' various parameters. As another benchmark classifier we also include the WEKA implementation of a Decision Stump (that is, a decision tree of height one) learner in the survey. As argued by Holte (1993), decision stumps are representationally very simple, but often feature only slightly worse predictive accuracy than more sophisticated approaches. They are therefore well suited as a baseline classifier with a fixed, very high simplicity, but unknown predictive accuracy.

To obtain an answer to our first question concerning the relative accuracy of the SL^2 algorithm (see Table 4.2), we performed ten runs of each system on each data set, where each run consists of a ten-fold cross validation. Each value in Table 4.2 denotes the mean and the standard deviation over the ten runs. Results are marked with a bullet ("•"), if they are significantly worse according to a paired two-sided t-test than the SL^2 result

[3]We performed some preliminary experiments with PFOIL (Mooney, 1995), a propositional version of FOIL. Unfortunately, PFOIL always induces rule sets with zero training error, so it strongly overfits on the data sets in this study. FOIL uses a MDL heuristic to generate small rule sets, so it was more suitable for this study.

on the 1% level, and are marked with a circle ("$\circ$"), if they are significantly better. Table 4.1 denotes the number of data sets on which SL^2 had significantly better, significantly worse or comparable accuracy. Following the methodology recommended by Demšar (2006), the rightmost column gives the probability that a system has the same average predictive accuracy as SL^2 according to a signed Wilcoxon ranks test. The results indicate that there is no statistically significant difference beween SL^2's and the other system's predictive accuracy, with the exception of FOIL and decision stumps. On some data sets (such as heart-statlog), SL^2 is clearly outperformed by the other algorithms, but there are also data sets (such as labor), where SL^2 performs better than the competition. All in all, SL^2 achieves good predictive accuracy in most cases. The fact that decision stumps do suprisingly well (it features the best accuracy of all algorithms on two data sets) indicates that for the presented data sets overfitting seems to be an important issue.

The situation is much more pronounced for what concerns the second question. The sizes of the rule sets and their standard deviations are shown in Table 4.3. We applied again a paired two-sided t-test and annotated the results in the same way as before. The table shows that SL^2 delivers significantly smaller rule sets than its competitors in almost all cases, and is inferior to those of the other algorithms in only three comparisons. CN2 with WRACC is the most competitive system. It induces smaller rule sets for the kr-vs-kp and the mushroom data sets, both times at the expense of predictive accuracy when compared to SL^2. Interestingly, there are a lot of extreme cases, such as CN2 learning 124 rules on the german data set, when SL^2 generates one or two rules. Often, the difference in the size of the rule sets is grossly disproportionate to the difference of the corresponding predictive accuracies.

Even though SL^2 is not particularly aggressive in minimizing the number of literals, the results on the third question in Table 4.4 show a similar picture. Again, CN2 with WRACC is the most competitive system, deriving less literals than SL^2 in four cases. In general, though, the competing systems tend to generate many more literals than necessary. All in all, the results confirm that a "pure bias" towards small rule sets can work well in practice and that the investigated established rule learners tend to generate unnecessarily large rule sets. We therefore believe that our investigation indicates that an important direction for further research in rule learning is concerned with searching at the level of rule sets, rather than at the level of individual rules, and also with techniques for obtaining small rule sets. Another, important area of research for rule learning concerns capacity control with ensembles rather than regularized risk minimization. We present such an approach in the next section.

Data set	CN2 (Orig)	RIPPER w Opt	C5.0	RIPPER wo Opt	CN2 WRACC	PART	FOIL	Dec Stump	SL^2
australian	84.3±1.1	85.0±0.4●	85.1±0.7	84.4±1.0●	85.1±0.3	83.4±0.8●	71.9±1.0●	85.5±0.0	85.4±0.2
breast-cancer	71.0±1.6	71.2±1.6	70.2±1.1	71.0±1.9	69.5±1.0	66.3±1.3●	67.2±1.8●	70.3±1.4	70.1±1.8
breast-w	94.1±0.6	94.4±0.4	94.4±0.3	94.0±0.6	94.5±0.4	94.9±0.7○	93.8±0.5	88.3±0.0●	94.1±0.6
german	72.0±0.9○	70.6±1.0	71.9±1.1○	70.9±0.6	68.5±0.5●	68.8±1.3	60.9±1.1●	70.0±0.0	70.0±0.9
glass(G2)	77.2±2.0○	76.7±1.6○	75.5±3.2	73.3±3.0	70.4±2.6	75.8±2.7○	67.6±2.6●	55.8±0.8●	71.6±2.7
heart-c	77.0±1.6○	79.5±2.0○	77.8±1.5○	79.9±1.3○	75.1±1.4○	77.4±1.3○	70.4±2.2	72.8±1.7	72.9±1.5
heart-h	79.2±1.1●	79.8±1.0	80.1±0.9	79.4±1.3	79.3±1.3●	77.6±2.0●	72.6±2.7●	80.3±1.2	80.5±0.8
heart-statlog	77.9±1.5○	80.6±1.3○	78.9±1.2○	80.4±1.7○	76.4±0.8○	76.7±2.2○	71.2±1.3	72.5±0.9	72.1±1.4
hepatitis	81.2±1.5	80.1±1.9	79.4±1.4●	78.7±1.8●	78.1±1.1●	79.7±1.8	76.3±2.4●	82.6±1.2○	81.3±1.4
horse-colic	81.6±0.8	85.1±0.8○	82.7±1.2	83.3±1.0○	83.2±0.7○	79.0±2.5●	75.4±1.5●	81.5±0.2	81.5±0.4
ionosphere	89.7±0.9●	89.2±1.0●	90.4±1.0	88.9±0.8●	91.9±0.5	87.7±1.5●	83.7±2.0●	82.6±0.0●	91.1±1.1
kr-vs-kp	98.3±0.1	99.3±0.1○	99.3±0.1○	98.9±0.1○	94.1±0.0●	99.1±0.1○	99.2±0.1○	66.1±0.0●	98.1±0.5
labor	88.1±3.0●	83.7±4.0●	87.9±0.9●	80.2±4.2●	91.2±1.1●	88.2±0.8●	86.0±1.6●	79.5±2.1●	94.2±3.2
mushroom	100.0±0.0●	100.0±0.0	99.9±0.0●	100.0±0.0	98.5±0.0●	100.0±0.0	100.0±0.0	88.7±0.0●	100.0±0.0
pima-indians	69.7±0.8●	72.7±0.9	70.4±1.4●	71.8±0.6	71.0±0.9	69.3±1.1●	60.6±1.2●	72.1±0.0	72.1±1.0
sick	93.9±0.0	93.8±0.2	93.7±0.1●	93.8±0.2	93.9±0.0	92.8±0.2●	92.8±0.2●	93.9±0.0	93.8±0.0
sonar	68.8±3.1○	61.9±1.8	64.4±1.6	58.3±2.1●	66.6±1.7○	65.6±2.9	52.8±2.9●	51.1±1.3●	63.6±3.1
vote	94.9±0.5	94.8±0.5	95.6±0.5○	95.0±0.3	95.2±0.4○	94.3±1.0	92.8±0.5●	95.5±0.3○	94.3±0.8
average	83.3±1.2	83.2±1.1	83.2±1.0	82.4±1.2	82.4±0.8	82.0±1.3	77.5±1.4	77.2±0.6	82.6±1.2

Table 4.2: Results: percentage of correct classifications and standard deviation.

Data set	CN2 (Orig)	RIPPER w Opt	C5.0	RIPPER wo Opt	CN2 WRACC	PART	FOIL	SL^2
australian	44.9±0.7●	2.9±0.4●	11.8±0.7●	4.9±0.7●	2.0±0.0●	22.8±1.0●	22.8±0.8●	1.0±0.0
breast-cancer	40.0±1.0●	1.4±0.2○	6.8±0.6●	3.6±0.8●	2.8±0.1●	25.9±1.0●	10.8±0.6●	1.8±0.2
breast-w	19.6±0.3●	6.1±0.8●	7.2±0.2●	6.3±0.7●	3.0±0.0●	5.3±0.6●	9.6±0.3●	1.3±0.1
german	124.0±1.6●	3.6±0.4●	21.2±1.2●	5.1±0.7●	2.3±0.1●	60.5±1.5●	36.3±2.1●	1.5±0.2
glass(G2)	21.5±0.4●	5.6±0.2●	7.6±1.0●	6.2±1.0●	4.2±0.3●	7.4±0.5●	6.1±0.2●	1.6±0.2
heart-c	31.3±0.5●	3.5±0.2●	11.3±0.8●	5.1±0.6●	2.9±0.2●	12.4±0.7●	12.4±0.6●	1.3±0.2
heart-h	27.1±0.8●	2.2±0.2●	8.9±0.9●	3.5±0.5●	2.7±0.2●	13.3±0.8●	11.1±0.5●	1.1±0.1
heart-statlog	31.8±0.6●	3.6±0.2●	9.7±0.5●	5.0±0.7●	3.4±0.2●	10.7±0.4●	12.6±0.6●	1.2±0.2
hepatitis	16.4±0.4●	1.9±0.2●	7.0±0.9●	3.0±0.4●	3.0±0.2●	7.6±0.8●	7.8±0.3●	1.1±0.1
horse-colic	30.8±0.6●	2.8±0.3●	8.5±0.8●	3.7±0.7●	3.0±0.1●	14.7±0.5●	12.4±0.6●	1.1±0.1
ionosphere	26.9±0.5●	6.9±0.3●	7.9±0.6●	6.6±0.4●	3.0±0.0●	4.0±0.1●	6.6±0.4●	1.3±0.1
kr-vs-kp	31.5±0.5●	16.0±0.6●	11.7±1.3●	17.0±0.8●	5.0±0.0○	21.0±0.7●	21.5±0.3●	8.9±0.9
labor	5.1±0.2●	2.7±0.4●	4.6±0.1●	2.5±0.2●	2.8±0.0●	1.2±0.2	2.9±0.1●	1.1±0.1
mushroom	7.8±0.2●	5.3±0.1●	8.0±0.0●	6.3±0.2●	2.9±0.0○	3.0±0.0	4.0±0.0●	3.1±0.1
pima-indians	113.6±1.5●	3.8±0.4●	15.4±1.0●	6.0±0.8●	2.2±0.2●	43.5±1.0●	31.6±1.5●	1.1±0.1
sick	115.0±1.9●	3.3±0.4●	.8±0.6	3.6±1.0●	2.0±0.0●	73.3±1.5●	18.1±0.6●	1.1±0.1
sonar	23.7±0.4●	5.4±0.4●	8.7±0.8●	3.9±1.2●	4.6±0.2●	5.9±0.4●	9.0±0.2●	1.4±0.1
vote	13.9±0.4●	2.5±0.4●	5.1±0.4●	3.4±0.4●	2.0±0.0●	5.7±0.4●	6.9±0.2●	1.5±0.1
average	40.3±0.7	4.4±0.3	9.0±0.7	5.3±0.7	3.0±0.1	18.8±0.7	13.5±0.5	1.8±0.2

Table 4.3: Results: average number of rules learned by the algorithms and standard deviations.

Data set	CN2 (Orig)	RIPPER w Opt	C5.0	RIPPER wo Opt	CN2 WRACC	PART	FOIL	SL^2
australian	169.2±2.1●	9.0±1.7●	42.9±3.6●	18.4±2.6●	5.2±0.1●	137.4±7.9 ●	322.7±13.7●	2.1±0.7
breast-cancer	136.2±2.8●	4.0±1.0○	18.8±1.7●	13.5±3.1●	14.9±0.9●	173.7±5.8 ●	79.7±5.9 ●	8.5±2.2
breast-w	61.4±1.0●	20.7±2.8	14.1±0.6○	24.4±1.8●	14.5±0.3○	26.0±1.8 ●	76.6±1.8 ●	18.0±2.9
german	439.1±4.9●	17.0±2.0●	125.7±5.3●	27.6±3.7●	13.2±0.7●	454.9±16.7●	539.6±34.8●	8.6±1.2
glass(G2)	61.4±1.0●	12.0±0.4	21.3±2.4●	14.0±2.5	22.5±0.9●	28.4±1.9 ●	80.4±3.7 ●	15.5±3.7
heart-c	105.4±1.0●	11.3±0.9●	30.6±2.5●	18.1±2.2●	13.1±1.7●	64.3±4.7 ●	117.1±7.9 ●	6.2±2.2
heart-h	90.1±1.6●	6.6±0.7●	22.5±3.2●	12.3±1.9●	14.6±1.7●	85.7±5.7 ●	102.5±4.8 ●	4.3±0.7
heart-statlog	102.5±2.2●	11.1±0.8●	24.7±1.8●	16.9±2.6●	14.0±1.1●	54.2±3.6 ●	106.7±3.7 ●	3.6±2.6
hepatitis	46.0±0.9●	5.5±0.8	16.2±3.0●	9.3±1.3●	16.5±0.8●	33.6±2.2 ●	38.4±1.4 ●	6.3±1.4
horse-colic	100.6±1.7●	9.0±1.0●	19.7±2.8●	12.9±2.7●	11.6±0.4●	79.9±3.6 ●	116.4±4.1 ●	2.0±0.4
ionosphere	63.6±1.0●	17.8±1.0●	15.2±2.4●	16.9±1.1●	15.6±0.3●	24.6±0.9 ●	74.5±5.4 ●	9.3±2.5
kr-vs-kp	109.7±1.9●	63.9±2.6●	45.0±3.9	74.4±4.3●	12.1±0.0○	67.3±2.1 ●	85.8±1.7 ●	44.2±4.5
labor	12.5±0.3●	6.0±0.7●	7.0±0.2●	5.9±0.7●	9.9±0.1●	3.8±0.2 ○	9.2±0.3 ●	4.5±0.5
mushroom	19.2±0.3●	16.1±0.4	16.7±0.1	18.2±0.5●	10.3±0.1○	15.0±0.0 ○	16.8±0.2	16.1±0.9
pima-indians	394.9±3.6●	10.3±1.8●	98.9±6.8●	22.4±3.4●	18.8±1.2●	264.4±8.8 ●	648.4±31.6●	6.1±1.6
sick	402.8±5.9●	19.4±2.8●	1.3±1.0	22.5±6.1●	7.4±0.2●	640.8±17.5●	264.3±10.6●	1.6±0.5
sonar	59.2±1.0●	11.6±1.2○	24.9±2.2●	8.7±2.9○	28.5±0.6●	28.4±1.2 ●	145.4±3.2 ●	19.1±2.8
vote	42.6±1.1●	7.9±1.6●	10.5±1.0●	11.3±1.5●	2.6±0.1○	12.9±1.6 ●	33.3±1.1 ●	3.5±0.7
average	134.2±1.9	14.4±1.3	30.9±2.5	19.3±2.5	13.6±0.6	122.0±4.8	158.8±7.6	10.0±1.8

Table 4.4: Results: average number of literals in the rules induced by the algorithms and standard deviations.

4.4 Ensemble Based Capacity Control

So far, we have applied the structural risk minimization principle for capacity control. This is a natural choice as it allows to incorporate a bias towards simple rule sets in an elegant way. The experiments in the previous section have shown that such a bias compares favorably with existing rule learning systems. However, most traditional separate-and-conquer rule learning systems feature similar biases. It is very well possible that other biases work better on average for the problems considered here. Indeed, many empirical studies have shown that ensemble methods such as bagging succeed remarkably well in improving predictive accuracy with unstable predictors such as decision trees and rule learners. This can also be seen in the experiments in Table 4.5 in Section 4.4.4, where bagged PART outperforms the original PART significantly on 29 of the 35 data sets and bagged RIPPER outperforms the original RIPPER in 27 cases.

This phenomenon can be explained in various ways. A traditional explanation is the observation that bagging reduces the estimation part of the prediction error by averaging over many classifiers. This explanation indicates that bagging essentially performs capacity control, but does so in a superior way when compared to the heuristics and concepts in traditional rule learners. In fact, the results in the preceding section indicate that a comparably small number of rules and literals is sufficient in order to achieve competitive predictive accuracy. This means, that the capacity control facilities in those rule learners appear to not work very well. Since most of the rule sets in Table 4.3 contain only one or two rules, even the structural risk minimization approach of SL^2 might be too coarse-grained, because it regularizes essentially with regard to the number of rules.

Another explanation is given by Grandvalet (2004), who states that bagging equalizes the influence of training instances on the classifier. In this way, bagging reduces the influence of outliers and thus improves robustness and predictive performance on noisy data. This is consistent with an observation that was made by Holte *et al.* (1989) quite early in the research on rule learning: Most errors in separate-and-conquer rule learning are made by small disjuncts, that is, rules, which are induced late and are therefore based on only few training examples. As these instances can not be explained by the existing rules, they are often outliers. Consequently, small disjuncts often predict badly, and reducing the influence of the outliers, they are based on, countervails this effect.

Finally, aiming for a short DNF representation is sometimes diametrical to robustness considerations. Consider the case, where the four Boolean attributes a_2, a_5, a_6 and a_9 have a positive influence on the concept to be learned, so that an instance is certain to feature a positive class label, whenever all four of them are true. Furthermore, even when only three of four attributes are true, the influence is still strong enough to make an instance very likely to have the positive class. Representing such a robust concept as a DNF formula requires at least four terms: $(a_2 \wedge a_5 \wedge a_6) \vee (a_2 \wedge a_5 \wedge a_9) \vee (a_2 \wedge a_6 \wedge a_9) \vee (a_5 \wedge a_6 \wedge a_9)$. On the other hand, the concept can be elegantly expressed by a voting procedure such as $a_2 + a_5 + a_6 + a_9 \geq 3$, where the influence of individual components is added up. So, in some cases, the bias towards short DNF representations might indeed be harmful, whereas the voting procedures applied in typical ensemble methods could compensate for this representational shortcoming. In those cases, ensembles improve robustness by changing the bias rather than the variance part of the error.

In the following we present an ensemble based approach to performing capacity control on DNF rule sets. The general design is motivated by Breiman's Random Forest (Breiman, 2001). Instead of aiming at a single simple and comprehensible predictive rule set (which, as explained above, might not even exist), the system generates an ensemble of diverse rule sets. The loss of comprehensibility due to the ensemble can be compensated by extracting comprehensible consensus rules and statistics over frequently used features and feature combinations along the lines of the work by Pfahringer *et al.* (2001) on ADTrees. As an interesting side-effect, the linear combinations of ensembles are generally more amenable to analytical investigations than the combinatorial properties of DNF representations. For instance, the prediction error of ensembles can be easily bounded using the PAC-Bayesian theorem (Theorem 3.6.1 in Section 3.6). Doing so for SL^2 is a non-trivial problem. The natural tool to do so would be VC theory, but the VC-dimension of k-term DNF is not known exactly.

4.4.1 The Learning Setting

To give a precise description of the learning algorithm and the underlying theory, let us recall a few definitions and state basic assumptions. As usual, training and test data are given as sets of labeled *instances*, taken from an *instance space* $\mathcal{X}$. The instance space is structured as the cartesian product of m domains A_i, so that each instance x is represented by an m-tuple $(x_1, x_2, \ldots, x_m)$, where each x_i denotes a value taken from A_i. We require that all domains are finite, so continuous attributes need to be discretized in order to fit into our framework. For example, if $A_1 = \{a, b, c\}$, $A_2 = \{0, 1\}$, and $A_3 = \{red, blue\}$, then $\mathcal{X} = A_1 \times A_2 \times A_3$ and $(a, 1, red)$ and $(c, 0, blue)$ are two out of the 12 instances in $\mathcal{X}$. Since the system is designed to also work in multi-class settings, $\mathcal{Y}$ denotes a finite, but otherwise arbitrary set of *classes*. As usual, we assume a fixed, but unknown distribution D on the pairs $(x, y) \in \mathcal{X} \times \mathcal{Y}$. A training set (X, Y) is generated by drawing n *labeled instances* (x, y) independently according to D. The learning algorithm is given the sample (X, Y). Upon termination it outputs a classifier c that maps instances to classes. The main goal, of course, is to come

Algorithm 7 The rule learning algorithm. S is the training set, k_{max} the maximal number of rules per rule set, o the number of rule sets per ensemble and rule set size k, and ν is the noise probability.

```
procedure LearnEnsemble($S$, $k_{max}$, $o$, $\nu$)
    for $cl = 1$ to $p$ do
        $Q_{cl} \leftarrow$ empty ensemble
        for $k = 1$ to $k_{max}$ do
            for $s = 1$ to $o$ do
                $rs \leftarrow$ SLSearch($k$)
                $w \leftarrow \exp(-\nu\, \hat{l}(rs, S))$
                $Q_{cl} \leftarrow Q_{cl} \cup \{(rs, w)\}$
            end for
        end for
        Normalize the weights in $Q_{cl}$ to sum up to one
    end for
    return $\{Q_1, \ldots, Q_p\}$
end procedure
```

up with an algorithm that generates a classifier c with provably low error $\mathbf{Pr}_{(x,y)\sim D}[c(x) \neq y]$.

Since we are working in the field of rule learning, we use a particular representation for the classifiers to be learned. First of all, a *literal* is of the form $(a_i = v_i)$ or $\neg(a_i = v_i)$ for any $1 \leq i \leq m$, where a_i identifies the attribute i and v_i is a value taken from the corresponding domain A_i. For example, for the instance space defined above, $(a_2 = 0)$ and $\neg(a_3 = red)$ are valid literals. A conjunction of literals is a *rule*. A rule is said to *cover* an instance, if the conditions imposed by all literals in the rule are met by the instance. For example, the rule $(a_1 = b) \wedge \neg(a_3 = red)$ covers the instance $(b, 0, blue)$, but does not cover the instance $(b, 0, red)$. Not surprisingly, a *rule set* is a set of rules. A rule set covers an instance, if any rule in the rule set covers it. If we have p distinct class labels, a *labeled ruleset* $r_i := (r, y_i)$ is a ruleset together with a class label. We can use a labeled rule set r_i to predict the class $r_i(x)$ given an instance x:

$$r_i(x) := \begin{cases} +1 & \text{if } x \text{ is covered by } r_i \\ -1 & \text{otherwise} \end{cases}$$

The class of all possible y_i-labeled rule sets is denoted by R_i.

4.4.2 Learning Ensembles of Rule Sets

With these notations, we can now describe the actual ensemble based rule learning approach. Algorithm 7 learns one ensemble per class. The rule sets in the ensemble are conveniently sampled using the SLSearch algorithm, outlined in Section 4.2.2. Since we do not know the optimal number of rules per rule set k in advance, we sample a constant number of rule sets for each k up to a user-specified limit k_{max}.

Of course, the sampled rule sets differ in their ability to explain the training set. It makes sense to keep, along with each rule set, a probability $Q(r_i)$ depending on the R_i's error on the training set. If the user has information about the noise behavior of the underlying target distribution, she can use this information to generate a matching noise model and generate Q accordingly. For example, if one is learning in a noise-free setting, it makes sense to assign a probability of 0 to inconsistent rule sets and $1/h$ to the h consistent rule sets that were found. If, on the other hand, the noise level is higher for some parts of the instance space $\mathcal{X}$, one would want to base $Q(r_i)$ on the misclassified instances at hand: if a rule set misclassifies only "noisy" instances, a higher probability is assigned than for a rule set that misclassifies relatively stable instances. As a conservative default strategy we use a Q that assigns a probability that is exponentially decreasing with its empirical error on (X, Y) to each r_i:

$$Q(r_i) := \frac{\exp(-\nu\,\hat{l}(r_i, X, Y))}{\Sigma_{Q(r_i)\neq 0} \exp(-\nu\,\hat{l}(r_i, X, Y))},$$

where $\hat{l}(r_i, X, Y)$ denotes the empirical error of r_i on the sample (X, Y) (see also Section 4.4.3), and ν specifies the rate of decay. This corresponds to a white noise model, where the error probability is constant $\alpha := \exp(-\nu/n)$ across all instances. Similar models have been studied, for instance, by Freund *et al.* (2004) and Littlestone and Warmuth (1994).

The learning algorithm should also be able to handle multi-class problems directly. Up to now, we only consider the case of two-class problems. Dealing with multi-class problems has traditionally been problematic for rule learning systems. For the sake of simplicity, we employ a simple one-vs-all scheme here: we learn p ensembles Q_i, where ensemble Q_i distinguishes between class label y_i and $\mathcal{Y}\setminus\{y_i\}$. As the scores can be seen as a measure of how certain an ensemble is about its prediction (see Section 4.4.3), it makes sense to choose that class y_i with the largest score $c(Q_i, x)$.

4.4.3 Bounds for Rule Learning

In the following, we will present a theoretical analysis of the above algorithm. The goal is to bound its expected prediction error. In the first part, we show how McAllester's PAC-Bayesian theorem (see also Section 3.6) can be used to bound the error in ensemble rule learning. In the second part, we prove that the PAC-Bayesian bound can be further improved by allowing the learner to abstain from uncertain predictions. Together, these results are more generally applicable, but they serve well the purpose of bridging the gap between theory and practice for rule learning.

The PAC-Bayesian Bound for Rule Learning

Two-class problems are generally easier to handle than multi-class problems. We therefore use *characteristic functions* χ_i to split a p-class problem into p 2-class problems: let $\mathcal{Y}$ be a set of p class labels. Just like $r_i(x)$, $\chi_i(y)$ is defined to be +1, if $y = y_i$, and -1 otherwise. To estimate the rule set's predictive behavior we are especially interested in the cases where the class y_i assigned by the rule set disagrees with the "true" class y of an instance x. To measure the rate of misclassification, we generalize the *zero-one loss function* introduced in Section 3.2 towards the multi-class setting: let (x, y) be a labeled instance drawn according to D, and r_i a labeled ruleset. Then the *loss* $l(r_i, x, y) := \mathbf{I}[r_i(x) \neq \chi_i(y)]$ is 0, if the prediction agrees with the "true" class y, and 1 otherwise. Intuitively, l is a 0-1 loss for measuring whether the prediction of a rule set agrees with an observation drawn from D, when we are only distinguishing between class label y_i and the remaining class labels.

The *expected loss* $l(r_i) := \mathbf{E}_{(x,y)\sim D}[l(r_i, x, y)]$ indicates, how often the rule set r_i disagrees with the "true class" on average. Given a sample (X, Y) of size n, the *empirical loss* $\hat{l}(r_i, X, Y) := \frac{1}{n}\Sigma_{(x,y)\in(X,Y)} l(r_i, x, y)$ yields the fraction of instances in (X, Y) that are misclassified by r_i. It is easy to see that the PAC-Bayesian theorem (Theorem 3.6.1 in Section 3.6) can be slightly adapted to deal with (prior and posterior) distributions on the space R_i of y_i-labeled rule sets. If Q is an arbitrary probability measure on R_i, we can extend the notion of a loss to distributions instead of single rule sets: $\hat{l}(Q, X, Y) := \mathbf{E}_{r_i\sim Q}[\hat{l}(r_i, X, Y)]$ and $l(Q) := \mathbf{E}_{r_i\sim Q}[l(r_i)]$. Recall that the PAC-Bayesian theorem uses the well known Kullback-Leibler divergence, denoted by $D(Q\|P) := \Sigma_{r\in R}(Q(r)\ln\frac{Q(r)}{P(r)})$, to compare prior and posterior distributions. As a notational shortcut we write $\forall^\delta(X, Y) \quad \Phi(X, Y)$ instead of $\mathbf{Pr}_{(X,Y)\sim D^n}[\Phi(X, Y)] \leq 1-\delta$ to express that Φ holds for all but a fraction δ of the cases. With this we can reformulate Theorem 3.6.1:

Theorem 4.4.1 (PAC-Bayesian, (McAllester, 1999))**.** *Let $y_i \in \mathcal{Y}$ be a class label, let P be a distribution over the space of y_i-labeled rule sets R_i, let $\delta > 0$, and define:*

$$B(Q, P, n, \delta) := \hat{l}(Q, X, Y) + \sqrt{\frac{D(Q\|P) + \ln\frac{1}{\delta} + \ln n + 2}{2n-1}}$$

Then, where Q ranges over all distributions on R_i:

$$\forall^\delta(X, Y) \quad \forall Q \quad l(Q) \leq B(Q, P, n, \delta)$$

This theorem can be used to upper-bound the expected error of an ensemble classifier in the following way: the user provides a prior distribution P_i on the rule set space R_i. If she has some information about which rule sets are most likely to resemble the “true” target distribution D, she can use this information by selecting a P_i that assigns high probabilities to those rule sets. If she does not have any such information, she can simply select an uninformative, flat prior. In the next step she draws a sample (X, Y) of size n from D. It is now the task of the learning algorithm to select a distribution Q_i on R_i for each class label y_i. The algorithm aims at finding Q_i that minimize the right hand side of the inequality and thus provides a tight upper bound on the expected error of rule sets drawn according to Q_i.

Using those Q_i and the PAC-Bayesian theorem one can construct a voting ensemble of weighted rule sets, and bound its generalization error. Let $\bar{Q} := (Q_i, \ldots, Q_p)$. Then:

$$\begin{aligned} c(Q_i, x) &:= \mathop{\mathbf{E}}_{r_i \sim Q_i}[r_i(x)] \\ c_V(\bar{Q}, x) &:= \operatorname*{argmax}_{y_i \in \mathcal{Y}} c(Q_i, x) \end{aligned}$$

$c(Q, x)$ is the *score* of Q on x and $c_V(\bar{Q}, x)$ denotes the *voting classifier* for $\bar{Q}$. The expected error of the voting classifier is thus $l_V(\bar{Q}) := \mathbf{E}_{(x,y)\sim D}\, \mathbf{I}[c_V(\bar{Q}, x) \neq y]$. The following theorem bounds the expected error of the voting classifier for two-class problems, that is, $\bar{Q} = (Q_1, Q_2)$. Bounds for multi-class problems can be derived in a similar fashion.

Theorem 4.4.2. *For $i \in \{1, 2\}$ let $y_i \in \mathcal{Y}$ be class labels, P_i be distributions over the spaces of y_i-labeled rule sets R_i, let $\delta > 0$, and $\bar{Q} = (Q_1, Q_2)$ be as above. Then:*

$$\forall^\delta (X, Y) \quad \forall Q \quad l_V(\bar{Q}) \leq B(Q_1, P_1, n, \delta) + B(Q_2, P_2, n, \delta)$$

Proof. First, observe that for $\bar{Q} = (Q_1, Q_2)$:

$$l_V(\bar{Q}) = \mathop{\mathbf{Pr}}_{D}[\chi_1(y) c(Q_1, x) + \chi_2(y) c(Q_2, x) \leq 0] \tag{4.7}$$

Additionally,

$$\begin{aligned} 1 - 2l(Q_i) &= 1 - 2 \mathop{\mathbf{E}}_{D}\big[\mathop{\mathbf{E}}_{Q_i}[\mathbf{I}[r_i(x) \neq \chi_i(y)]]\big] \\ &= 1 - 2 \mathop{\mathbf{E}}_{D}\big[\mathop{\mathbf{E}}_{Q_i}[\frac{1}{4}(r_i(x) - \chi_i(y))^2]\big] && (4.8) \\ &= 1 - \frac{1}{2} \mathop{\mathbf{E}}_{D}\big[\mathop{\mathbf{E}}_{Q_i}[r_i(x)^2 - 2r_i(x)\chi_i(y) + \chi_i(y)^2]\big] \\ &= 1 - \frac{1}{2}\Big(1 - 2 \mathop{\mathbf{E}}_{D}\big[\mathop{\mathbf{E}}_{Q_i}[r_i(x)\chi_i(y)]\big] + 1\Big) && (4.9) \\ &= \mathop{\mathbf{E}}_{D}[\chi_i(y)\, c(Q_i, x)] \end{aligned}$$

(4.8) uses the fact that $\mathbf{I}(a \neq b) = \frac{1}{4}(a-b)^2$ for $a, b \in \{-1, +1\}$, (4.9) uses $a^2 = 1$ for $a \in \{-1, +1\}$. It follows from Theorem 4.4.1:

$$\begin{aligned} \forall^\delta\, (X, Y) \quad \underset{D}{\mathbf{E}}[\chi_i(y)c(Q_i, x)] &= 1 - 2l(Q_i) \\ &\geq 1 - 2B(Q_i, P_i, n, \delta) \end{aligned} \tag{4.10}$$

Now, let $C := 2 - \chi_1(y)c(Q_1, x) - \chi_2(y)c(Q_2, x)$ be a random variable. Since $C \geq 0$ for all x, y, Q_1, Q_2, we can apply Markov's inequality:

$$\forall \varepsilon > 0 : \underset{D}{\mathbf{Pr}}\left[C \geq \varepsilon\, \underset{D}{\mathbf{E}}[C]\right] \leq \frac{1}{\varepsilon}$$

By definition of C,

$$\begin{aligned} \forall \varepsilon > 0 : \underset{D}{\mathbf{Pr}} \Big[&\chi_1(y)c(Q_1, x) + \chi_2(y)c(Q_2, x) \leq \\ &2 - 2\varepsilon + \varepsilon\, \mathbf{E}[\chi_1(y)c(Q_1, x)] + \varepsilon\, \mathbf{E}[\chi_2(y)c(Q_2, x)] \Big] \leq \frac{1}{\varepsilon} \end{aligned}$$

and because of (4.10)

$$\begin{aligned} \forall \varepsilon > 0 \quad \forall^\delta (X, Y) : \underset{D}{\mathbf{Pr}} \Big[&\chi_1(y)c(Q_1, x) + \chi_2(y)c(Q_2, x) \leq \\ &2 - 2\varepsilon(B(Q_1, P_1, n, \delta) + B(Q_2, P_2, n, \delta)) \Big] \leq \frac{1}{\varepsilon} \end{aligned} \tag{4.11}$$

The theorem follows from (4.7) by setting

$$\varepsilon = \frac{1}{B(Q_1, P_1, n, \delta) + B(Q_2, P_2, n, \delta)}.$$

□

In order to keep this bound as tight as possible, one would like to find Q_i that have low empirical error on the sample (X, Y) and that differ from the P_i only to a small degree. While this may be possible theoretically, such a "PAC-Bayesian-optimal" algorithm would require calculating the empirical error of all possible concepts in the underlying concept class. Of course, this is not practical for our purposes, because the space of rule sets is way too large to be evaluated exhaustively.

We try to keep the computational costs within a reasonable range by taking two measures: first, we limit the maximum size of the rule sets to be considered. This is necessary anyway, because rule sets of arbitrary size can represent all possible dichotomies. If a flat prior is used and we choose Q only depending on the training set, this is effectively bias-free learning, which is known to be equivalent to rote learning. Thus, we restrict our concept space to the set of all rule sets with at most k_{max} rules per rule set. This bias towards short hypothesis is motivated by the principle of William of Ockham and – in one form or the other – included in virtually any existing rule learning algorithm. Second, instead of considering all possible rule sets for a Q_i, we sample a small number of rule sets from R_i, and set the probability measure Q_i to zero for all rule sets outside the sample. In this way, we have to deal with only a rather small number of concepts; the calculation of $c(Q_i, x)$ involves only the summation over the few rule sets with non-zero $Q_i(r_i)$ and the task of predicting and estimating the expected error becomes feasible.

Unfortunately, it is a bad idea to sample uniformly or according to P_i from R_i, because the bound depends on the empirical error $\hat{l}$ on the training set. Thus, if the sampled rule sets have a high error on the training set, the bound will be loose. It is therefore essential that we select rule sets with low empirical error. As explained in the preceding sections, we employ the SLSearch algorithm that finds DNFs with low empirical error in a randomized fashion to achieve this goal.

Improving the Bound Through Abstaining

Most rule learning systems are designed to assign a class to any instance that was input for classification. In practice, though, it is often the case that some instances clearly belong to one class, while other instances are just in between two classes or particularly susceptible to noise. A classifier that abstains from classification for instances of the latter kind might feature a much higher predictive accuracy, because it avoids errors on uncertain predictions. Sometimes, abstaining can give important hints to the user, e.g. about the existence of a previously unknown class label.

These considerations lead to a different approach for getting tighter bounds: allowing the classifier to abstain from a classification for uncertain instances. In our case we can assess the deviation of the votes in an ensemble as a measure of how certain the corresponding prediction is. If all rule sets in an ensemble vote for the same class label, the classification is quite certain. If, on the other hand, the weight of the rule sets that vote for y_i differs only by a small margin from the weight of the rule sets that vote for a different y_j, the classification can be regarded as uncertain. Thus, it might make sense to abstain from a classification, if the absolute value of the margin is lower than a certain threshold $\theta > 0$. For the two-class setting, the following definition gives the *abstaining voting classifier* c_V^θ. Again, the concept can be easily extended to the multi-class setting.

$$c_V^\theta(\bar{Q}, x) := \begin{cases} y_1 & \text{if } c(Q_1, x) - c(Q_2, x) \geq \theta \\ 0 & \text{if } -\theta < c(Q_1, x) - c(Q_2, x) < \theta \\ y_2 & \text{if } c(Q_1, x) - c(Q_2, x) \leq -\theta \end{cases}$$

The expected error of this abstaining classifier is

$$\begin{aligned} l^{\theta}_{V}(\bar{Q}) &:= \mathop{\mathbf{E}}_{(x,y)\sim D}[\mathrm{I}(c^{\theta}_{V}(\bar{Q},x) \neq 0 \wedge c^{\theta}_{V}(\bar{Q},x) \neq y)] \\ &= \mathop{\mathbf{Pr}}_{(x,y)\sim D}[\chi_1(y)c(Q_1,x) + \chi_2(y)c(Q_2,x) \leq -\theta]. \end{aligned} \tag{4.12}$$

The following adaption of Theorem 4.4.2 improves the PAC-Bayesian bound for an ensemble $\bar{Q}$, if the abstaining voting classifier is used instead of the voting classifier.

Theorem 4.4.3. *Let the y_1, y_2, P_1, P_2, Q_1, Q_2, and δ be as above, let $\theta > 0$. Then:*

$$\forall^{\delta} S \quad \forall \bar{Q} \quad l^{\theta}_{V}(\bar{Q}) \leq \frac{B(Q_1, P_1, n, \delta) + B(Q_2, P_2, n, \delta)}{1 + \frac{1}{2}\theta}$$

Proof. The result follows from (4.12) and by setting

$$\varepsilon = \frac{1 + \frac{1}{2}\theta}{B(Q_1, P_1, n, \delta) + B(Q_2, P_2, n, \delta)}$$

in (4.11). □

4.4.4 Experiments

In this section we describe an empirical evaluation of the ensemble based rule learning algorithm. The goal of the first experiment is to investigate how the presented algorithm compares to modern rule learning algorithms. To get results on learning problems with varying characteristics, we select 35 data sets from the UCI repository (Asuncion and Newman, 2007). Since the presented rule learning algorithm works only on nominal attributes, we discretize continuous attributes using a frequency-based discretization with ten intervals. If a data set contains unknown values, we simply add a new value "unknown" to the corresponding domains, so that unknown values are treated just like any other value.

To estimate the predictive accuracy of the algorithms we averaged over ten runs of tenfold cross-validation. The presented algorithm was set up to build rule sets with up to eight rules per rule set, and we set the o parameter to twenty rule sets per level. To calculate the $Q(r)$ probabilities, we chose the "white noise" model described in Section 4.4.2 with the noise parameter α set to 0.9. The SLSearch algorithm was set up to search for 5000 iterations, with p_{g1}, p_{g2}, p_s set to the default values of 0.1, 0.2, and 0.1, respectively. We compare the results for the presented algorithm with the results of a support vector machine with RBF kernel and two state-of-the art rule learning systems. PART (Frank and Witten, 1998) is a separate-and-conquer-based rule learning algorithm, that avoids over pruning by obtaining

Data Set	SVM	PART	JRIP	Bagged PART	Bagged JRIP	Rule Ens.
anneal	92.9	97.2	98.0 ± 0.4	98.2 ± 0.2	98.4 ± 0.2	98.3 ± 0.1
audiology	43.8	78.8	72.8 ± 1.6	82.6 ± 0.6	77.6 ± 1.3	81.5 ± 0.9
autos	62.0	70.7	74.2 ± 2.1	82.2 ± 0.9	82.3 ± 1.3	83.4 ± 0.4
balance-scale	90.7	77.3	71.7 ± 0.9	85.1 ± 0.6	80.9 ± 1.2	83.6 ± 0.5
breast-cancer	70.3	71.0	71.8 ± 1.4	72.8 ± 1.7	74.2 ± 0.9	73.8 ± 0.5
breast-w	97.3	94.3	94.1 ± 0.5	95.3 ± 0.2	94.8 ± 0.3	96.6 ± 0.2
bupa	58.0	58.3	62.1 ± 0.9	60.9 ± 1.8	59.7 ± 1.2	69.3 ± 0.9
colic	85.9	83.4	84.3 ± 0.4	85.1 ± 0.5	85.2 ± 0.4	82.9 ± 0.5
credit-a	86.2	85.9	85.4 ± 0.4	87.0 ± 0.5	85.7 ± 0.5	86.1 ± 0.6
credit-g	71.3	70.7	69.6 ± 0.7	74.7 ± 0.6	73.1 ± 0.6	73.6 ± 0.5
diabetes	72.4	73.4	71.5 ± 0.4	73.7 ± 0.7	72.5 ± 0.6	73.9 ± 0.9
glass	54.2	52.3	64.2 ± 2.5	65.2 ± 2.0	70.2 ± 1.8	73.2 ± 1.1
haberman	73.5	73.5	71.4 ± 0.7	70.9 ± 0.7	72.1 ± 0.5	70.7 ± 0.4
heart-c	84.8	80.9	78.7 ± 1.9	83.0 ± 1.3	81.1 ± 0.9	78.5 ± 0.7
heart-h	83.3	78.9	79.4 ± 1.5	81.3 ± 1.2	81.0 ± 0.6	79.8 ± 0.6
heart-statlog	84.1	78.5	77.6 ± 1.3	81.2 ± 0.8	81.3 ± 0.8	77.1 ± 1.2
hepatitis	79.4	80.0	78.2 ± 1.4	81.9 ± 0.9	79.4 ± 0.7	83.9 ± 0.4
hypothyroid	96.8	97.9	97.9 ± 0.1	98.2 ± 0.1	98.1 ± 0.1	98.1 ± 0.1
ionosphere	90.3	88.0	90.8 ± 0.9	90.4 ± 0.4	91.9 ± 0.5	91.8 ± 0.3
iris	90.7	92.0	88.1 ± 1.5	92.6 ± 0.6	90.9 ± 1.3	91.1 ± 0.3
kr-vs-kp	91.4	99.1	99.2 ± 0.0	99.4 ± 0.1	99.4 ± 0.1	97.8 ± 0.1
labor	70.2	87.7	76.7 ± 2.8	84.4 ± 2.0	83.3 ± 1.2	93.3 ± 1.3
lymph	80.4	79.1	78.6 ± 1.9	85.3 ± 1.6	80.1 ± 1.2	84.9 ± 0.7
mushroom	99.9	100.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0
primary-tumor	24.8	40.7	38.7 ± 1.0	45.3 ± 1.1	42.0 ± 0.8	43.6 ± 0.6
segment	94.4	94.2	91.8 ± 1.0	95.7 ± 0.2	96.4 ± 0.2	97.2 ± 0.1
sick	97.6	98.1	97.6 ± 0.1	98.3 ± 0.1	97.7 ± 0.0	98.1 ± 0.0
sonar	76.9	62.0	64.2 ± 3.3	72.1 ± 2.1	70.4 ± 1.2	76.2 ± 1.4
soybean	88.4	91.8	92.2 ± 0.5	93.6 ± 0.3	93.5 ± 0.3	93.2 ± 0.2
splice	96.1	92.5	94.3 ± 0.4	94.9 ± 0.1	95.8 ± 0.1	93.8 ± 0.3
tic-tac-toe	76.2	94.5	97.5 ± 0.4	99.7 ± 0.2	98.2 ± 0.1	99.2 ± 0.1
vehicle	68.4	67.0	58.8 ± 1.0	69.9 ± 1.3	70.0 ± 0.6	71.8 ± 0.7
vote	95.2	95.9	95.3 ± 0.3	96.0 ± 0.3	95.8 ± 0.2	95.9 ± 0.1
waveform-5000	84.6	73.7	72.6 ± 0.7	80.3 ± 0.3	80.7 ± 0.3	82.0 ± 0.3
zoo	73.3	92.1	87.7 ± 0.8	92.7 ± 0.9	90.7 ± 0.8	95.0 ± 0.9

Table 4.5: Results: percentage of correct classifications, together with standard deviation.

rules from partial decision trees. JRIP (an implemention of Cohen's RIPPER (Cohen, 1995) in the WEKA workbench (Witten and Frank, 1999)) combines separate-and-conquer with incremental reduced error pruning and an iterated post-processing optimization step. To include ensemble-based approaches, we estimated the predictive accuracy of twentyfold-bagged versions of the two algorithms. The results are given in Table 4.5. Table 4.6

	SVM	PART	JRIP	Bagged PART	Bagged JRIP	Rule Ens.
SVM		16	16	10	11	9
PART	19		17	2	5	7
JRIP	14	9		0	1	4
Bagged PART	23	29	28		15	14
Bagged JRIP	19	22	26	6		7
Rule Ensemble	23	24	25	12	17	

Table 4.6: Results: each number identifies the number of data sets, on which the method in the row significantly outperforms the method in the column.

shows how different methods compare to each other. Each entry indicates the number of data sets for which the method associated with its row is significantly more accurate than the method associated with its column according to a paired two-sided t-test on a 1% significance level over the runs. As can be seen, the presented algorithm performs favorably. It clearly outperforms SVM, PART, JRIP, and Bagged JRIP, and is slightly worse than Bagged PART. A comparison of the rule ensemble's predictive accuracy with the one of SL^2 (described in Section 4.3.1) on eighteen data sets with binary target attributes can be found in Table 5.1 in Section 5.5. There, the rule ensemble outperforms SL^2 in fifteen of the eighteen cases, a clear indication that capacity control based on ensembles works better than the structural risk minimization approach taken by SL^2.

For the second experiment, the main goal is to investigate the gap between the PAC-Bayesian bound and the true error as estimated by tenfold cross-validation and to compare our results with related approaches. Multi-class problems require the application of the union bound on the PAC-Bayesian bounds for the p ensembles, so the resulting bound is rather loose. We therefore focus on two-class problems. We apply the presented algorithm with the same parameters as above and a flat prior P to a selection of two-class problems taken from the UCI repository (Asuncion and Newman, 2007). To the best of our knowledge, there are no comparable results on theoretical bounds for rule learning systems in the literature. The closest approaches in the literature are SLIPPER (Cohen and Singer, 1999), LRI (Weiss and Indurkhya, 2000), and the set covering machine (Marchand and Shawe-Taylor, 2001; Sokolova *et al.*, 2003; Marchand *et al.*, 2003). SLIPPER and LRI are rule learning algorithms based on ensembles of individual rules instead of rule sets. Since they employ voting schemes, they are amenable to theoretical analysis and also would be able to abstain from predictions. However, standard approaches to bounding the error applied to SLIPPER and LRI give rather loose bounds. Like a rule learning system, the set

	Rule Ens.		Consistent		Simple	
Data Set	Bound	CV	Bound	CV	Bound	CV
breast-w	30.6	3.4	43.6	4.8	**15.8**	**2.3**
bupa	**54.3**	**30.7**	100.0	38.0	84.6	30.7
credit-a	**57.7**	**13.9**	100.0	39.1	87.6	29.9
diabetes	**64.1**	**26.1**	99.9	29.9	78.6	26.8
glassg2	76.8	26.8	91.4	**22.1**	**63.8**	**22.1**
haberman	**68.0**	**29.3**	99.7	38.1	78.2	31.6
vote	**36.5**	**4.1**	71.2	11.5	44.2	11.5

Table 4.7: Results: the prediction error in percent as estimated by ten runs of tenfold cross validation and the size of the bound for the rule set ensemble algorithm, the consistent set covering machine, and the simple set covering machine.

covering machine uses disjunctions[4] of Boolean-valued features as concepts. However, unlike rule learners, it disjunctively joins data-dependent features such as generalized balls and half-spaces instead of conjunctions of literals. Marchard *et al.* derive a bound based on a compression scheme, that can be compared to the PAC-Bayesian bound. They report empirical and analytical results for a whole range of parameter settings. In Table 4.7 we reproduce the values for two particular settings: the "consistent" columns give the results for the unparameterized version of the data-dependent ball SCM, which induces only consistent classifiers. The "Simple SCM" is able to derive inconsistent classifiers, but the results are given for experiments that use only the best parameter settings among an exhaustive scan of many values for each data set. Those values are thus much more optimistic than our results, which are based on default parameter values that are fixed for all data sets. Nevertheless, the PAC-Bayesian bound is better in five out of seven cases, and the presented algorithm achieves a lower prediction error in five out of the seven cases.

We also performed preliminary experiments with abstaining ensembles of rule sets. To test the validity of this approach empirically, we estimate the prediction error of the abstaining voting algorithm on the Haberman data set using tenfold cross-validation. We used the same parameters as before, but varied θ between 0 and 2. Figure 4.2 shows the difference between bound and estimated error. As can be seen, the relative accuracy of the bound is optimal for values of θ near 1. Thus, the bound can in fact be improved on this data set for a certain level of abstention.

[4]The set covering machine can induce disjunctions or conjunctions. Since we are dealing with rule sets, we consider the disjunctive case only.

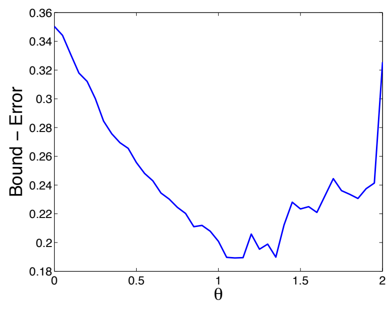

Figure 4.2: The difference between the bound and the estimated prediction error depending on the abstaining parameter θ.

4.5 Summary and Related Work

In this chapter we dealt with empirical risk minimization and capacity control for DNF rule sets. We started with empirical risk minimization in the noise-free setting. In this setting, rule sets must classify all training instances correctly. Finding a consistent rule set with at most k rules is an NP-hard problem. We described three simple enumeration schemes for solving such k-term DNF consistency problems, some of which were also considered in previous work (Rückert, 2002; Rückert *et al.*, 2002). Those enumeration schemes are slower than necessary, because they do not make use of the structure in the training data to speed up the search. Thus, we devise RandomSearch, a probabilistic version of Algorithm 4 in Rückert (2002), which prunes the search depending on the training data. A theoretical analysis similar to the one by Paturi *et al.* (2005) yields a bound on the worst case time complexity, which depends on the *block size*, a structural property of the training data.

For the noisy setting, the goal is to find a rule set that agrees with as many training instances as possible. We resort to the stochastic local search approach introduced previously (Rückert, 2002) and investigated in the phase transition framework by Rückert and Kramer (2003). The approach is conceptually similar to *WalkSAT* (Selman *et al.*, 1996) and has been successfully applied in a multi-relational setting by Paes *et al.* (2007). In Section 4.3 we describe the rule learner SL^2, which combines SLS with capacity control based on cross-validation and a postprocessing step to induce small, yet predictive rule sets. This allows for an investigation of how existing rule learners perform in terms of rule set size and number of literals as compared to an "optimal" strategy that is designed to output the smallest possible rule sets while preserving predictive accuracy. The results were surprising in that some (but not all) rule learners produced rule sets that were orders of magnitude larger than necessary to achieve comparable predictive accuracy.

Simplicity has always been an important issue in rule learning (see for instance Weiss and Indurkhya, 1993; Lavrač *et al.*, 2004) and decision tree learning (for example Bohanec and Bratko, 1994). It is also known that simple classifiers can feature high predictive accuracy. For instance, Holte (1993) showed that for some real-world data sets, decision stump classifiers can have predictive accuracies that are as high as 97% of C4's performance. However, the work reported here (also published in Rückert and De Raedt, 2007) differs in that it is based on basic statistical induction principles rather than heuristics and that SL^2 is not restricted to a fixed hypothesis class. This allows for well-supported insights into the trade-off between predictive accuracy and simplicity of rule learning systems

Finally, in Section 4.4.2 we investigate capacity control based on ensembles (also published in Rückert and Kramer, 2004b). We apply SLS to address the k-term DNF maximum agreement problem for various settings of k and combine the resulting rule sets in an ensemble, similar to the way it is done with decision trees in Random Forest (Breiman, 2001) and with rule sets in LRI (Weiss and Indurkhya, 2000). We analyze the system empirically and analytically. On the analytical side we show that the empirical error of the algorithm can be bounded theoretically by applying McAllester's PAC-Bayesian theorem (McAllester, 1999) and that the system can be seen as a straightforward approach to optimize this bound. In most cases, the ratio of the bound to the empirical error is smaller than for the set covering machine, for which one of the tightest bounds is known. We prove that the PAC-Bayesian bound can be further improved by allowing the model to abstain from making uncertain predictions and give an example of how abstention can improve predictive accuracy. Work on abstaining classifiers dates back to the 1970s (Chow, 1970) and has been investigated empirically and theoretically by Friedel (2005); Friedel *et al.* (2006) and Pietraszek (2005), among others. Recent theoretical results on abstention with ensembles can be found in the work of Freund *et al.* (2004).

On the empirical side, we evaluated the performance of the algorithm on standard UCI data sets. We found that it compares very favorably with state-of-the-art rule learning algorithms and their bagged variants. Experiments showed that the calculated bounds are reasonably close to the actual prediction errors as estimated by tenfold cross-validation. It should be noted that the bound (and with it the algorithm's predictive accuracy) can be improved in various ways. For instance, one can

- provide an informative prior instead of a flat prior,
- use a matching noise model instead of the white noise default,
- increase the ensemble size, or
- allow for abstaining.

Thus, the algorithm can be easily adopted to a particular setting in the presented framework. Moreover, it should be possible to extract comprehensible consensus rules and statistics over frequently used features and feature combinations along the lines of the work on ADTrees by Pfahringer *et al.* (2001) and RuleFit by Friedman and Popescu (2005).

Chapter 5

Weighted Rule Sets

This chapter deals with rule sets where a weight is attached to each rule. Empirical risk minimization for this kind of rule sets is a numerical optimization problem. We describe one established and two novel convex optimization criteria for empirical risk minimization on those rule sets. For capacity control, we extend the Rademacher and PAC-Bayesian bounds to work directly on the new criteria, and we derive a new inequality, which gives estimates of the structural risk depending on the size of the rule repository. Based on these tools, we implement the rule learner RUMBLE, which combines the three criteria with capacity control based on the rule repository size bound. An empirical evaluation of RUMBLE shows that the Margin Minus Variance criterion performs best and that it generates small and highly predictive rule sets.

5.1 Motivation

Though many traditional rule learning systems induce DNF rule sets or decision lists, weighted rule sets offer some advantages that justify a further analytical and empirical investigation. In the following we highlight two particular points and refer to Section 2.2.3 for a more thorough description of the three common rule set representations. The first fundamental difference to DNF rule sets or decision lists is the fact that weighted rule sets introduce the concept of a *margin* into rule learning. To classify an instance weighted rule sets add up the weights of rules that vote for the positive class and subtract the weights of rules that vote for the negative class. Whereas the sign of the resulting quantity indicates the prediction for the particular instance, the margin, that is, its absolute value can be seen as a measure of the certainty of the prediction. This is intuitively intriguing: the prediction appears to be more justified, when all rules vote for the same class as opposed to a near-draw situation where the weights of rules favoring one class are of the same size as the weights of rules for the other class. This notion can also be stated more rigorously, for instance, by noting the equivalence of linear classifiers and the "Naive Bayes" assumption, which effectively assigns probabilities to margins (see Section 3.6 for a more detailed description).

Margin-based learning has been made popular by the support vector machine, which induces linear classifiers that maximize the margin to the training instances. Aiming for such a *large margin* turned out to be a remarkably well working bias. Since a linear classifier essentially represents a hyperplane in feature space, large margin classification can be seen as finding the hyperplane that maximizes the distance to the nearest training instance. A popular explanation for the success of SVMs is the fact that such a hyperplane is robust against noise in the test data, at least if the variation caused by the noise is smaller than the margin. Hence, unlike classical empirical risk minimization and the systems outlined in the preceding chapter, margin-based methods do not minimize the empirical error directly, but only implicitly through the margin.

Another nice property of weighted rule sets is the fact that sums of weighted quantities are well-investigated mathematical objects. There is a wealth of tools in linear algebra and probability theory to deal with linear forms and derive analytical results about them. If, for instance, the weights of a weighted rule set add to one, then the set of possible rule sets forms a convex set and the Jensen inequality can be used to upper-bound convex functions of weighted rule sets. Furthermore, linear functions of that type can be interpreted geometrically (as hyperplanes in a high-dimensional vector space) or probabilistically (as distributions over the set of rules). Since linear forms are continuous, related optimization criteria are generally differentiable and can thus be solved easily by established numerical methods. This is in contrast to the mathematics of DNF rule sets, whose discrete nature leads to hard combinatorial optimization problems.

In this chapter, we follow a margin-based approach to rule learning. Instead of the pure empirical and structural risk minimization procedures employed in the preceding chapter, we compare different margin-based optimization procedures. As before, we aim at simple, that is, small rule sets. However, we do so implicitly through regularization, instead of the structural risk minimization approach taken in Section 4.3. The main goal is to incorporate the favorable large margin bias into a rule learning system that favors small rule sets. Since we aim at small sets of explicit rules, there is no need for the kernel trick, which is applied in SVMs to induce an implicit higher dimensional feature space. Consequently, we can consider other margin-based optimization criteria, which are not compatible with the standard kernel-based approaches. As before, capacity control is an important component. In Section 5.3, we describe various bounds for capacity control with two optimization criteria, namely the empirical margin and the margin minus variance (MMV).

5.2 Empirical Risk Minimization for Weighted Rule Sets

Before delving into details, we need to introduce some basic definitions. First of all, recall that we assume the instances to be drawn i.i.d. from a fixed but unknown distribution D. D ranges over $\mathcal{X} \times \mathcal{Y}$, where $\mathcal{X}$ is the set of all possible instances and $\mathcal{Y} := \{-1, 1\}$ contain the target labels. The values -1 and 1 represent the negative and positive class respectively. A sample $X = \{x_1, \ldots, x_n\}$ of size n is drawn. In addition to this standard setting, we assume we already have a (possibly infinite) repository of rules $R = \{r_1, r_2, \ldots\}$, where a rule $r_j : \mathcal{X} \to [-1, 1]$ assigns a value between -1 and 1 to each instance. A typical rule could be "assign +1, if the color is red, and -1 otherwise" or "assign +1, if the size is greater than 1.2 and -1 otherwise". We will later present a scheme to enumerate the rules in the repository declaratively. The following results also hold for rules that assign intermediate values, such as 0.5 ("probably positive") or 0 ("don't know"), but in general we assume that $r(x) \in \{-1, +1\}$. Let $x_i(j)$ denote the result of the application of rule j on instance x_i. If we consider only the first m rules, we can represent the ith instance by the vector of rule values $x_i := (x_i(1), x_i(2), \ldots, x_i(m))^T$. Likewise, a weighted rule set can be given by a weight vector $w \in [-1, 1]^m$. An individual rule set w assigns class label $\text{sgn}(w^T x_i)$, so that instance x_i is positive, if $\sum_{j=1}^{m} w_j x_i(j) \geq 0$ and negative otherwise. Representing a rule set as a weight vector might seem unusual. However, even if the number of rules m is large, we can still deal with small rule sets, if most components of w are set to zero. Also note that the weight vector defines a hyperplane separating $[-1; 1]^m$ into two half-spaces so that rule sets in our setting are related to linear classifiers and perceptrons.

As already described, one usually uses the *zero-one loss* $l(w^T x, y) = \mathbf{I}[\text{sgn}(w^T x) \neq y]$ to define the *empirical error* $\hat{\varepsilon}_w := \frac{1}{n} \sum_{i=1}^{n} l(w^T x_i, y_i)$ and the *true error* $\varepsilon_w := \mathbf{E}_{(x,y) \sim D}\, l(w^T x, y)$. We are interested in finding a weighted rule set w that minimizes the true error ε_w. However, since D is unknown, we have no information about the true error and the best we can do is to minimize the empirical error $\hat{\varepsilon}$ instead. Note that scaling a weight factor w with a positive factor does not change the actual classification of the instances. Thus, in practice, one often restricts the space of possible weight factors. Unfortunately, due to the discontinuity of the zero-one loss, empirical risk minimization is a combinatorial optimization problem. It has been shown to be NP-hard and is also hard to approximate up to a fixed constant. One way to avoid those computational expenses is to optimize $\hat{\varepsilon}$ not directly, but through a related quantity. For example, support vector machines restrict the hypothesis space to weight vectors of unit length $w \in \{x \mid \|x\| = 1\}$ (as scaling w with a factor does not change the classification) and search for a w that maximizes the margin (or the soft margin) to the nearest training instances.[1] As discussed in the previous setting, this approach does not optimize the empirical risk directly, but only through a related quantity. While this does not identify the w, which minimizes the empirical error, it optimizes the empirical error to a certain extent, because the *hinge loss* $l_H(w, x, y) = \max\{0, yw^T x\}$ used in support vector machines dominates the zero-one loss. Hence, the training hinge loss can be seen as an upper bound for the empirical error. In a sense, the SVM sacrifices some training accuracy for the increased robustness of large margins. This compromise leads not only to better predictive accuracy in most cases, it also replaces the NP-hard empirical risk minimization problem with a feasible convex optimization problem. In the following, we present three different quantities that can be used in a similar way as a substitute for the empirical error and that allow for an efficient optimization procedure.

5.2.1 Soft-Margin Loss

The first quantity is simply the soft-margin loss as used in the SVM. Optimizing for a *large margin* is theoretically well-founded and the resulting quadratic programming problem can be solved efficiently. For rule learning, we are looking for an explicit representation of the rule set, not one that is implicitly defined by a kernel. Thus, we do not depend on the applicability of the kernel trick and can use other hypothesis spaces than the weight vectors of unit length $w \in \{x \mid \|x\| = 1\}$. In particular, for ensembles it is much more common to demand that the absolute weights in w sum to one, that is, to use the 1-norm instead of the 2-norm. This leads to the so-called *1-norm support vector machine* (Zhu *et al.*, 2004):

$$\begin{aligned} &\underset{w}{\text{minimize}} \sum_{i=1}^{m} \left[1 - w^T x_i\right]_+ \\ &\text{subject to } \|w\|_1 = \sum_{j=1}^{n} |w_j| = s \end{aligned} \tag{5.1}$$

where s is a free tuning parameter. Unlike the 2-norm SVM, the 1-norm SVM sets most weights to zero and assigns non-zero weights only to a few highly relevant rules. This behavior matches well with our purposes in rule learning. To implement the 1-norm SVM, we reformulate criterion (5.1) as

[1]Of course, the actual optimization problem is formulated differently.

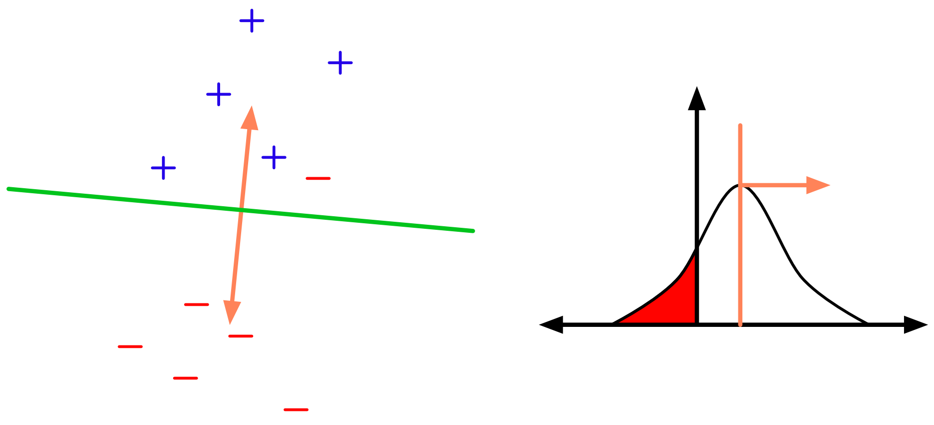

Figure 5.1: The distances between a hyperplane and the training instances (left) induce a distribution of margins (right). Empirical margin optimization aims for hyperplanes whose margin distribution features a large mean.

a linear program:

$$\begin{aligned} \underset{p}{\text{minimize}} \quad & \sum_{i=1}^{m} c_i \\ \text{subject to} \quad & \forall i \; y_i w^T x_i - c_i \geq 1, \\ & \sum_{j=1}^{n} |w_j| = s, \forall i \; c_i \geq 0 \end{aligned} \tag{5.2}$$

This can be solved using any of the established methods for linear programming.

5.2.2 Empirical Margin

The second quantity is the *empirical margin.* One problem of the SVM's large margin approach is that the classifier depends only on the support vectors and ignores the other data. If—as in Figure 5.2—the support vectors are not very representative of the underlying distribution, the classifier performs worse than necessary. A different approach is to consider the margin to all training instances. Ideally one would want to maximize the average distance from the separating hyperplane to the training instances. More formally: Given an instance (x, y) let $\mu_w(x, y) := w^T x \cdot y$ denote the *margin.* The margin is positive, if the instance is correctly classified by w and negative otherwise.[2] Similar to true and empirical error, one can

[2]Often, linear classifiers are defined to include an *intercept* w_0 so that the margin is $(w^T x + w_0)y$. However, a linear classifier $w \in \mathbb{R}^d$ with intercept $w_0 \in \mathbb{R}$ is equivalent to a

define the *empirical margin* $\hat{\mu}_w := \frac{1}{n}\sum_{i=1}^{n}\mu_w(x_i, y_i)$ and the *true margin* $\mathbf{E}_{(x,y)\sim D}\,\mu_w(x, y)$. As illustrated in Figure 5.1, a straightforward approach is to maximize $\hat{\mu}_w$ subject to $\|w\|_1 = 1$. However, it is easy to see that this procedure assigns the full weight to the single rule that agrees with the target class on most examples. It sets weight zero to all the other rules. In essence, aiming for maximum empirical margin with the 1-norm constraint is equivalent to selecting the single best rule. We therefore resort to using the 2-norm and minimize for large $\hat{\mu}_w$ subject to $\|w\|_2 = 1$. This sets almost all weights to non-zero values. The main advantage of this criterion is the fact that its solution can be calculated in closed form. Setting $d = \frac{1}{n}\sum_{i=1}^{n} x_i$, the problem can be stated as $\hat{\mu}_{opt} = \min_w -d^T w$ subject to $\|w\|_2^2 = 1$. The corresponding Lagrangian is $\mathcal{L}(w, \lambda) = -d^T w + \lambda(\|w\|_2^2 - 1)$ and setting its gradient $\nabla_w \mathcal{L}(w, \lambda) = 0$ yields $w_{opt} = \frac{1}{2\lambda}d$. It follows from duality that $\hat{\mu}_{opt} = \max_\lambda \mathcal{L}(\frac{1}{2\lambda}d, \lambda)$ and setting the derivative with regard to λ to zero we have $\lambda_{opt} = \frac{1}{2}\|d\|_2$. Thus, $w_{opt} = \frac{d}{\|d\|_2}$ is a solution to this optimization problem. From a theoretical point of view, the following lemma guarantees that optimizing for large μ_w also minimizes the true error ε_w:

Lemma 5.2.1. *Let (X, Y) be drawn according to D and let w be a weight vector with true error ε_w and true margin μ_w. Then:*

$$\varepsilon_w \leq 1 - \mu_w$$

Proof. Consider the random variable $C := 1 - \mu_w(X, Y)$. Since C is always positive, we can apply Markov's inequality. For any $\varepsilon > 0$:

$$\begin{aligned}\frac{1}{\varepsilon} &\geq \mathbf{Pr}\left[C \geq \varepsilon\,\mathbf{E}\left[C\right]\right]\\ &= \mathbf{Pr}\left[1 - \mu_w(X, Y) \geq \varepsilon - \varepsilon\,\mathbf{E}\left[\mu_w(X, Y)\right]\right]\\ &= \mathbf{Pr}\left[\mu_w(X, Y) \leq 1 - \varepsilon(1 - \mu_w)\right]\end{aligned}$$

Now set $\varepsilon = 1/(1 - \mu_w)$ and yield:

$$\varepsilon_w = \mathbf{Pr}[\mu_w(X, Y) \leq 0] \leq 1 - \mu_w$$

□

As the lemma is valid for all possible distributions, it also applies to every empirical distribution induced by a training set. Hence, the empirical error can also be upper-bounded by the empirical margin. Thus, optimizing

linear classifier $w' \in \mathbb{R}^{d+1}$ without intercept on a $d+1$-dimensional instance space, where the $d+1$th component is always set to one. For the sake of simplicity we therefore omit the intercept and instead assume that the rule repository includes a rule, which outputs $+1$ for all instances.

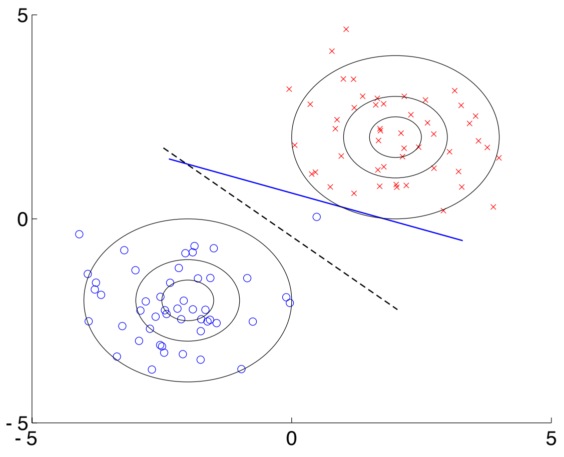

Figure 5.2: The maximum margin hyperplane (solid) and the margin minus variance hyperplane (dotted) for a small data set where positive and negative instances are drawn from two normal distributions with mean at (-2, -2) and (2, 2).

$\hat{\mu}$ can be seen as a computationally feasible relaxation of the infeasible direct empirical risk minimization problem. In particular, if $\hat{\mu}_p$ reaches the largest possible value one, the empirical error is guaranteed to be zero.

5.2.3 Margin Minus Variance

The third approach tries to improve on the second one by also incorporating the variance. One can estimate the sample variance of the margins from a training set; the *empirical variance* is $\hat{\sigma}_w := \frac{1}{n-1}\sum_{i=1}^{n}(\mu_w(x_i, y_i) - \hat{\mu}_w)^2$, the *true variance* is $\sigma_w := \mathbf{E}_{(x,y)\sim D}(\mu_w(x, y) - \mu_w)^2$. Using this notation, one can look for weight vectors w that maximize the empirical margin but minimize the empirical variance. This is illustrated in Figure 5.3. The goal is to minimize the probability of having a negative margin, that is the red area in the (empirical) margin distribution in Figure 5.3. To do so, we aim for a classifier which has high average margin and low variance, more formally:

$$\begin{aligned} &\underset{w}{\text{maximize}}\, \hat{\mu}_w - \hat{\sigma}_w &(5.3)\\ &\text{subject to } \|w\|_1 = \sum_{j=1}^{m} |w_j| = 1 \end{aligned}$$

We call the optimization function *margin minus variance* (MMV) and denote it by $\hat{\gamma}_w := \hat{\mu}_w - \hat{\sigma}_w$. It is easy to see that maximizing $\hat{\gamma}$ is a quadratic

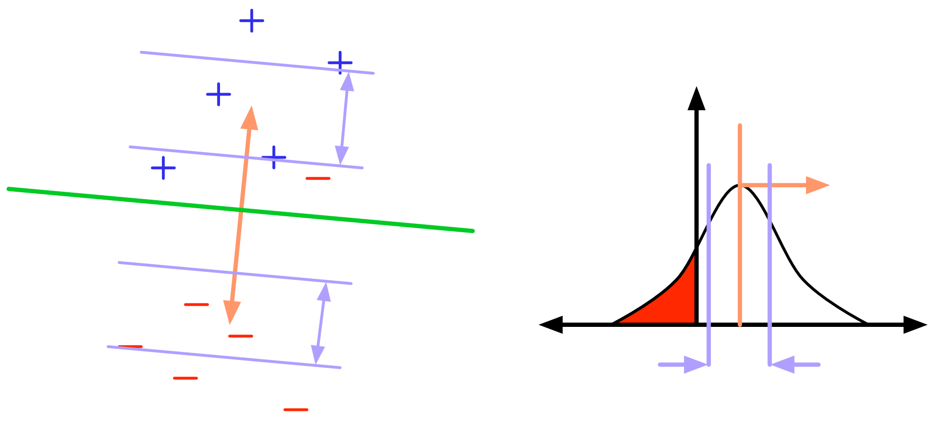

Figure 5.3: The distances between a hyperplane and the training instances (left) induce a distribution of margins (right). MMV optimization aims for hyperplanes whose margin distribution features a large mean and a small variance.

optimization problem and can therefore be solved efficiently. In Figure 5.2, the dotted line denotes the maximum-MMV hyperplane, while the solid line indicates the maximum-soft-margin hyperplane. Even though the MMV hyperplane misclassifies one instance in the training set, it obviously approximates the decision boundary for the underlying data distribution (the bisecting line at the origin) better than the soft-margin hyperplane. Just as the 1-norm SVM, MMV sets most weights to zero and assigns non-zero weights only to the best few rules. For example, during a test run on the australian data set MMV maximization sets non-zero weights only to 15 out of the 1600 rules.

Another nice property is that $\hat{\gamma}_w$ is an unbiased estimator of the *true margin minus variance* $\gamma_w := \mu_w - \sigma_w$ and this quantity can be used to upper-bound the true error. This is intuitively intriguing: A hyperplane that features a large true margin and a low true variance should have a low true error, because most instances are clearly classified correctly (large margin) and there are only a few exceptions (low variance). The following lemma quantifies this:

Lemma 5.2.2. *Let S be drawn according to D and let w be a hyperplane with true error ε_w and true MMV γ_w. Then:*

$$\varepsilon_w \leq \begin{cases} \frac{\gamma_w}{4\gamma_w^2+\gamma_w} & \textit{if } \gamma_w \leq 0.5 \\ \frac{1-\gamma_w}{2-\gamma_w} & \textit{if } \gamma_w > 0.5 \end{cases}$$

Proof. The Chebyshev inequality states that

$$\begin{aligned}\varepsilon_w &= \mathbf{Pr}[\mu(X,Y) \leq 0] \\ &\leq \frac{Var[\mu(X,Y)]}{Var[\mu(X,Y)] + (\mathbf{E}[\mu_w(X,Y)])^2} \\ &= \frac{\mu_w - \gamma_w}{\mu_w - \gamma_w + \mu_w^2}\end{aligned}$$

We will now determine the value of μ_w that maximizes the right hand side of this inequality to yield the final result. Denote the right hand side by $f(\mu_w, \gamma_w)$:

$$f(\mu_w, \gamma_w) := \frac{\mu_w - \gamma_w}{\mu_w - \gamma_w + \mu_w^2}$$

Then we would like to find

$$\mu'_w := \operatorname*{argmax}_{\mu_w} f(\mu_w, \gamma_w)$$

subject to the constraint that $\gamma_w \leq \mu_w \leq 1$ (since the variance is always positive). To find this optimum, we set the derivative to zero:

$$\frac{\partial f}{\partial \mu_w} = \frac{-\mu_w^2 + 2\mu_w\gamma_w}{(\mu_w - \gamma_w + \mu_w^2)^2} = 0$$

The only positive solution to this equation is $\mu'_w = 2\gamma_w$. So, for $\gamma_w \leq 0.5$ we yield:

$$\varepsilon_w \leq f(2\gamma_w, \gamma_w) = \frac{\gamma_w}{4\gamma_w^2 + \gamma_w}$$

If $\gamma_w > 0.5$, the optimal value of μ_w is violating the constraint that $\mu_w \leq 1$. As f is increasing until $2\gamma_w$, the optimal margin is thus 1 and we yield:

$$\varepsilon_w \leq f(1, \gamma_w) = \frac{1 - \gamma_w}{2 - \gamma_w}$$

□

Just as Lemma 5.2.1 did for the empirical margin, this lemma ensures that $\hat{\gamma}$ can be seen as a computationally feasible relaxation of the infeasible direct empirical risk minimization problem. Again, if $\hat{\gamma}_w$ reaches the largest possible value one, the empirical error is guaranteed to be zero.

We formulate the MMV optimization task as a standard quadratic programming problem: minimize$_x$ $x^T G x + d^T x$, where $G \in \mathbb{R}^{n \times n}$ denotes the quadratic part and $d \in \mathbb{R}^d$ is the linear part. This corresponds naturally to the two parts in the MMV $\hat{\gamma}_w = \hat{\mu}_w - \hat{\sigma}_w$. The empirical margin $\hat{\mu}$ is linear, while the quadratic part can be written as a function of the empirical covariance matrix G. If the training set is given as a $m \times n$ matrix T with $t_{ij} := x_i(j)y_i$, G and d can be computed as follows. The linear part is $d := \frac{1}{n}(\vec{1}Tw)$. The quadratic part is $G := -\frac{2}{n-1}[(T^T T) - \frac{1}{n}(\vec{1}T)^T(\vec{1}T)]$, where $\vec{1}$ is a vector containing n ones. Typical approaches to solving this optimization problem compute the Lagrangian $\mathcal{L}(w, \lambda) := \frac{1}{2}w^T G w + d^T w - \lambda(\sum_{i=1}^{m} |w_i| - 1)$ and search for parameters (w, λ) that set the Lagrangian's gradient to zero. Unfortunately, the 1-norm contains absolute values, and is thus not differentiable at weight vectors that have some weight set to zero. Consequently, the gradient is not defined at those weight vectors, and gradient-based optimization approaches can not be used. Fortunately, it is easy to remove the non-differentiabilities in the 1-norm-constraint. To do so, one can decompose each weight w_i in the weight vector w into a positive part $w_i^+ \geq 0$ and a negative part $w_i^- \geq 0$ so that $w_i = w_i^+ - w_i^-$. Using the notation $w = (w_1^+, \ldots, w_n^+, w_1^-, \ldots, w_n^-)^T$ the optimization problem becomes:

$$\underset{w}{\text{maximize}} \quad \frac{1}{2}w^T \begin{pmatrix} G & -G \\ -G & G \end{pmatrix} w + w^T \begin{pmatrix} d \\ -d \end{pmatrix}$$
$$\text{subject to} \sum_{j=1}^{n}(w_j^+ + w_j^-) = 1, \forall j \; w_j^+, w_j^- \geq 0$$

This is a semidefinite quadratic programming problem with one equality and $2n$ inequality constraints. It can be solved using any established algorithm for constrained optimization. Usually, the computationally most demanding part of the process is to determine the set of active constraints. As the number of non-zero weights is usually rather small, this optimization step is typically very fast.

All of the three described optimization criteria can be applied to derive weights for an existing repository of rules. Before we describe a system to generate such rules in Section 5.4, we deal with the problem of capacity control for weighted rule sets.

5.3 Structural Risk Minimization for Weighted Rule Sets

In the preceding section we presented three quantities that can be used to determine weights for a class of predefined rules. Obviously, the success of this algorithm depends to a large degree on the choice of this class, and, in particular, on its size. If the class is too small, the algorithm will most likely underfit, if it is too large, it will overfit. The standard approach to tackle this dilemma is to start with a small class, then iteratively increase the size of the class. In each step, one determines for each class the hypothesis that explains the training set best, and an estimate of the structural risk of the class. The best hypothesis class is the one which minimizes the sum of empirical risk and structural risk. Similar approaches include structural risk minimization (Vapnik, 1995) or model selection based on Rademacher risk bounds (Koltchinskii, 2001).

Unfortunately, most of the established model selection strategies do not work well in our setting. The first problem is, that often only a few large hypothesis classes are used, so that the model selection is rather coarse. For instance, a popular scheme is to use cascading classes of polynomial kernels with increasing exponent in SVMs. In rule learning, though, even very small hypothesis classes can overfit. For example the hypothesis class of 1-term DNF formulae is a subset of a linear kernel (that is, a polynomial kernel with exponent one), and nevertheless, empirical risk minimization overfits with this hypothesis class on the UCI hearth data set (see Section 4.3.2). Apparently, in rule learning one needs more fine-grained and (at least initially) slower increasing hypothesis class sizes.

The risk bounds that are used for model selection usually deal directly with the empirical error. This does not match very well with the approaches taken in the preceding section. First of all, the empirical error is a discrete quantity and often does not change in steps from one hypothesis class to the other, making it hard to handle. The quantities presented in Section 5.2 provide a more fine-grained estimate of the empirical risk. In the following, we present concentration inequalities that work directly on the empirical margin and the MMV. It seems to be more sensible to use those risk bounds depending on the the actual quantities that are optimized instead of the (possibly fluctuating) empirical error. We present three different ways to derive such risk bounds. First of all, in Section 5.3.1 we extend the work on Rademacher penalties to upper-bound the true margin and true MMV by quantities that depend on a randomized version of the training set. A different approach is taken in Section 5.3.2, where the PAC-Bayesian bound is adjusted to deal with true margin and MMV. Finally, in Section 5.3.3 we present novel bounds that are constructed to only incorporate the rule repository size. Since there is a wide range of results on the SVM, we concentrate on bounds for maximum margin and MMV optimization.

5.3.1 Rademacher Bounds for MMV

Rademacher complexities for data-dependent generalization bounds were introduced by Koltchinskii (2001) and Bartlett and Mendelson (2003). The main idea is to construct a randomized version of the training set by randomly flipping the class labels. The training accuracy of the learning algorithm on this randomized training set can then be used to estimate the data-dependent structural risk of the system. It can been shown that this estimate is always better than the traditional estimate based on VC-dimensions (Kääriäinen, 2004).

Consider *Rademacher random variables*, that is, variables taking the values -1 and +1 with probability 0.5 each. Assume we are given n Rademacher random variables ρ_i, one for each training instance. The Rademacher error complexity is given by

$$R_X := \sup_{c \in \mathcal{H}} \left| \frac{1}{n} \sum_{i=1}^{n} \rho_i l(c, x_i, y_i) \right| \tag{5.4}$$

This random variable can be used to upper-bound the true error of an linear classifier in the following way:

Theorem 5.3.1 (Rademacher bound (Koltchinskii, 2001; Kääriäinen *et al.*, 2004))**.** *Let (X, Y) be a training set of size n drawn i.i.d. from D and let $\{\rho_1, \dots, \rho_n\}$ be the outcome of tossing a fair coin n times. Then, for any classifier c and $\delta > 0$:*

$$\mathbf{Pr}\left[\varepsilon_c \leq \hat{\varepsilon}_c + 2R_X + 5\sqrt{\frac{\ln \frac{2}{n}}{2n}}\right] \geq 1 - \delta$$

If a learning system is known to find a classifier that minimizes the empirical risk on a data set, the supremum in R_X can be calculated by applying the system on two training sets where the class labels are flipped according to the r_i. An application of this trick can be found in Kääriäinen *et al.* (2004), where a pruning algorithm is applied to compute R_X so that the Rademacher bound can be used for model selection. This trick is intuitively appealing, because the application of the learning system on a randomized training set measures how strongly the learning system can overfit on data, which is random, but features the same characteristics as the original training set.

Since we can find a weight vector that minimizes the empirical margin on a data set, we could make use of a similar bound to upper-bound the true margin instead of the true error. Hence, we are interested in the *Rademacher margin complexity*

$$S_X := \sup_{w \in [-1,1]^m} \left| \frac{1}{n} \sum_{i=1}^{n} \rho_i w^T x_i \right| \tag{5.5}$$

and the *Rademacher second margin moment complexity*

$$T_X := \sup_{w \in [-1,1]^m} \left| \frac{1}{n} \sum_{i=1}^{n} \rho_i (w^T x_i)^2 \right| \tag{5.6}$$

Note that S_X and T_X are random variables that depend on the unlabeled training instances x_i and the Rademacher variables r_i, but not on a specific weight vector w or the labels y_i. Intuitively, S_x gives the best margin a weight vector can achieve on a training set with random class labels. In order to derive Rademacher bounds for the true margin and true MMV depending on S_X and T_X, we need the McDiarmid inequality, a powerful concentration inequality that can be used to bound functions of independent random variables.

Theorem 5.3.2 (McDiarmid, (McDiarmid, 1989))**.** *Let $X_1, X_2, \ldots, X_n$ be independent (not necessarily identically distributed) random variables. If for a function $g : X_1 \times \ldots X_n \to \mathbb{R}$ there are some nonnegative constants $c_1, \ldots, c_n$ so that*

$$\sup_{x_1,\ldots,x_n,x'_i} |g(x_1,\ldots x_n) - g(x1,\ldots,x_{i-1},x'_i,x_{i+1},\ldots,x_n)| \leq c_i, 1 \leq i \leq n,$$

then for all $\varepsilon > 0$:

$$\mathbf{Pr}\left[g(X_1,\ldots,X_n) - \mathbf{E}[g(X_1,\ldots,X_n)] \geq \varepsilon\right] \leq \exp\left(-\frac{2\varepsilon^2}{\sum_{i=1}^n c_i^2}\right) \text{ and}$$
$$\mathbf{Pr}\left[\mathbf{E}[g(X_1,\ldots,X_n)] - g(X_1,\ldots,X_n) \geq \varepsilon\right] \leq \exp\left(-\frac{2\varepsilon^2}{\sum_{i=1}^n c_i^2}\right)$$

In other words, if the value of a function g changes only up to a constant when replacing one of the input parameters, then the random variable $g(X_1,\ldots,X_n)$ is sharply concentrated around its mean. The application of McDiarmid's inequality and a symmetrization argument give rise to the following Rademacher bound for the true margin:

Theorem 5.3.3. *Let (X, Y) be the application of the rules to a training set of size n drawn i.i.d. from D and let $\{\rho_1,\ldots,\rho_n\}$ be the outcome of tossing a fair coin n times. Then, for any linear classifier $w \in [-1,1]^m$ and $\delta > 0$:*

$$\mathbf{Pr}\left[\mu_w \leq \hat{\mu}_w + 2S_X + 6\sqrt{\frac{\ln\frac{2}{n}}{2n}}\right] \geq 1 - \delta$$

Proof. The proof follows along the lines of its counterpart for the empirical error (Kääriäinen *et al.*, 2004). First we show that S_X is concentrated sharply around its mean. Observe, that replacing one pair (x_i, ρ_i) in the definition of S_X by any new pair (x'_i, ρ'_i) changes the value of S_X by at most $2/n$. Thus, one can apply Theorem 5.3.2 and gain:

$$\mathbf{Pr}\left[S_X \leq \mathbf{E}[S_X] - 2\sqrt{\frac{\ln(1/\delta)}{2n}}\right] \leq \delta \tag{5.7}$$

The following symmetrization argument shows how the expectation of S_X is related to the difference between empirical and true margin. Let (X_1, Y_1),

$\ldots, (X_n, Y_n)$ be a collection of independent random variables drawn according to D and $(X_1', Y_1'), \ldots, (X_n', Y_n')$ with the same distribution. Both collections form training sets of the size n. Let $\mathcal{B}_p^n = \{w \in \mathbb{R}^n | \|w\|_p \leq 1\}$ denote the p-norm ball in $\mathbb{R}^n$, so that the weight vectors $w \in \mathcal{B}_\infty^m$.

$$\begin{aligned}
\mathbf{E}\left[\sup_{w \in \mathcal{B}_\infty^m} |\mu_w - \hat{\mu}_w|\right] &= \mathbf{E}\left[\sup_{w \in \mathcal{B}_\infty^m} \left| \mathbf{E}\left[\frac{1}{n}\sum_{i=1}^n Y_i' w^T X_i'\right] - \frac{1}{n}\sum_{i=1}^n Y_i w^T X_i \right|\right] \\
&\leq \mathbf{E}\left[\sup_{w \in \mathcal{B}_\infty^m} \left| \frac{1}{n}\sum_{i=1}^n Y_i' w^T X_i' - \frac{1}{n}\sum_{i=1}^n Y_i w^T X_i \right|\right] && (5.8) \\
&= \mathbf{E}\left[\frac{1}{n}\sup_{w \in \mathcal{B}_\infty^m} \left| \sum_{i=1}^n \rho_i \left[Y_i' w^T X_i' - Y_i w^T X_i\right] \right|\right] && (5.9) \\
&\leq 2\,\mathbf{E}\left[\frac{1}{n}\sup_{w \in \mathcal{B}_\infty^m} \left| \sum_{i=1}^n \rho_i Y_i w^T X_i \right|\right] && (5.10) \\
&= 2\,\mathbf{E}\left[\frac{1}{n}\sup_{w \in \mathcal{B}_\infty^m} \left| \sum_{i=1}^n \rho_i w^T X_i \right|\right] \\
&= 2\,\mathbf{E}\left[S_X\right]
\end{aligned}$$

Inequality (5.8) and (5.10) are applications of Jensen's inequality, whereas (5.9) follows from the fact that the ρ_i essentially swap instances between (X, Y) and (X', Y'). Since the two samples are drawn i.i.d. swapping instances does not change the expectation.

Now, we apply McDiarmid's concentration inequality again to prove that the difference between the empirical and true quantities is also concentrated around its mean. Consider the random variable $\sup_{w \in \mathcal{B}_\infty^m} |\mu_w - \hat{\mu}_w| = \sup_{w \in \mathcal{B}_\infty^m} |\mu_w - n^{-1}\sum_{i=1}^n y_i w^T x_i|$. If one replaces a pair (x_i, y_i) with a pair (x_i', y_i'), the value changes by at most $(2/n)$. Thus, we can again apply Theorem 5.3.2:

$$\mathbf{Pr}\left[\sup_{w \in \mathcal{B}_\infty^m} |\mu_w - \hat{\mu}_w| \geq \mathbf{E}\left[\sup_{w \in \mathcal{B}_\infty^m} |\mu_w - \hat{\mu}_w|\right] + 2\sqrt{\frac{\ln(1/\delta)}{2n}}\right] \leq \delta \qquad (5.11)$$

We are now in the position to give a bound that does only depend on the empirical margin and the Rademacher margin complexity. Observe that for all w

$$\mu_w \geq \hat{\mu}_w - \sup_{w \in \mathcal{B}_\infty^M} |\mu_w - \hat{\mu}_w|$$

Because of (5.11) and the symmetrization inequality above we know that with probability at least $1 - \delta/2$:

$$\begin{aligned}\mu_w &\geq \hat{\mu}_w - \sup_{w \in \mathcal{B}_\infty^m} |\mu_w - \hat{\mu}_w| \\ &\geq \hat{\mu}_w - \mathbf{E}\Big[\sup_{w \in \mathcal{B}_\infty^m} |\mu_w - \hat{\mu}_w|\Big] - 2\sqrt{\frac{\ln(2/\delta)}{2n}} \\ &\geq \hat{\mu}_w - 2\,\mathbf{E}\big[S_X\big] - 2\sqrt{\frac{\ln(2/\delta)}{2n}}\end{aligned}$$

Thus, with (5.7) we have that with probability $1 - \delta$

$$\mu_w \geq \hat{\mu}_w - 2S_X - 6\sqrt{\frac{\ln(2/\delta)}{2n}} \tag{5.12}$$

□

It is easy to see how this theorem can be used for capacity control with the maximum margin optimization criterion. Observe that the term in the absolute value of the definition of S_X in (5.5) is essentially the empirical margin for a training set, where the class labels y_i are replaced by the Rademacher variables ρ_i. Thus, one can simply generate two new training sets, the first by setting $y_i = \rho_i$, the second with $y_i = -\rho_i$ and apply the empirical margin maximization algorithm on them. The value of S_X is then the maximum of the two values, because for $x \in \mathbb{R} : \max\{x, -x\} = |x|$ so that taking the maximum accommodates for the absolute value in (5.5). With this, $S_X + \hat{\varepsilon}_w$ gives a practical data-dependent estimate of the structural risk for empirical margin optimization.

To obtain a similar result for MMV optimization, the following theorem considers the second moment of the margin distribution.

Theorem 5.3.4. *Let (X, Y) be the application of the rules to a training set of size n drawn i.i.d. from D and let $\{\rho_1, \ldots, \rho_n\}$ be the outcome of tossing a fair coin n times. Then, for any linear classifier $w \in [-1, 1]^m$ and $\delta > 0$:*

$$\mathbf{Pr}\left[\gamma_w \geq \hat{\gamma}_w - 6S_X - 2T_X - 20\sqrt{\frac{\ln(3/\delta)}{2n}}\right] \geq 1 - \delta$$

Proof. The proof follows the same scheme as the one for the margin, except for a more involved argument to apply the symmetrization. It is clear that T_X is concentrated around its mean, just as its counterpart S_X:

$$\mathbf{Pr}\left[T_X \leq \mathbf{E}[T_X] - \sqrt{\frac{\ln(1/\delta)}{2n}}\right] \leq \delta \tag{5.13}$$

Let $\hat{\nu}_w := \frac{1}{n}\sum_{i=1}^n (w^T x_i)^2$ denote the empirical second margin moment and $\nu_w := \mathbf{E}(w^T x)^2$ be the true second margin moment. Now, observe that

$$\begin{aligned}\mathbf{E}\Big[\sup_{w\in\mathcal{B}_\infty^m}|\gamma-\hat{\gamma}|\Big] &\leq \mathbf{E}\Big[\sup_{w\in\mathcal{B}_\infty^m}|\mu_w-\hat{\mu}_w| + \sup_{w\in\mathcal{B}_\infty^m}|\nu_w-\hat{\nu}_w| + \sup_{w\in\mathcal{B}_\infty^m}|\mu_w^2-\hat{\mu}_w^2|\Big] \\ &\leq 3\,\mathbf{E}\Big[\sup_{w\in\mathcal{B}_\infty^m}|\mu_w-\hat{\mu}_w|\Big] + \mathbf{E}\Big[\sup_{w\in\mathcal{B}_\infty^m}|\nu_w-\hat{\nu}_w|\Big]. \qquad (5.14)\end{aligned}$$

The second inequality holds, because $|x^2-y^2| \leq 2|x-y|$ for all $x, y \in [-1,1]$. Applying the symmetrization argument in the proof of Theorem 5.3.3 to the two summands above yields $3\,\mathbf{E}[\sup_{w\in\mathcal{B}_\infty^m}|\mu_w-\hat{\mu}_w|] \leq 6S_X$ and $\mathbf{E}[\sup_{w\in\mathcal{B}_\infty^m}|\nu_w-\hat{\nu}_w|] \leq 2T_X$. Thus, the sum of the two expectations can be upper-bounded as follows:

$$\mathbf{E}\Big[\sup_{w\in\mathcal{B}_\infty^m}|\gamma_w-\hat{\gamma}_w|\Big] \leq 6\,\mathbf{E}\left[S_X\right] + 2\,\mathbf{E}\left[T_X\right] \qquad (5.15)$$

Now, we show that the random variable $\sup_{w\in\mathcal{B}_\infty^m}|\gamma-\hat{\gamma}|$ is also concentrated around its mean. Consider the case where one replaces a pair (x_k, y_k) in the training set with a pair (x_k', y_k'). As discussed in the proof of Theorem 5.3.3, the margin part of MMV changes by at most $(2/n)$. The empirical variance part can be written as follows:

$$\hat{\sigma}_w = \frac{1}{n(n-1)}\sum_{i>j}(y_i w^T x_i - y_j w^T x_j)^2$$

Thus, if one changes the instance (x_k, y_k), $n-1$ summands are affected. Each summand can change by at most 4, because $\sup_{b,b'\in[-1,1]}[(a-b)^2-(a-b')^2] \leq 4$ for $a \in [-1,1]$. Altogether the variance part can change by at most $4/n$, so that the empirical MMV changes by at most $6/n$. McDiarmid's inequality yields:

$$\mathbf{Pr}\left[\sup_{w\in\mathcal{B}_\infty^m}|\gamma_w-\hat{\gamma}_w| \geq \mathbf{E}\Big[\sup_{w\in\mathcal{B}_\infty^m}|\gamma_w-\hat{\gamma}_w|\Big] + 6\sqrt{\frac{\ln(1/\delta)}{2n}}\right] \leq \delta \qquad (5.16)$$

As above, we can combine (5.16) and (5.15) to yield that with probability at least $1-\delta/3$:

$$\begin{aligned}\gamma_w &\geq \hat{\gamma}_w - \sup_{w\in\mathcal{B}_\infty^m}|\gamma_w-\hat{\gamma}_w| \\ &\geq \hat{\gamma}_w - \mathbf{E}\Big[\sup_{w\in\mathcal{B}_\infty^m}|\gamma_w-\hat{\gamma}_w|\Big] - 6\sqrt{\frac{\ln(3/\delta)}{2n}} \\ &\geq \hat{\gamma}_w - 6\,\mathbf{E}\left[S_X\right] - 2\,\mathbf{E}\left[T_X\right] - 6\sqrt{\frac{\ln(3/\delta)}{2n}}\end{aligned}$$

From (5.13) and (5.7) we have with probability $1-\delta$:

$$\gamma_w \geq \hat{\gamma}_w - 6S_X - 2T_X - 20\sqrt{\frac{\ln(3/\delta)}{2n}}$$

□

The bound for MMV contains considerably larger constants than the bound for the empirical margin. This is mainly due to the loose bound in (5.14), which is necessary to make the symmetrization trick work for the squared margin part of the variance. It is unclear how one can work around this problem. From a practical point of view, the Rademacher penalization for MMV involves the computation of four additional convex optimization problems, two for S_X and two for T_X. This means the time necessary for computing the structural risk estimate $6S_X + 2T_X + \hat{\gamma}_w$ is prohibitively large for larger data sets. Rademacher complexities are therefore only suitable for a comparably small number of instances and rules.

5.3.2 PAC-Bayesian Bounds for MMV

For an estimate that can be computed in closed form without the need for expensive optimization procedures, we consider PAC-Bayesian bounds. The PAC-Bayesian bound for the empirical loss of a classifier is given in Theorem 3.6.1 in Section 3.6. In the following, we will modify the original inequality so that it bounds the true margin and the true MMV given its empirical counterparts. Recall that the original PAC-Bayesian theorem deals with the expected error of the Gibbs classifier, that is, the meta-classifier, which draws a hypothesis according to a probability distribution over the hypothesis space $\mathcal{H}$ and then predicts with this hypothesis. To make this bound work in our case, we need to relate a weighted rule set, or, more generally, a linear classifier to a Gibbs classifier.

This is rather straightforward. Recall that the margin of a weighted rule set $w \in \{x \in \mathbb{R}^m | \|x\|_1 = 1\}$ on instance (x, y) is $yw^T x$. The weight vector is "almost" a distribution, in the sense that the absolute values of its components sum to one: $\sum_{i=1}^m |w_i| = 1$. If all $w_i \geq 0$, w would indeed define a probability distribution on the index set $\{1, \ldots, m\}$. This probability distribution could then form the basis for a Gibbs classifier, allowing for a PAC-Bayesian analysis. To make this happen, we apply the following trick: Instead of dealing with the original weight vector w and the original training data X, we construct a $2m$-dimensional weight vector $w' := ([w_1]_+, \ldots, [w_m]_+, [w_1]_-, \ldots, [w_m]_-)^T$ and use a training matrix $X' := (X - X)$ with twice the number of columns. Here, $[x]_+ := \max\{x, 0\}$ is defined to be zero for negative weights and $|x|$ otherwise, and $[x]_- := \max\{0, -x\}$ is zero for positive values and $|x|$ otherwise. It is easy to see that the margin of the original weight vector w on an original training instance (x, y) is equal to the margin of the new weight vector on a duplicated instance: $yw^T x = yw'^T x'$. Since the components of w' are positive, it induces a distribution on $\{1, \ldots, 2m\}$ and the PAC-Bayesian analysis applies. For ease of notation, we will assume for the rest of this section that the trick has already been applied to the data and that $w \in [0, 1]^m$.

Thus, the weight vector w defines a distribution R_w on $\{1, \ldots, m\}$, where the probability of drawing an index k is simply the kth weight $\mathbf{Pr}[R_w = k] = w_k$. In this notation, the margin of w on an instance (x, y) can be written as $y\,\mathbf{E}_{r\sim R_w}[x(r)]$. The empirical margin on a training set (X, Y) is then $\hat{\mu}_w := \frac{1}{n}\sum_{i=1}^n y_i\,\mathbf{E}_{r\sim R_w}[x_i(r)]$, and the true margin is $\mu_w := \mathbf{E}_{(x,y)\sim D}\, y\,\mathbf{E}_{s\sim R_w}[x(r)]$. Likewise, the empirical and true MMV are:

$$\hat{\gamma}_w := \hat{\mu}_w - \frac{1}{n-1}\sum_{i=1}^n \Big(\mathop{\mathbf{E}}_{r\sim R_w}\left[x_i(r)\right] - \hat{\mu}_w \Big)^2$$

$$\gamma_w := \mu_w - \mathop{\mathbf{E}}_{(x,y)\sim D}\left[\Big(\mathop{\mathbf{E}}_{r\sim R_w}\left[x_i(r)\right] - \mu_w \Big)^2 \right]$$

Before we can give the results, we need to introduce some additional notation. We denote the *Kullback-Leibler divergence* between two distributions R and S over a countable space $\mathcal{H}$ by $D(R\|S) := \sum_{c\in\mathcal{H}} R(c)\ln(R(c)/S(c))$. Furthermore, given a distribution R on space $\mathcal{R}$ and a distribution S on space $\mathcal{S}$, $R \times S$ denotes the *product distribution* on $\mathcal{R} \times \mathcal{S}$, which assigns probability $\mathbf{Pr}[R = r]\,\mathbf{Pr}[S = s]$ to the event $(r, s) \in \mathcal{R} \times \mathcal{S}$. Sometimes we write R^2 as a short cut for $R \times R$. The following technical lemma is used in the proofs of the PAC-Bayesian theorems for margin and MMV.

Lemma 5.3.5. *For $\beta > 0, K > 0$, and $R, S, x \in \mathbb{R}^m$ satisfying $R_j \geq 0, S_j \geq 0, x_j \geq 0, \sum_{i=1}^m R_j = 1$, we have that if*

$$\sum_{j=1}^n R_j e^{\beta x_j^2} \leq K$$

then

$$\sum_{j=1}^n S_j x_j \leq \sqrt{\frac{D(R\|S) + \ln K}{\beta}}$$

For a proof, see lemma 21 in McAllester (1999). With this, we are in the position to give PAC-Bayesian bounds for average margin and MMV.

Theorem 5.3.6. *Let $v \in [0, 1]^m, \|v\|_1 = 1$ be a fixed weight vector, and $w \in [0, 1]^m, \|w\|_1 = 1$ be a weight vector depending on a training set (X, Y) of size n drawn i.i.d. from a fixed distribution D. Then, for any $\delta > 0$:*

$$\mathop{\mathbf{Pr}}_{(X,Y)\sim D^n}\left[\mu_w \leq \hat{\mu}_w + \sqrt{\frac{D(R_w\|R_v) + \ln\frac{1}{\delta} + \ln n}{0.5n - 1}} \right] \geq 1 - \delta$$

Proof. Let $\bar{\mu}_r := |\mathbf{E}_{(x,y)\sim D}[yx(r)] - \frac{1}{n}\sum_{i=1}^n y_i x_i(r)|$ denote the difference between the expectation of the rth feature and the corresponding empirical estimate. This is a random variable depending on (X, Y). As a first step, we prove that

$$\underset{(X,Y)\sim D^n}{\mathbf{Pr}}\left[\underset{r\sim R_v}{\mathbf{E}}\, e^{(0.5n-1)\bar{\mu}_r^2} \leq \frac{n}{\delta}\right] \geq 1-\delta \tag{5.17}$$

To prove this, note that changing one instance in (X, Y) changes $\bar{\mu}_r$ by at most $\frac{2}{n}$. Thus, for arbitrary $r \in \mathbb{N}$, McDiarmid's inequality (Theorem 5.3.2) yields:

$$\underset{(X,Y)\sim D^n}{\mathbf{Pr}}\left[\bar{\mu}_r \geq x\right] \leq 2e^{-0.5nx^2} \tag{5.18}$$

Now, we investigate the distribution of the random variable $\bar{\mu}_r$. Let $f : [0, 2] \to \mathbb{R}$ denote the density function of $\bar{\mu}_r$ so that $\mathbf{Pr}[\bar{\mu}_r \leq x] = \int_0^x f(a)da$. Since we want to find an upper bound for $\mathbf{E}_{(X,Y)\sim D^n}\, e^{(0.5n-1)\bar{\mu}_r^2}$, we look for a density f_{max} that achieves the maximum of this term. More precisely, we look for the density f which maximizes $\int_0^\infty e^{(0.5n-1)\bar{\mu}_r^2} f(\bar{\mu}_r)\, d\bar{\mu}_r$, subject to the constraint (5.18) that $\int_x^\infty f(\bar{\mu}_r)\, d\bar{\mu}_r \leq 2e^{-0.5nx^2}$. The maximum is achieved when $\int_x^\infty f(\bar{\mu}_r)\, d\bar{\mu}_r = 2e^{-0.5nx^2}$. Taking the derivative yields that $f_{max}(\bar{\mu}_r) = 2n\bar{\mu}_r e^{-0.5n\bar{\mu}_r^2}$. Therefore,

$$\begin{aligned}
\underset{(X,Y)\sim D^n}{\mathbf{E}}\, e^{(0.5n-1)\bar{\mu}_r^2} &\leq \int_0^\infty e^{(0.5n-1)\bar{\mu}_r^2} f_{max}(\bar{\mu}_r)\, d\bar{\mu}_r \\
&= \int_0^\infty 2n\bar{\mu}_r e^{(0.5n-1)\bar{\mu}_r^2} e^{-0.5n\bar{\mu}_r^2}\, d\bar{\mu}_r \\
&= \int_0^\infty 2n\bar{\mu}_r e^{-\bar{\mu}_r^2}\, d\bar{\mu}_r \\
&= n
\end{aligned}$$

Since this upper bound is valid for all indices r, it holds also for the expectation over those indices:

$$\underset{(X,Y)\sim D^n}{\mathbf{E}}\ \underset{r\sim R_v}{\mathbf{E}}\, e^{(0.5n-1)\bar{\mu}_r^2} \leq n$$

Inequality (5.17) follows from this and Markov's inequality. Applying Lemma 5.3.5 to (5.17) with $K = \frac{n}{\delta}, R = R_w, S = R_v, x = (\bar{\mu}_1, \ldots, \bar{\mu}_m)^T, \beta = 0.5n-1$ yields:

$$\underset{(X,Y)\sim D^n}{\mathbf{Pr}}\left[\sum_{j=1}^n w_j\bar{\mu}_j \leq \sqrt{\frac{D(R_w\|R_v) + \ln\frac{n}{\delta}}{0.5n-1}}\right] \geq 1-\delta$$

The result follows from the definition of $\bar{\mu}_j$. □

A similar result can be established for the MMV:

Theorem 5.3.7. *Let $v \in [0,1]^m, \|v\|_1 = 1$ be a fixed weight vector, and $w \in [0,1]^m, \|w\|_1 = 1$ be a weight vector depending on a training set (X,Y) of size n drawn i.i.d. from a fixed distribution D. Then, for any $\delta > 0$:*

$$\Pr_{(X,Y)\sim D^n}\left[\gamma_w \leq \hat{\gamma}_w + \sqrt{\frac{D(R_w^2 \| R_v^2) + \ln\frac{1}{9\delta} + \ln n}{\frac{n}{18} - 1}}\right] \geq 1 - \delta$$

Proof. The proof follows along the same lines as the preceding one, except for the fact that γ_w needs to be decomposed into m^2 components. We define the $m \times m$ matrix $\widehat{\Gamma} = (\hat{\gamma}_{rs})_{1\leq r,s\leq m}$ with

$$\hat{\gamma}_{rs} := \frac{1}{n}\sum_{k=1}^{n} y_k x_k(r) - \frac{1}{n(n-1)}\sum_{k<l}^{n}\big(y_k x_k(r) - y_l x_l(r)\big)\big(y_k x_k(s) - y_l x_l(s)\big) \tag{5.19}$$

A simple calculation confirms that $\hat{\gamma}_w = w^T\widehat{\Gamma}w$, or, in terms of distributions $\hat{\gamma}_w = \mathbf{E}_{(r,s)\sim R_w^2}\,\hat{\gamma}_{rs}$. Similarly, $\gamma_{rs} := \mathbf{E}_{(X,Y)\sim D^n}\,\hat{\gamma}_{rs}$, so that $\gamma_w = \mathbf{E}_{(r,s)\sim R_w^2}\,\gamma_{rs}$. Let $\bar{\gamma}_{rs} := |\gamma_{rs} - \hat{\gamma}_{rs}|$ denote the difference between the expectation of the MMV restricted to feature r and s and the corresponding empirical estimate. This is a random variable depending on (X,Y). As a first step, we prove that

$$\Pr_{(X,Y)\sim D^n}\left[\mathbf{E}_{(r,s)\sim R_v^2} e^{(\frac{1}{18}n-1)\bar{\gamma}_{rs}^2} \leq \frac{n}{9\delta}\right] \geq 1-\delta \tag{5.20}$$

To prove this, we investigate changing one instance in (X,Y). The leftmost sum of (5.19) (the margin part) changes by at most $\frac{2}{n}$. The right summand (the variance part) changes by at most $\frac{4}{n}$. This can be seen as follows: Changing an instance affects $n-1$ summands of the form $(y_k x_k(r) - y_l x_l(r))(y_k x_k(s) - y_l x_l(s))$ in the rightmost sum of (5.19), and each summand changes by at most 4, because $\sup_{b,b'\in[-1,1]}[(a-b)^2 - (a-b')^2] \leq 4$ for $a \in [-1,1]$. Altogether, $\bar{\gamma}_{rs}$ changes by at most $\frac{6}{n}$. Thus, for arbitrary $r \in \mathbb{N}$, McDiarmid's inequality (Theorem 5.3.2) yields:

$$\Pr_{(X,Y)\sim D^n}\left[\bar{\gamma}_{rs} \geq x\right] \leq 2e^{-\frac{1}{18}nx^2} \tag{5.21}$$

Now, we follow the same procedure as in the proof for Theorem 5.3.6. We look for the density f which maximizes $\int_0^\infty e^{(\frac{1}{18}n-1)\bar{\gamma}_{rs}^2} f(\bar{\gamma}_{rs})\, d\bar{\gamma}_{rs}$, subject to the constraint (5.21) that $\int_x^\infty f(\bar{\gamma}_{rs})\, d\bar{\gamma}_{rs} \leq 2e^{-\frac{1}{18}nx^2}$. The maximum is achieved when $\int_x^\infty f(\bar{\gamma}_{rs})\, d\bar{\gamma}_{rs} = 2e^{-\frac{1}{18}nx^2}$. Taking the derivative yields that

$f_{max}(\bar{\gamma}_{rs}) = \frac{2}{9} n\bar{\gamma}_{rs} e^{-\frac{1}{18}n\bar{\gamma}_{rs}^2}$. Therefore,

$$\begin{aligned}\mathop{\mathbf{E}}_{(X,Y)\sim D^n} e^{(\frac{1}{18}n-1)\bar{\gamma}_{rs}^2} &\leq \int_0^\infty e^{(\frac{1}{18}n-1)\bar{\gamma}_{rs}^2} f_{max}(\bar{\gamma}_{rs})\ d\bar{\gamma}_{rs} \\ &= \int_0^\infty \frac{2}{9} n\bar{\gamma}_{rs} e^{(\frac{1}{18}n-1)\bar{\gamma}_{rs}^2} e^{-\frac{1}{18}n\bar{\gamma}_{rs}^2}\ d\bar{\gamma}_{rs} \\ &= \int_0^\infty \frac{2}{9} n\bar{\gamma}_{rs} e^{-\bar{\gamma}_{rs}^2}\ d\bar{\gamma}_{rs} \\ &= \frac{n}{9}\end{aligned}$$

Since this upper bound is valid for all indices r, s, it holds also for the expectation over those indices:

$$\mathop{\mathbf{E}}_{(X,Y)\sim D^n} \mathop{\mathbf{E}}_{(r,s)\sim R_v^2} e^{(\frac{1}{18}n-1)\bar{\gamma}_{rs}^2} \leq \frac{n}{9}$$

Inequality (5.20) follows from this and Markov's inequality. Applying Lemma 5.3.5 to (5.20) with $K = \frac{n}{9\delta}, R = R_w^2, S = R_v^2, x = (\bar{\gamma}_{11}, \ldots, \bar{\gamma}_{mm})^T, \beta = \frac{n}{18} - 1$ yields:

$$\mathop{\mathbf{Pr}}_{(X,Y)\sim D^n} \left[\sum_{r,s=1}^{n} w_r w_s \bar{\gamma}_{rs} \leq \sqrt{\frac{D(R_w^2 \| R_v^2) + \ln \frac{n}{9\delta}}{\frac{n}{18} - 1}} \right] \geq 1 - \delta$$

The result follows from the definition of $\bar{\gamma}_{rs}$. □

Unfortunately, the bound is not very tight. In the next section, we describe a novel bound that depends only on the rule repository size and is therefore better suited for our purposes.

5.3.3 Repository Size Bounds

In the following we derive two concentration inequalities that apply directly to the empirical margin and the MMV without the need to compute Rademacher complexities or the mutual entropy between weight vectors. Instead of modifying existing bounds for the empirical error (so that they apply to the empirical margins and MMV), we derive a new result using properties of the specific hypothesis class in our case, that is, the class of linear classifiers. The bounds turn out to be quite tight when compared to their Rademacher and PAC-Bayesian counterparts. The structural risk penalty depends only on the number of rules. This avoids costly calculations, but turns out to still yield good risk estimates in empirical experiments. In order to prove the actual concentration inequalities, we need the following lemma:

Lemma 5.3.8 (Hoeffding's inequality (Hoeffding, 1963))**.** *Let X be a random variable with $\mathbf{E}\, X = 0, a \leq X \leq b$. Then, for $s > 0$,*

$$\mathbf{E}\left[e^{sX}\right] \leq e^{s^2(b-a)^2/8}$$

For the preceding concentration results we only required that $w \in [-1,1]^m$. For the actual optimization problems in Section 5.2.2 and 5.2.3, however, we had stricter requirements. If one denotes the unit-ball in $\mathbb{R}^m$ according to the p-norm by $\mathcal{B}_p^m := \{w \in \mathbb{R}^m \,|\, \|w\|_p \leq 1\}$, we had constraints to ensure $w \in \mathcal{B}_2^m$ for empirical margin maximization and $w \in \mathcal{B}_1^m$ for MMV optimization. It turns out that one can make use of these stricter requirements to improve the concentration properties. In the case of empirical margins, we give the following result, which relates the size of the weight vector space to the instance space. The special case of regularization with the two-norm can then be derived as a corollary.

Theorem 5.3.9. *Let (X, Y) be the application of the rules to a training set of size $n \geq 3$ drawn i.i.d. from D. Let $p \geq 1$ and $q = \frac{p}{p-1}$ (so that $q = \infty$, if $p = 1$). If D is such that for all $(x, y) \sim D : x \in \mathcal{B}_q^m$, then, for any linear classifier $w \in \mathcal{B}_p^m$ and $\delta > 0$:*

$$\mathbf{Pr}\left[\mu_w \geq \hat{\mu}_w - \sqrt{\frac{2}{n} m^{\frac{2}{q}} \ln \frac{2m}{\delta}}\right] \geq 1 - \delta$$

Proof. Let $X \in [-1,1]^{n \times m}$ denote the training matrix and $Y \in [-1,1]^n$ the class label vector. Then, the empirical margin can be conveniently written as a bilinear form $\hat{\mu}_w = \frac{1}{n} Y^T X w$. Correspondingly, the true margin is simply $\mu_w = \mu^T w$, where $\mu := \mathbf{E}_{(x,y)\sim D}\, yx$ is the expected mean instance vector, pre-multiplied with the class. With this, one can write the maximal difference between empirical and true margin $\bar{\mu}$ as the supremum over a dot product with the weight vector w:

$$\begin{aligned} \bar{\mu} &:= \sup_{w \in \mathcal{B}_p^m} |\hat{\mu}_w - \mu_w| \\ &= \sup_{w \in \mathcal{B}_p^m} \left| \left(\frac{1}{n} X^T Y - \mu\right)^T w \right| \end{aligned}$$

We are interested in upper-bounding the probability that $\bar{\mu}$ exceeds a threshold ϵ:

$$\begin{aligned}
\mathbf{Pr}\left[\bar{\mu} \geq \epsilon\right] &= \mathbf{Pr}\left[e^{s\frac{1}{2}\bar{\mu}} \geq e^{s\frac{1}{2}\epsilon}\right] && (5.22)\\
&\leq e^{-\frac{1}{2}s\epsilon}\,\mathbf{E}\left[\exp(s\frac{1}{2}\bar{\mu})\right] && (5.23)\\
&= e^{-\frac{1}{2}s\epsilon}\,\mathbf{E}\left[\exp\left[s\sup_{w\in\mathcal{B}_p^m}\left|(\frac{1}{2n}X^TY-\frac{1}{2}\mu)^T w\right|\right]\right]\\
&\leq e^{-\frac{1}{2}s\epsilon}\,\mathbf{E}\left[\exp\left[s\left\|\frac{1}{2n}X^TY-\frac{1}{2}\mu\right\|_q\Big(\sup_{w\in\mathcal{B}_p^m}\|w\|_p\Big)\right]\right] && (5.24)\\
&\leq e^{-\frac{1}{2}s\epsilon}\,\mathbf{E}\left[\exp\left[s\left\|\frac{1}{2n}X^TY-\frac{1}{2}\mu\right\|_q\right]\right] && (5.25)
\end{aligned}$$

In (5.22) and (5.23) we perform Chernoff's bounding method: First, we introduce an (arbitrary) parameter $s > 0$, which can be adjusted to fine-tune the bound later on. Then, we apply Markov's inequality. Since $\frac{1}{p}+\frac{1}{q}=1$ we can apply Hölder's inequality to the dot product in (5.23), so that the supremum in (5.24) is only over the p-norm of w. Since w is taken from $\mathcal{B}_p^m$, the supremum in (5.25) is at most one and can be omitted.

Since the average over a vector's components is smaller than the vector's largest component, we have for any $x \in \mathbb{R}^m$ and any $q \geq 1$: $\frac{1}{m}\sum_{i=1}^m |x_i|^q \leq \sup_{1\leq i\leq m}|x_i|^q$ and therefore $(\frac{1}{m}\sum_{i=1}^m |x_i|^q)^{\frac{1}{q}} \leq \sup_{1\leq i\leq m}|x_i|$, or, in terms of norms $\|x\|_q \leq m^{\frac{1}{q}}\|x\|_\infty$. Therefore, where $[x]_i$ denotes the ith component of vector x:

$$\begin{aligned}
\mathbf{Pr}\left[\bar{\mu} \geq \epsilon\right] &\leq e^{-\frac{1}{2}s\epsilon}\left[\mathbf{E}\sup_{1\leq i\leq m}\exp\left|sm^{\frac{1}{q}}\left[\frac{1}{2n}Y^TX-\frac{1}{2}\mu\right]_i\right|\right]\\
&\leq e^{-\frac{1}{2}s\epsilon}\left[\mathbf{E}\sum_{i=1}^m\exp\left|sm^{\frac{1}{q}}\left[\frac{1}{2n}Y^TX-\frac{1}{2}\mu\right]_i\right|\right]\\
&= e^{-\frac{1}{2}s\epsilon}\left[\sum_{i=1}^m\mathbf{E}\prod_{j=1}^n\exp\left|sm^{\frac{1}{q}}\left[\frac{1}{2n}y_jx_j-\frac{1}{2}\mu\right]_i\right|\right]\\
&= e^{-\frac{1}{2}s\epsilon}\left[\sum_{i=1}^m\prod_{j=1}^n\mathbf{E}\exp s\left|m^{\frac{1}{q}}\left[\frac{1}{2n}y_jx_j-\frac{1}{2}\mu\right]_i\right|\right] && (5.26)
\end{aligned}$$

(5.26) holds, because the expectation of the product of independent random variables equals the product of expectations. Define $d_{ij} := m^{\frac{1}{q}}[\frac{1}{2n}y_jx_j-\frac{1}{2}\mu]_i$. We would like to upper-bound $\exp(s|d_{ij}|)$. We can not apply Hoeffding's inequality directly, because $\mathbf{E}[|d_{ij}|] \neq 0$. However, when writing $\exp(s|d_{ij}|) \leq \exp(sd_{ij}) + \exp(-sd_{ij})$, Hoeffding's lemma is applicable to the two summands. The d_{ij} can take values in the interval between $-\frac{1}{2n}m^{\frac{1}{q}} - \frac{1}{2}\mu_i$ and

$\frac{1}{2n}m^{\frac{1}{q}} - \frac{1}{2}\mu_i$. Since the size of this interval is $\frac{1}{n}m^{\frac{1}{q}}$, we get:

$$\begin{aligned}\mathbf{Pr}\left[\bar{\mu} \geq \epsilon\right] &\leq e^{-\frac{1}{2}s\epsilon}\left[\sum_{i=1}^{m}\prod_{j=1}^{n} 2\exp\left[\frac{1}{8}s^2 m^{\frac{2}{q}}\frac{1}{n^2}\right]\right] \\ &= e^{-\frac{1}{2}s\epsilon} 2m\exp\left[\frac{1}{8}s^2 m^{\frac{2}{q}}\frac{1}{n}\right]\end{aligned}$$

Setting $s = 2n\epsilon m^{-\frac{2}{q}}$ yields:

$$\mathbf{Pr}\left[\bar{\mu} \geq \epsilon\right] \leq 2m\exp\left[-\frac{1}{2}n\epsilon^2 m^{-\frac{2}{q}}\right]$$

The result follows by setting $\epsilon = \sqrt{\frac{2}{n}m^{\frac{2}{q}}\ln\frac{2m}{\delta}}$. □

It is interesting to see how the regularization of the weight vector influences the bound. If $w \in \mathcal{B}_1^m$, $q = \infty$ and the penalty grows only logarithmically with $\sqrt{\ln 2m}$. If, on the other hand, $w \in \mathcal{B}_2^m$, the penalty contains an additional factor of $\sqrt{m}$. This is not very suprising. In the worst case, if $w \in \mathcal{B}_\infty^m$ the bound needs to hold for the case where each weight is set to one. If the individual attributes are independent, the variances of each attribute add up, so that $Var(w^T\mu) = \sum_{i=1}^m Var(\mu_i)$. To accommodate for this m-fold uncertainty, the penalty contains the additional factor m. If, on the other hand, the weights add to one, the variance of $w^T\mu$ is determined essentially by the component with the largest variance.

For the MMV we give a similar result. We restrict ourselves to the case where $w \in \mathcal{B}_1^m$.

Theorem 5.3.10. *Let D be a fixed unknown distribution over $\mathcal{B}_\infty^m \times \{-1, +1\}$ and $(X, Y) = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ be a sample of size n drawn i.i.d from D. Then for all $\delta > 0$ and all $w \in \mathcal{B}_1^m$:*

$$\underset{(X,Y)\sim D^n}{\mathbf{Pr}}\left[\gamma_w \geq \hat{\gamma}_w - \sqrt{18\frac{2\ln m + \ln\frac{1}{\delta}}{n}}\right] \geq 1 - \delta \qquad (5.27)$$

Proof. We assume that the training data is given as a $n \times m$ matrix $X \in \mathbb{R}^{n\times m}$. Here, each of the n row vectors is an instance $x_i := (x_{i1}, \ldots, x_{im})$ and each of the m columns represents a rule. When given a weight vector $w \in [-1; 1]^m$, we are interested in the distribution of the empirical margins $y_i x_i w$. If a training set contains a negative instance (that is, $y_i = -1$) one can simple multiply y_i and x_i with -1 to gain an equivalent positive training instance that gives rise to the same margin. Therefore, we assume without loss of generality that $y_i = 1$ for all $1 \leq i \leq n$ and omit the y_i in the following. One can then conveniently represent the empirical margin $\hat{\mu}_w$ of w with regard to X as the quadratic form $\hat{\mu}_w := \frac{1}{n}\mathbf{1}_n X w$, where $\mathbf{1}_n$ denotes the vector containing n ones. To compute the MMV, we need the (unweighted) empirical mean vector $\hat{\mu} \in \mathbb{R}^n := \frac{1}{n} X^T \mathbf{1}_n$ and the (unweighted) empirical covariance matrix $\hat{\Sigma} \in \mathbb{R}^{m \times m} := \frac{1}{n-1}(X^T X - \frac{1}{n} X^T \mathbf{1}_n \mathbf{1}_n^T X)$. Later on, we will find it convenient to compute the individual entries $\hat{\sigma}_{ij}$ in $\hat{\Sigma}$ by

$$\hat{\sigma}_{ij} = \frac{1}{n(n-1)} \sum_{k<l} (x_{ki} - x_{li})(x_{kj} - x_{lj}) \tag{5.28}$$

The corresponding true quantities are $\mu := \mathbf{E}\,\hat{\mu}$, $\Sigma := \mathbf{E}\,\hat{\Sigma}$ and $\sigma_{ij} := \mathbf{E}\,\hat{\sigma}_{ij}$. With these tools, the empirical MMV $\hat{\gamma}_w$ can be defined as $\hat{\gamma}_w := \hat{\mu}^T w - w^T \hat{\Sigma} w$, and the true MMV is just the expectation over all training sets $\gamma_w := \mathbf{E}_{X \sim D^n}\, \hat{\gamma}_w$. We would like to find an upper bound of the probability $\mathbf{Pr}[\sup_{w \in \mathcal{B}_1^m} (\hat{\gamma}_w - \gamma_w) \geq \varepsilon]$ that the empirical MMV differs from its expectation by more then ε for the worst possible weight vector.

First of all, observe that

$$\begin{aligned}
\sup_{w \in \mathcal{B}_1^m} \Big(\hat{\gamma}_w - \gamma_w\Big) &= \sup_{w \in \mathcal{B}_1^m} \Big((\hat{\mu} - \mu)^T w - w^T (\hat{\Sigma} - \Sigma) w\Big) \\
&= \sup_{w \in \mathcal{B}_1^m} \Big(\sum_{k=1}^m w_k (\hat{\mu}_k - \mu_k) - \sum_{k,l=1}^m w_k w_l (\hat{\sigma}_{kl} - \sigma_{kl})\Big) \\
&\leq \sup_{w \in \mathcal{B}_1^m} \Big(\sum_{k,l=1}^m w_k w_l \big[(\hat{\mu}_k - \mu_k) - (\hat{\sigma}_{kl} - \sigma_{kl})\big]\Big) \qquad (5.29) \\
&\leq \sup_{v \in [-1;1]^{m \times m}} \Big(\sum_{k,l=1}^m v_{kl} \big[(\hat{\mu}_k - \mu_k) - (\hat{\sigma}_{kl} - \sigma_{kl})\big]\Big) \qquad (5.30) \\
&\leq \sup_{k,l} \Big((\hat{\mu}_k - \mu_k) - (\hat{\sigma}_{kl} - \sigma_{kl})\Big) \qquad (5.31)
\end{aligned}$$

Inequality (5.29) holds because the w_l sum up to at most one, (5.30) is a relaxation of the condition in the supremum, and (5.31) is due to the fact that a linear function on a convex hull reaches its optimum at one of the vertices.

Then, we can upper-bound the structural risk as follows, where $s > 0$ is a free parameter that can be tuned to make the inequality sharper later on:

$$\begin{aligned}
\mathbf{Pr}[\sup_{w \in \mathcal{B}_1^m} (\hat{\gamma}_w - \gamma_w) \geq \varepsilon] &= \mathbf{Pr}[\sup_{w \in \mathcal{B}_1^m} e^{s(\hat{\gamma}_w - \gamma_w)} \geq e^{s\varepsilon}] \\
&\leq e^{-s\varepsilon}\, \mathbf{E}\Big[\sup_{w \in \mathcal{B}_1^m} e^{s(\hat{\gamma}_w - \gamma_w)}\Big] \qquad (5.32) \\
&\leq e^{-s\varepsilon}\, \mathbf{E}\Big[\sup_{k,l} e^{s[(\hat{\mu}_k - \mu_k) - (\hat{\sigma}_{kl} - \sigma_{kl})]}\Big] \qquad (5.33) \\
&\leq e^{-s\varepsilon} \sum_{k,l=1}^m \mathbf{E}\Big[e^{s[(\hat{\mu}_k - \mu_k) - (\hat{\sigma}_{kl} - \sigma_{kl})]}\Big] \qquad (5.34)
\end{aligned}$$

Inequality (5.32) is an application of Markov's inequality, (5.33) is given above, and (5.34) follows because $e^x > 0$ and the sum over all k, l includes the summand that achieves the supremum.

Now, define the set of random variables

$$Z^{(k,l)} := (\hat{\mu}_k - \mu_k) - (\hat{\sigma}_{kl} - \sigma_{kl})$$

depending on X. If we can find an upper bound for the $Z^{(k,l)}$, (5.34) gives us an upper bound of the structural risk. We follow the method by McDiarmid (1989) and write $Z^{(k,l)}$ as a sum of martingale differences to find such a bound. Let

$$V_r^{(k,l)} := \mathbf{E}\left[Z^{(k,l)} \mid x_1, \ldots, x_r\right] - \mathbf{E}\left[Z^{(k,l)} \mid x_1, \ldots, x_{r-1}\right]$$

Then, $Z^{(k,l)} = \sum_{r=1}^n V_r^{(k,l)}$. Replacing the $\hat{\sigma}_{ij}$ in $V_r^{(k,l)}$ with their definition (5.28) and $\hat{\mu}_i$ with $\frac{1}{n}\sum_j x_{ji} w_i$ yields:

$$V_r^{(k,l)} = \frac{1}{n}(x_{rk} - \mathbf{E}\, x_{rk}) - \tag{5.35}$$

$$\frac{1}{n(n-1)} \sum_{j \neq r} \left[\mathbf{E}\left[\hat{\sigma}_{rj}^{kl} \mid x_1, \ldots, x_r\right] - \mathbf{E}\left[\hat{\sigma}_{rj}^{kl} \mid x_1, \ldots, x_{r-1}\right] \right] \tag{5.36}$$

where $\hat{\sigma}_{ij}^{kl} := (x_{ik} - x_{jk})(x_{il} - x_{jl})$. It turns out that the random variable $V_r^{(k,l)}$ can take only values in the interval $[c; c + \frac{6}{n}]$ for some constant c. The first part in (5.35) depends only on $x_{rk} \in [-1; 1]$, so it can differ from its expectation by at most $\frac{1}{n}$. In the second part (5.36) each summand depends only on x_{rk} and x_{rl}, so that the $\hat{\sigma}_{ij}^{kl}$ can take values that differ by at most four. Since there are $n - 1$ summands, the part in (5.36) changes within a range of at most $\frac{4}{n}$ depending on X. Also, since the conditional expectation $\mathbf{E}[V_r^{(k,l)} \mid x_1, \ldots, x_{r-1}] = 0$, we can apply a conditional version of Hoeffding's inequality (Lemma 5.3.8). Applying this inequality conditionally to the $V_r^{(k,l)}$, we obtain:

$$\mathbf{E}\left[e^{sV_r^{(k,l)}} \mid x_1, \ldots, x_{r-1}\right] \leq e^{\frac{1}{8}s^2(\frac{6}{n})^2} \tag{5.37}$$

Since the instances are drawn independently from each other, we can apply (5.37) iteratively on all summands in $Z^{(k,l)} = \sum_{r=1}^n V_r^{(k,l)}$:

$$\begin{aligned}
\mathbf{E}\left[e^{sZ^{(k,l)}}\right] &= \mathbf{E}\left[e^{s\sum_{i=1}^{n-1} V_i^{(k,l)}} \cdot e^{sV_n^{(k,l)}}\right] \\
&= \mathbf{E}\left[e^{s\sum_{i=1}^{n-1} V_i^{(k,l)}} \cdot \mathbf{E}\left[e^{sV_n^{(k,l)}} \mid x_1, \ldots, x_{n-1}\right]\right] \\
&\leq \mathbf{E}\left[e^{s\sum_{i=1}^{n-1} V_i^{(k,l)}} \cdot e^{\frac{1}{8}s^2(\frac{6}{n})^2}\right] \\
&\leq \ldots \leq e^{n\frac{1}{8}s^2(\frac{6}{n})^2} \\
&= e^{\frac{9}{2}s^2\frac{1}{n}}
\end{aligned}$$

Finally, we may plug this result into (5.34):

$$\begin{aligned}\mathbf{Pr}\left[\sup_{w\in\mathcal{B}_1^m}(\hat{\gamma}_w-\gamma_w)\geq\varepsilon\right] &\leq e^{-s\varepsilon}m^2e^{\frac{9}{2}s^2\frac{1}{n}}\\ &\leq m^2e^{-\frac{1}{9}\varepsilon^2n+\frac{1}{18}\varepsilon^2n}\quad\text{(by choosing } s=\frac{1}{9}\varepsilon n)\\ &= m^2e^{-\frac{1}{18}\varepsilon^2n}\end{aligned}$$

Choosing

$$\varepsilon=\sqrt{\frac{18(2\ln m+\ln\frac{1}{\delta})}{n}}$$

yields

$$\mathbf{Pr}\left[\sup_{w\in\mathcal{B}_1^m}(\hat{\gamma}_w-\gamma_w)\geq\sqrt{\frac{18(2\ln m+\ln\frac{1}{\delta})}{n}}\right]\leq\delta$$

This is equivalent to the statement in Theorem 5.3.10. □

The two inequalities basically state that the true margin or MMV is with high probability larger than its empirical counterpart minus a risk term that depends logarithmically on the size of the class of rules. A straightforward application of lemmas 5.2.1 and 5.2.2 also give bounds on the true error. The inequalities provide a pessimistic estimate of the structural risk that is involved with using larger classes of rules. The constants in the the two risk terms are too pessimistic for most real world distributions and applicable only for worst-case analysis; for practical model selection applications one should use smaller constants.

5.4 The Rumble System

In Section 5.2 we described three optimization criteria to perform empirical risk minimization on weighted rule sets. These criteria allow for an efficient way to identify good weight vectors for a given repository of rules. In Section 5.3 we dealt with the question of how large the rule repository should be in order to minimize the structural risk. In this section we describe how these building blocks are put together to form the statistically motivated Rumble (Rule and Margin Based Learner) system.

Rumble's main design is inspired by a simple application of structural risk minimization. An outline is given in Algorithm 8. The system starts with the training set given in a set X and an empty set of rules. In the main loop, it adds a new rule to the set of rules and applies the rules to the examples in the training set. It determines the weight vector $p^{(i)}$ optimizing

Algorithm 8 The basic RUMBLE rule learning algorithm.

```
procedure RUMBLE($X$)
    $R^{(0)} \leftarrow \emptyset$
    for $i = 1, 2, \ldots$ and while new rules available do
        $R^{(i)} \leftarrow$ add new rule to $R^{(i-1)}$
        $T^{(i)} \leftarrow$ apply rules in $R^{(i)}$ to instances in $X$
        $p^{(i)} \leftarrow \mathrm{argmax}_p\, \hat{\gamma}_p(T^{(i)})$
        $\gamma^{(i)} \leftarrow$ bound calculated from $p^{(i)}$ and $|R^{(i)}|$
    end for
    return $(R^{(i)}, p^{(i)})$ with the maximal $\gamma^{(i)}$
end procedure
```

the soft margin loss, the empirical margin or the empirical MMV on this data and calculates an upper bound on the corresponding true values given the empirical quantity and the number of rules. If there are no new rules available, the algorithm terminates the loop and returns the set of rules and the weight vector which achieved the best bound. To complete the system, one needs two additional subroutines: one to solve the empirical optimization problems and another to generate new rules for the training set.

The optimization tasks for empirical risk minimization are described in Section 5.2. For MMV optimization, the problem is essentially a semidefinite quadratic programming problem with one equality and $2n$ inequality constraints. It can be solved using any established algorithm for constrained optimization. Usually, the computationally most demanding part of the process is to determine the set of active constraints. It is a good idea to keep the list of active constraints even during the main loop. In this way one can use the active constraints for iteration i as a starting point for the optimization problem in iteration $i + 1$. If the active set is known, the optimization procedure boils down to solving a linear equation system. This system features one equation for every weight set to a non-zero value and can be solved in quadratic time. As the number of non-zero weights is usually rather small, the optimization step is typically very fast.

In principle, one can use any arbitrary method to generate new rules. Ideally, one would like to generate only rules, that are informative for the prediction task at hand. However, selecting the rules depending on the class labels in the training set can easily lead to overfitting.[3] It is a better idea to select the rules based on information about the structure of the instances without the class labels or on other background knowledge. For example, one could use information obtained in a previous clustering step to generate

[3]In the extreme case one could generate just one rule that perfectly explains the training set. Of course, this would render the bound useless and lead to overfitting.

rules that describe certain characteristics of the clusters in the data. It is not clear how to devise a general scheme that generates "good" rules for all kinds of data sets. Instead, we found it more sensible to keep the rule generation process as generic and flexible as possible. In this way the user can adjust the rule generation depending on her knowledge of the particular domain. In the following we will present a rule generation strategy that resembles the approach taken in traditional rule learning systems. It appears to work well on typical attribute-value data sets as those taken from the UCI repository and is easy to adjust to more complex data, for example in multi-relational settings.

We start by generating rules that describe the original features found in the data set. For nominal-valued feature we generate one rule of the form "feature=value" for each possible value. We use a simple frequency based discretization step to split continuous attributes into four bins and generate one rule per bin. Each rule outputs +1, if its condition is met, -1 if it is not, and 0, if the value is missing. This gives a certain base set of rules encoding most information in the training sample. In a second step we apply refinement operators on the existing rules to generate more complex rules. One straight-forward approach is to combine two rules by calculating the maximum or minimum of two base conditions. This resembles the logical "and" and "or" used in traditional rule learning systems. The weighted sets of rules representation seems to be a relaxed form of Boolean conjunction, because optimizing for a large margin means that positive examples are satisfied by a large majority of the rules. This suggests using the "or" (that is, maximum) operation for the rules to generate a kind of relaxed *conjunctive normal form* (CNF). However, the representation also allows for negative weights, that is, negated rules. For negated rules, the "and" (that is, minimum) operator appears to be the better choice. In effect, using the minimum instead of the maximum operator is equivalent to swapping the target label.

The repository size bounds used for capacity control depend only on the number of rules that might be used to generate a weighted rule set, not on any structural properties of the rules. While it is tempting to provide a fixed default order of rule classes, best results can be achieved if the classes are selected in an application-specific manner. For example, if the user has some information on which kind of rules might be more informative than others, she should start with a rule class containing the informative rules and add the presumably non-informative rules only later on. This is an easy, yet flexible and powerful way to adjust the learner's bias and incorporate background knowledge, without the need to adjust parameters or kernels. The next section contains a small empirical study on how to adjust the rule classes in order to boost predictive accuracy.

5.5 Experiments

In order to test the performance of Rumble on real world data, we implemented a version in C++ on Linux. In practice, the overhead of optimizing for rules with very low weights is not necessary, as their weight does not contribute in a significant way. Thus, whenever the number of generated rules exceeds one hundred, we ignore the rule with the lowest weight assigned during the optimization step (but not during the rule refinement step). We apply the system to all two-class data sets from Frank and Witten (1998). We select the rule generation bias outlined above: nominal attributes are added as one "feature=value" rule for each feature value, continuous attributes are discretized into four bins. After those base rules, we build all combinations of two rules and form new rules by combining the conditions in each rule pair with the maximum operator. We use the model selection procedure using Theorem 5.3.9 for the 1-norm SVM and the empirical margin maximization, and Theorem 5.3.10 for MMV optimization. We set the risk term constants to 0.1 for the 1-norm SVM and empirical margin, and to 1.0 for MMV to avoid overly pessimistic model selection. Also, for the empirical margin optimization, we adjusted the training sets in a preprocessing step to contain equally many positive and negative instances. Otherwise, it would have performed worse on data sets with skewed class distribution.

Table 5.1 gives the predictive accuracy of the induced rule sets as estimated by tenfold cross-validation for the proposed system in comparison to PART (Frank and Witten, 1998) and SLIPPER (Cohen and Singer, 1999). We also include the results for the benchmark learner SL^2 presented in Section 4.3.1 and the rule ensemble described in Section 4.4.2. The table indicates that MMV performs better than the 1-norm SVM in 14 of the 18 cases and better than empirical margin optimization in 15 cases. To assess the statistical significance of this result we use a Wilcoxon signed ranks test. The test yields a chance probability of less than 0.02 in both comparisons. Thus, we can be very certain that MMV outperforms the 1-norm SVM and empirical margin optimization on similar data sets. MMV is also better than PART and SLIPPER in 10 of the 18 cases. The corresponding chance probabilities are 0.71 and 0.81. Even with this rather limited rule generation bias, MMV optimization appears to be competitive with existing rule learning algorithms. It outperforms SL^2 in 12 of the 18 cases, but is better than the rule ensemble on only 8 data sets. Thus, it is safe to say that MMV's bias works better than SL^2's bias towards small DNF sets on those data sets. While the rule ensemble appears to be slightly better when averaged over all data sets, the differences for individual data sets can be large. For instance, MMV optimization outperforms the rule ensemble on the heart-c data set by a margin of over 9 percent, while the rule ensemble's accuracy on the labor data set exceeds the one of MMV by over 12 percent. It seems that the biases of Rumble and the rule ensemble work differently well on

Data Set	PART	SLIP-PER	SL^2	Rule Ens.	1-norm SVM	Emp. margin	MMV
australian	84.3	85.1	85.4	86.1	85.5	80.9	85.5
breast-cancer	69.6	71.4	70.1	73.8	70.3	52.8	69.2
breast-w	94.9	96.0	94.1	96.6	90.8	96.6	94.0
colic	84.4	84.6	81.5	82.9	81.5	80.7	82.9
diabetes	74.0	73.5	72.1	73.9	74.2	65.0	75.8
german	70.0	71.8	70.0	73.6	70.0	61.1	74.4
glass(G2)	80.0	83.1	71.6	73.2	61.3	68.1	80.4
heart-c	78.5	79.3	72.9	78.5	75.9	79.5	85.1
heart-h	80.5	79.2	80.5	79.8	78.6	78.9	81.6
heart-statlog	78.9	78.5	72.1	77.1	75.2	78.5	83.3
hepatitis	80.2	79.5	81.3	83.9	79.4	59.4	84.5
ionosphere	90.6	93.2	91.1	91.8	78.6	78.1	86.9
kr-vs-kp	99.3	59.4	98.1	97.8	68.3	77.7	91.2
labor	77.3	92.3	94.2	93.3	75.4	84.2	80.7
mushroom	100.0	100.0	100.0	100.0	88.7	97.3	91.8
sick	98.6	98.5	93.8	98.1	93.9	66.0	93.9
sonar	76.5	74.2	63.6	76.2	56.7	69.2	77.4
vote	95.9	94.1	94.3	95.9	95.6	89.2	95.6

Table 5.1: Results: percentage of correct classifications.

different data sets.

Table 5.2 gives the number of rules that were generated by PART, SLIPPER, and MMV. For reference, we also give the average number of rules induced by SL^2 as specified in Table 4.3 in Section 4.3.2. MMV generated the most compact rule set in 7 cases, PART in 6, and SLIPPER in 7 cases, so there is hardly a significant win for any of the three systems. However, in contrast to PART, MMV never generated an excessively large rule set (more than 30 rules). Also, unlike PART and SLIPPER, each of MMV's rules contains at most two literals, so the individual rules are much more compact than those of the other systems. While regularization with the 1-norm in MMV optimization induces sparse weight vectors, its bias towards small rule sets is not as strong as the one of SL^2. In a sense, RUMBLE appears to offer a viable compromise between the comprehensibility of SL^2's rule sets and the predictive accuracy of rule sets induced by the rule ensemble.

One additional benefits of the flexible rule generation process, though, is the fact that the user can adjust the learner's bias by rearranging the order in which the rules are added or by including new rule conditions. The order matters, because the risk of overfitting increases with the number of generated rules. Hence, the structural risk term in (5.27) penalizes rules

Data Set	SL^2	PART	SLIPPER	MMV
australian	1.01	32	9	15
breastcancer	1.76	20	2	20
breastw	1.31	10	15	8
colic	1.08	9	2	15
diabetes	1.14	13	21	17
german	1.46	78	24	17
glassg2	1.59	7	30	17
heartc	1.29	25	19	22
hearth	1.13	10	8	10
heartstatlog	1.19	24	30	25
hepatitis	1.09	8	3	10
ionosphere	1.27	10	19	10
krvskp	8.91	23	35	6
labor	1.11	3	3	12
mushroom	3.07	13	10	5
sick	1.11	20	22	1
sonar	1.4	8	26	22
vote	1.46	7	4	2

Table 5.2: Results: number of induced rules.

that are added only very late. To avoid the pruning of good rules, the user should configure the system to evaluate the (presumedly) more informative rules first. We investigate this principle in three cases.

Consider the mushroom data set. Rules with just two conditions do not seem to work very well on it. Taking a closer look at the data set, it appears that the attributes are not isolated, but come in groups depending on what part of the mushroom they describe. For instance, there are three attributes describing the cap (shape, surface and color), four about the gills (attachment, spacing, size, color) and six specifying the stalk (shape, root, surface above ring, surface below ring, color above ring and color below ring). In order to find rules that specify a certain appearance of a specific part of the mushroom, we split the attributes into the segments cap, odor, gill, stalk, veil, ring, and occurrence. For each segment we form all disjunctions of three base features and add those as rules. As the rule generation process in our implementation is controlled by a set of simple declarative statements, this bias adjustment does not require the modification of any code. Running the system with this new bias, we yield a predictive accuracy of 98.5% according to tenfold cross-validation, a 6.7% increase over the default bias.

When looking at the labor data set, one immediately notices that there are lots of missing values. The instances in the data set represent contracts and it seems to be sensible that the inclusion or omission of a certain attribute value is on purpose rather than a random phenomenon. Consequently, it might make sense to include new rules that are satisfied only if a specific attribute value is missing. Also, some of the features are obviously correlated (e.g. wage-increase-first-year and wage-increase-second-year). To avoid combinations of attributes with similar informative value, we calculate the mutual information of all pairs of base features and add the 200 disjunctions with the smallest mutual information as rules. This two modifications boost the estimated preditive accuracy from 80.7% to 91.2%, an increase of over ten percent.

Finally, we investigate the mutagenesis data set (Helma *et al.*, 2004). This data set contains 684 instances, each of which represents the molecular structure of a compounds. The goal is to predict whether or not a specific compound is mutagenic. We represent each molecule as a graph, where the nodes specify atoms and the edges are bonds. With this representation one can form rules that are satisfied, if a specific substructure (that is, a subgraph) is present in the molecule. Of course, one would like to have rules about substructures that occur with a minimal frequency in the data set, otherwise the rules would identify individual instances and the learner would certainly overfit. We run the substructure mining system FreeTreeMiner (Rückert and Kramer, 2004a) with a minimum threshold of 2% to find the frequently occurring acyclic subgraphs of the molecules in the data set. This yields 3940 frequent structures, most of them containing long chains of carbon atoms A straightforward approach is to generate the 3940 rules – one for each substructure – in the same order as output by FreeTreeMiner. Tenfold cross validation with this simple bias yields an accuracy estimate of 68.4%. In order to improve this score one might want to include some background knowledge. For example, it is well known that halogen compounds are often mutagenic. Thus, we generate rules containing bromine or chlorine atoms first. Also, the presence or absence of aldehyde, alcoholic or carboxylic groups might indicate certain functional behavior; thus, we generate rules containing oxygen atoms after the halogen rules. Running the MMV system on this slightly reordered rule set yields an accuracy estimate of 70.9%. To further improve this score, we order the substructures according to their size, because larger substructures are usually rather specific and more likely to play a role in overfitting. Also, a occurrence of two (unconnected) small substructures might be more predictive than the existence of one large group. Therefore we first generate all rules containing up to two atoms, and then build disjunctive rules testing the occurrence of two small fragments before continuing with larger substructures. This improves the score slightly to 73.1%, while retaining a small and informative rule set containing only 19 rules. Further experiments on multi-relational data can be found in the next chapter.

5.6 Summary and Related Work

In this chapter, we discussed empirical risk minimization and capacity control for weighted rule sets. We started with three different criteria for margin-based risk minimization. The 1-norm SVM has been introduced by Bradley and Mangasarian (1998) and is investigated in more detail by Zhu *et al.* (2004). The feature-selection bias of regularization based on the 1-norm is well known and used for instance in the lasso penalty, see Section 3.4.3. Maximizing the empirical margin directly can be seen as a reformulation of empirical risk minimization with a *linear loss* $l_l(c, x, y) := y(1 - c(x))$, which dominates the zero-one loss for $x \in [-1, 1]$. In a sense, the linear loss is an extreme version of the hinge loss, because it optimizes the margin for every instance. Thus, empirical margin maximization can be seen as an SVM, which treats all instances as support vectors, see Cristianini and Shawe-Taylor (2000) for a discussion of SVMs. Finally, we introduced Margin Minus Variance minimization, a new optimization criterion aiming at classifiers that agree with most instances (large margin) and have only a few exceptions (low variance). MMV is similar to logistic regression (Hosmer and Lemeshow, 2000) and Fisher's linear discriminant (Fisher, 1936) (see Hastie *et al.*, 2001, for a concise discussion) in that it uses the variance part. However, MMV differs in that it uses the total variance information instead of the within-class variance.

In Section 5.3 we gave bounds for the true margin and MMV depending on their empirical counterparts and some penalty terms depending on the hypothesis class complexity. We provide three different ways to measure the hypothesis class complexity. For Rademacher penalties, we use the empirical margins or second moments on randomized versions of the training data. Rademacher penalties for model selection have been studied by Bartlett *et al.* (2002). On the theoretical side, localized versions of Rademacher complexities (Bartlett *et al.*, 2005) can be shown to be tighter than their global counterparts. There are still some interesting open questions regarding Rademacher bounds for margin based quantities (Wang and Feng, 2007). In the PAC-Bayesian framework, the original bounds apply to the Gibbs and voting classifiers and depend on the availability of a good "prior" distribution. There are many different ways to adapt the PAC-Bayesian bound to margin-based classification, see for instance Herbrich and Graepel (2002) and the tutorial by Langford (2005). More recently, "Occam's Hammer" (Blanchard and Fleuret, 2007) provides a pointwise version of the PAC-Bayes bound, which applies to individual classifiers instead of the combined Gibbs or voting classifiers. Finally, the repository size bounds depend only on the number of rules in the rule repository. It is noteworthy that the penalties in the bounds are $O(\sqrt{\log m})$, that is, logarithmic in the rule repository size, whereas a standard VC-dimension analysis yields a $O(\sqrt{m})$ penalty.

In the final sections, we combine margin-based empirical risk minimization with capacity control based on the repository size bounds. An empirical investigation of the resulting Rumble system (also published in Rückert and Kramer, 2006b; Rückert and Kramer, 2007a) on UCI data sets (Asuncion and Newman, 2007) indicates that MMV works best. Margin-based rule learning has been employed by SLIPPER (Cohen and Singer, 1999) and RuleFit (Friedman and Popescu, 2005). Similar to Rumble, the set covering and decision list machines (Marchand and Taylor, 2003; Marchand and Shawe-Taylor, 2001; Sokolova *et al.*, 2003; Marchand *et al.*, 2003) build sparse rule sets from a predefined rule repository. However, they do not provide weights and use the Occam's razor bound for capacity control. Finally, Lightweight Rule Induction (Weiss and Indurkhya, 2000) applies a simple voting procedure to combine individual rule sets into a predictive ensemble, but it does not explicitly optimize the margin.

Chapter 6

First-Order Rule Learning

This chapter deals with rule learning on data in first-order representation. After a short description of the new challenges introduced by first-order representations, we describe how the rule learner RUMBLE can be extended in a modular and flexible fashion to address some of the pragmatic considerations posed by the multi-relational setting. To address the challenges in a systematic way, we formulate a framework to assess and categorize multi-relational learning systems according to the way they extract information from the available data. Based on this framework, we devise a new rule generation procedure for graph-based data, which aims at highly diverse rule sets. The procedure applies stochastic local search to optimize a dispersion score quantifying the diversity of the rule set. Finally, we perform experiments to compare RUMBLE with other margin-based first-order learners, to demonstrate the utility of the framework, and to evaluate the dispersion-based rule generation procedure.

6.1 Motivation

In the preceding chapters we approached rule learning from a statistical perspective and proposed new methods to address the two main challenges in rule learning, namely, efficient empirical risk minimization and capacity control. The proposed methods, systems and analyses dealt with the traditional statistical setting, where a training or test instance is essentially a row in a table, that is, a vector. This data representation is easy to handle and applicable whenever the data is readily available in a single table format. However, in many real-world applications, the data is structured or too complex to be adequately modeled in a single table. For instance, it is awkward to represent molecules in a single table and impossible to so, if one row in the table should represent an instance. A much more natural and convenient way is to store the information in three tables, one for atoms, one for bonds and a third one to represent molecule instances. In this way each molecule is represented by one entry in the molecule table and its structure can be easily determined by following the references to the corresponding entries in the atom and bond relations. An additional advantage of such a multi-relational representation is the fact that one can incorporate *background knowledge* in additional tables. In the molecule example, one could give information about partial charges, hydrophobicity and so on. If supported by the learning system, some of the background information might even be computed dynamically so that a wealth of complex information can be accessed on demand without the need to store and manage large databases.

Learning systems which can deal with more than one data table are said to work in the *multi-relational* setting. Multi-relational representations are a special case of *first-order logic*, where relations can also be defined intentionally and recursive definitions allow for the formulation of complex relations. Not surprisingly, many multi-relational learning systems are based on Prolog or similar logic concepts. Research on inductive logic programming in particular has led to a range of first-order learning systems. The large majority of them are based on separate-and-conquer rule learning, although decision trees (Blockeel and De Raedt, 1998; Kramer, 1996) have also been employed.

From a statistical point of view, multi-relational classification is not very different from propositional classification. In both cases, the goal is to find classifiers that are accurate models of $\mathbf{Pr}[Y|X]$, that is, the conditional probability of the class label given a training or test instance. The main difference is the size and the structure of the input space. In propositional domains, the input space is simply the space of all possible attribute-value tuples. In many learning systems, this space is simply assumed to be equivalent to a vector space. In contrast, first-order input spaces can feature arbitrarily complex structure and be of immense size. Representations that work well in one case might fail in other cases. Sometimes, different aspects of the data contribute in different ways. For instance, in Structure-Activity Relationship (SAR) applications, small molecules might be represented as labeled graphs. There are many ways to extract relevant information from such a labeled graph: For instance, *structural local properties* of the graph's structure, such as the presence or absence of a specific subgraph, provide valuable information about active parts of a molecule. Sometimes, *descriptive local properties*, such as the partial charge of one specific atom can contribute to the classification. *Global summary statistics* or aggregated information over the whole graph might also contain significant information. Clearly, it is not feasible to make use of all possible structural or descriptive, global or local data properties in the learning step.

The main challenge in first-order rule learning is therefore to extract as much relevant information as possible from the training data. There are two aspects to this problem:

- Pragmatic considerations, such as data modelling and suitable methods to provide and use background knowledge, which might help to identify the relevant information to be extracted. We deal with those issues in Section 6.2.
- Fundamental considerations. These include analytical and empirical questions about how to generate features that extract information from the data and how to use certain kinds of information for classifier construction. We deal with those questions in the framework in Section 6.3 and propose a novel rule generation procedure based on the framework in Section 6.4.

From a user's perspective, the pragmatic issues are the more relevant ones. In most practical settings, there is some domain knowledge available that could be used to rate the relevance of individual data features. Incorporating such knowledge in the learning task can make a huge difference, because it allows the system to examine only presumedly relevant features and ignore irrelevant parts of the data. Also, the user might find it important to select a suitable representation for the input data. For example, deciding whether or not a substructure occurs in a molecular graph is an NP-hard problem. If graph and substructure data are stored in an unfavorable representation, occurrence checks can take considerable amounts of time. Therefore, relational learning systems should be designed to allow for a flexible and easy to understand, yet powerful way to specify meta information and represent input data. As stated above, we deal with such pragmatic considerations in Section 6.2, where we describe how RUMBLE can be used with first-order data.

The fundamental considerations mentioned above are more relevant for the designer of a multi-relational learning system. By extracting features from the input data, a learning system is essentially partitioning the input space $\mathcal{X}$ into pairwise disjoint parts $\mathcal{X}_1, \ldots, \mathcal{X}_t$, where the instances in each part give rise to the same feature values. Since the main goal is to model $\mathbf{Pr}(Y = y|X = x)$ accurately, the partitioning should be performed in a way that for each instance x in a part $\mathcal{X}_j$ $\mathbf{Pr}(Y = y|X = x) \geq 0.5$ for one class label y and $\mathbf{Pr}(Y = y'|X = x) \leq 0.5$ for any other class label $y' \neq y$. In other words, the input space should be partitioned so that the Bayes classifier assigns the same class label to all instances in one particular partition. Of course, since the true underlying distribution is not known exactly, it is impossible to find such a partitioning. However, we have access to the training sample, so we can estimate an approximate partitioning. Finding a good partitioning for relational data is a non-trivial task. It is complicated by the fact that most first-order learning systems perform the partitioning only implicitly by selecting features to be generated and included in the final model. Since this is often a complex process, it can be hard to compare and rate different approaches to feature generation. In Section 6.3 we describe a general framework that categorizes learning systems according to the way information is extracted from the input data. This allows theoretical and empirical investigations on how feature generation should be performed under which circumstances. In Section 6.4 we make use of the framework to derive a feature generation approach that aims specifically at features that complement each other. This leads to diverse, but small and therefore simple feature sets for first-order rule learning. We evaluate Rumble, the framework, and the feature generation approach empirically in Section 6.5.

6.2 Extending Rumble to First-Order Data

In principle, every propositional learning system can be upgraded to a first-order setting, for instance, by adding a preprocessing step that performs some form of propositionalization. However, when it comes to efficiency and ease of use, integrated systems that can deal directly with first-order data are often beneficiary. Unfortunately, upgrading propositional learners in an integrated fashion is not always easy. For instance, the SL^2 system presented in Section 4.3 applies a stochastic local search approach to empirical risk minimization. This works well for single relations with Boolean attributes, but it is unclear how to do it with many relations and inter-relation dependencies. Since the success of SLS depends to a certain degree on its efficient and fast inner loop and since specialization and generalization operators for first-order logic are expensive and complicated procedures, it is unlikely that SL^2's approach to empirical risk minimization can be extended towards the first-order case without sacrificing efficiency or accuracy. The situation is more favorable, though, for the margin-based approach taken by Rumble. Here, we already have a rather generic and flexible way to generate individual rules. Extending the rule generation procedure to incorporate first-order rules is not a large problem. In the following we describe two modifications to Rumble that were made specifically to improve support for multi-relational data.

The main goal for the design of the rule generation process is to allow for a flexible and generic way to specify a declarative bias. Consequently, we do not restrict the data representation to a particular format. For instance, it does not make sense to force the user to encode molecules as Prolog facts, if the data is already available in a much more efficient representation (for example, a canonical form for a graph mining tool). Instead, we implemented a plugin architecture, where the user can upgrade Rumble with plugins for various data representations. Each plugin offers a set of *refinement operators* that are called by the system to generate new rules. The refinement operator can access the existing set of rules, the weights of the current rule set, the training data and generic background knowledge to form new rules. A setup text file specifies, which plugins are loaded and which refinement operators are called in which order to generate the rules. Right now, RUMBLE provides five plugins:

- *Terms.* These are simple mathematical expressions such as $(a_1 + a_2)/2$ or $(a_1 \wedge a_3) \vee a_5$. The plugin provides refinement operators for adding new (fixed) term rules, modifying existing term rules or combining existing terms to form a new rule.
- *FreeTree.* Predicting the biological activity of small molecules is an important and popular application of ILP. Thus, we incorporate a version of the frequent substructure mining tool FreeTreeMiner (Rückert and Kramer, 2004a) as a plugin. It offers refinement operators that can build rules checking for the occurrence of a particular substructure, create all substructure rules that occur more frequently in the training data than a predefined threshold or refine existing rules depending on the existing rule set or weights.
- *Dispersion.* This plugin provides the dispersion-based feature generation method for graph data described in Section 6.4. As a side effect it can also be used to generate subgraph features using a frequent subgraph mining algorithm similar to gSpan (Yan and Han, 2002).
- *Prolog.* Prolog is a powerful and generic way to represent general data and background knowledge. The Prolog plugin incorporates a fully featured Prolog engine based on YAP Prolog (Costa and Lopes, 2007) or CxProlog (Dias, 2006). This enables the user to write her own refinement operators in Prolog.
- *Meta.* This plugin can combine existing rules from other plugins using simple logical combinations. The refinement operators can be configured to combine only rules whose instantiations meet certain conditions, for instance, minimum mutual information.

Figure 6.1 gives a small text file specifying a sample declarative bias for a small molecule data set. Initially, the file specifies the plugins to be loaded and the feature describing the target class. The following declaration states that the system should first generate 200 rules describing the substructures that occur in more than 20% of the molecules in the data set. Then, the Term plugin generates five rules that test the molecules' logp value as discretized in 5 equal-frequency bins. Finally, the Meta plugin combines those of the first 205 rules in a disjunction, whose mutual information is among the fifty lowest.

It is noteworthy that the system's predictive performance depends not only on the rule repository as a whole, but also on the order in which the rules are evaluted. If the system generates a large amount of rules, it is quite likely that one of the rules agrees with the target class label just by

```
# Required plugins
require <FreeTreePlugin>
require <TermPlugin>
require <MetaPlugin>

# Target feature
target Term yoshida.Class

# Refinement operators
refine FreeTree MinFreq yoshida.SMILES: 0.2 SortedBySize 200
refine Term NewFeature yoshida.Logp:bin1of5
refine Term NewFeature yoshida.Logp:bin2of5
refine Term NewFeature yoshida.Logp:bin3of5
refine Term NewFeature yoshida.Logp:bin4of5
refine Meta CombineMostUncorrelated [1-205] or 50
```

Figure 6.1: A sample bias description file

pure coincidence. Hence, the risk of overfitting increases with the number of generated rules and the structural risk term in equation (5.27) penalizes rules that are added only late. The user should therefore configure the system to generate presumedly informative rules first. For instance, in the task of predicting carcinogenicity of small molecules, one could generate rules that check for the existence of halogens (which are known to contribute to carcinogenicity) before presumedly less informative rules. This imprecise knowledge is in contrast to the design in many ILP systems, where the background knowledge is encoded in a precise logical form. The order can be interpreted as a Bayesian prior: early rules have greater prior probability.

Since the rules can be based on two or more relations, they aggregate information from different sources. Often, each part of a rule contributes some information about the target. Removing or replacing a part of the rule leads to a different, but related rule. This means that there is often not a single, but rather a whole number of related good rules. In many multi-relational domains the bias towards small rule sets does therefore not work very well, because it picks a single rule instead of making use of the complete information provided by all related rules. This was also the case for the experiments in Section 6.5. As a remedy, we generalize the MMV criterion in Rumble to use an arbitrary p-norm instead of the 1-norm for regularization. More formally, we compute the solution to

$$\begin{aligned} &\underset{w}{\text{maximize}}\, \hat{\mu}_w - \hat{\sigma}_w \\ &\text{subject to } \|w\|_p = \Big(\sum_{j=1}^{m} |w_j|^p\Big)^{\frac{1}{p}} = 1. \end{aligned} \tag{6.1}$$

Here, the main difference to the original MMV formulation in (5.3) is the norm parameter $p \geq 1$. It determines how evenly the weights should be distributed among the rules. In the original case $p = 1$, the space of feasible weight vectors $\{w||w|_1 = 1\}$ is peaked at the vertices, where one or more weights are zero. Hence, the optimization procedure sets the rule weight to exactly zero whenever the contribution of a rule is small compared to the contribution of other rules. The exact opposite is true when $p = \infty$. Here, the feasible set is peaked whenever $|w_i| = 1$ so that the resulting weight vector tends to assign full weight to many rules. Empirically, values between 1 and 4 work well on most problems. Regularization with $p = 2$ is a good compromise in many cases. In regression, a similar regularization criterion is known as *ridge penalty.* Before we perform experiments with this extended version of RUMBLE, we address the question of how rules should be generated in the first place. We frame this problem in a more precise framework in the next section and propose a dispersion-based approach to rule generation in Section 6.4.

6.3 A Framework for Relational Learning

As outlined in Section 6.2, the main difference between propositional and multi-relational learning is the fact that in multi-relational learning the input space contains complex, structured objects. Usually, these objects can be described in many ways and these descriptions can be aggregated or combined to form potentially useful criteria for classification. Hence, if a learning system wants to extract meaningful information, it has to deal with the combinatorial explosion of different information extraction possibilities. Since it is generally infeasible to generate all possible combinations, each multi-relational learning algorithm needs to address the problem of finding a good subset of features that are relevant for the classification problem at hand. In order to compare, analyze and assess different learning algorithms it is therefore essential to understand what kind of information is used in which way by each system. In the following we propose a framework that compares learning algorithms with regard to the information, which is extracted for classification. The framework is separated in two parts. In the high level view, we make no assumptions on the inner working of the actual learning algorithm. Instead, we focus only on the queries used to extract the information from the database. In the low-level view, we assume that the learner is designed based on a simple loop, which repeatedly generates, filters and sorts queries. This view is generic enough to cover a broad range of existing learning algorithms, while specific enough to allow for meaningful analytical and empirical investigations.

6.3.1 High-Level View

To model the case of first-order data, we assume from now on that $\mathcal{X}$ is simply a space of arbitrary instances. In contrast to the propositional case we do not make any assumptions about the semantics or representation of instances; they may be graphs, arbitrary objects in the real world or mathematical constructs. Instead of imposing any syntactical or semantical restrictions on the instances, we make use of a *query language* to derive pieces of information about specific instances. For our purposes a *query* is a function from the instance space to the real numbers extended with a "don't know" symbol: $q : \mathcal{X} \to R$, where $R := \mathbb{R} \cup \{\varepsilon\}$ and the abstention symbol ε denotes "no information available". We assume that a query extracts some information from the instances in $\mathcal{X}$, such as the presence or absence of some properties, the number of nodes, or the result of some mathematical function. In typical applications, one would encode nominal values, such as *red*, *blue* and *green* using two-valued indicator variables and numerical (for example size) or aggregated quantities (for example mean size of referenced objects) as real numbers. We assume a fixed space $\mathcal{Q}$ of possible queries. We also need a way to encode a query as a string of symbols in a computer. For that purpose, we define a language $\mathcal{L}_Q$, so that every string $s_q \in \mathcal{L}_Q$ represents exactly one query $q \in \mathcal{Q}$. For ease of notation, we do not distinguish between q and s_q explicitly, but simply write q whenever we mean "q as a string", and $q(x)$ whenever we apply the query q as a function to x.

Later on, we will apply queries to whole sequences of instances $X \in \mathcal{X}^n$. Again, we denote the component-wise application of q on such as sequence of instances X by $q(X)$, so that $q : \mathcal{X}^n \to R^n$ can be seen a a function of arbitrary arity n. Finally, let $\mathcal{Y} := \{-1, 1\}$ be the set of *class labels*, so that $\mathcal{X} \times \mathcal{Y}$ denotes the *labeled instance space*. We are concerned with a learning system, that induces classifiers[1] from data. We assume that the system is given a training set $(X, Y) \in (\mathcal{X} \times \mathcal{Y})^n$ of n labeled instances $(x_1, y_1), \ldots, (x_n, y_n)$. A classifier is a function that assigns a class to each instance in $\mathcal{X}$. In our setting we formalize this as follows: a classifier $c := (k, d_c, Q_c)$ of size k contains a *decision function* $d_c : R^k \to \mathcal{Y}$ and a sequence of queries $Q_c \in \mathcal{Q}^k$. A classifier assigns a class label to each instance by evaluating the queries and combing the query results through the decision function: if $Q_c = (q_1, \ldots, q_k)$, then $c(x) := d_c(q_1(x), \ldots, q_k(x))$. Let $\mathcal{C}$ be the space of all classifiers. In

[1]All of the following considerations are equally valid if one is concerned with regression rather than classification. Further, generalizations to more complex learning settings such as multi-class, multi-label, or multi-task learning are straightforward.

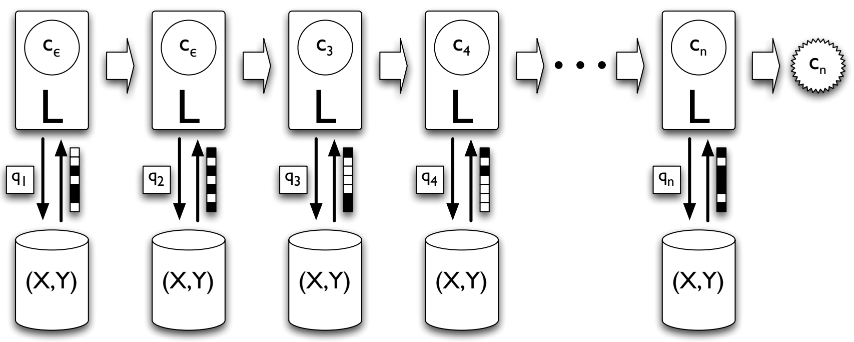

Figure 6.2: The high-level view of a relational learning system in the framework: The learning system L poses queries $q_1, \ldots, q_n$ to the database and receives an instantiation vector for each query. Sometimes, after receiving an instantiation vector, the system updates its current classifier. Here, c_i denotes the current (partial) classifier after posing query q_i.

particular, let c_ε denote the empty classifier. A *learning system* L is given a training set (X, Y) as input. After some computations it outputs a classifier c that can be used to make new predictions on new instances. Thus, we can formalize a learning system as a function $L : (X, Y) \to \mathcal{C}$.

Of course, the learning system is implemented as a computer program. In order to gather information about the training set, it poses queries to the database, which contains the training instances. We would like to classify and investigate the various ways a learning algorithm generates new queries and ultimately induces a final classifier. To do so, we observe the queries $q_1, q_2, \ldots, q_l$ that are sent from the system to the database in chronological order. We assume that L is a deterministic algorithm, so that each q_i is uniquely identified by the input to L, that is, the training set and (possibly) some input parameter. This is not a severe limitation as most non-deterministic algorithms use in practice a pseudo random number generator. For the ease of presentation we also assume that the algorithm never sends the same query twice, so that $q_i \neq q_j$ for all $1 \leq i, j \leq l$. An algorithm can be easily extended with a caching procedure to meet this condition. Finally, we do not demand that the queries are evaluated on all instances in the database. Some algorithms (for example separate-and-conquer approaches) perform the queries only on subsets of instances, others only on single instances (for instance SVMs). For simplicity of representation we assume that the database always returns a vector $s \in R^n$, where the undesired components of s are simply set to ε.

Since we are concerned with deterministic learning machines only, a query q_k is uniquely identified by the list of preceding queries $q_1, \ldots, q_{k-1}$, the corresponding result vectors $(q_1(X), \ldots, q_{k-1}(X))$ and the class label vector Y. Let $\vec{q}_k : \mathcal{L}_Q^k \times R^{n \times k} \times Y \to \mathcal{Q}$ be the function, that decides, which new query is sent to the database, after the system has sent the preceding k queries, and observed the corresponding instantiations and the class label vector Y.[2] Thus, the kth query can be reconstructed by an application of $\vec{q}_k$ on the data that is returned by the database for $\vec{q}_1, \ldots, \vec{q}_{k-1}$.

Some algorithms build the classifier to be output in an incremental fashion. For example, separate-and-conquer rule learning systems iteratively add new rules to an initially empty rule set until a stopping criterion is met. To model this stepwise refinement approach, we peek inside the learning system to observe which (partial) classifiers have been built so far. We denote the current classifier of the algorithm after sending k queries to the database as c_k. When the machine maintains no classifier until query k, we set the $c_1, \ldots, c_k$ to the empty classifier c_ε. By convention, $c_0 = c_\varepsilon$, because there is no current classifier before the system starts and $c_l = L(X, Y)$, because after the last query is posed, the algorithm builds and outputs the final classifier. Similar to the r_i, the system's current classifier after query q_k depends only on the queries so far, their instantiations and the class labels. We define the function $\vec{c}_k : \mathcal{L}_Q^k \times R^{n \times k} \times Y \to \mathcal{C}$ to output the current classifier of the system after seeing the first k queries, the corresponding instantiations and Y. The setup is illustrated in Figure 6.2.

This formalization allows us to categorize learning systems into certain categories. First of all, we can discriminate between systems that build the classifier incrementally and systems that generate all the queries first, but induce the final classifier only in the last step. We say that a learning system is a *propositionalization system*, if for all $k < l$: $c_k = c_\varepsilon$ and $c_l = L(X, Y)$. Otherwise, the system is a *stepwise refining system*. In principle, every refinement system can be reduced to a propositionalization system, because one can modify the $\vec{c}_k(q_1, \ldots, q_k, q_1(X), \ldots, q_k(X), Y)$ to output only the empty classifier for $k < l$ and keep only the last classifier generation function $\vec{c}_l$. However, having access to an initial (partial) classifier can speed up the computation of the $\vec{q}_k$ and $\vec{s}_k$ considerably. Many practical systems are therefore implemented as refining systems.

Another way to categorize multi-relational learning systems is by the information that is used in the $\vec{q}_k$ and $\vec{s}_k$. One can distinguish between three sources of information:

- The dependence of $\vec{q}_k$ and $\vec{c}_k$ on the preceding *query representations*. If the generation of new queries does not depend on the representation of the preceding queries, we call the system *agnostic propositionalization*. For instance, in the case of small molecule data, a learning system could simply process a list of predefined queries that check for certain active functional groups of a small molecule. However, this is

[2]In slight abuse of notation we use the arrow in $\vec{q}$ to indicate that $\vec{q}$ describes the transition from k to $k + 1$.

rarely practical. Many systems generate the queries according to a "more general than" order imposed on the query strings in $\mathcal{L}_Q$. Sometimes, the dependence of $\vec{q}_k$ on the preceding queries can by configured explicitly by a user, often by specifying *refinement operators*.

- The dependence of $\vec{q}_k$ and $\vec{c}_k$ on the preceding *instantiations*. The main motivation to use this information is to find a preferably diverse and informative set of queries. Some systems prune away queries whose instantiations are duplicates of existing queries. On graph data it is common to prune away queries that are satisfied or violated on less instances than a predefined threshold, because such queries introduce little discriminative power and are therefore less desirable. If the queries in $\mathcal{L}_Q$ are ordered by generality, threshold-based pruning can be implemented efficiently, because it depends only on the instantiation of the least general generalizations of a query. In Section 6.5.3 we evaluate a dispersion-based approach that depends on all preceding instantiations to produce a set of queries with preferably diverse instantiations.
- The dependence of $\vec{q}_k$ and $\vec{c}_k$ on the *class labels* Y. This is the most prominently used information for $\vec{c}_k$, because every good classifier is based on queries that are as informative as possible about the target. The influence of the class labels on $\vec{q}_k$ is less clear, because it is hard to predict whether or not a new feature will offer valuable information about the target before evaluating it. We will see later that one can do remarkably well with $\vec{q}_k$ that do not depend on the class labels.

The categorization of learning algorithms according to the utilization of available information can be used as a foundation for theoretical analyses of existing and novel systems. For instance, a system that makes use of the class labels Y to build the queries is more susceptible to overfitting than a learner that ignores this information. An analytical investigation could quantify this phenomenon. A similar effect takes place when a learner generates queries whose instantiations are highly correlated. A simple application of the PAC-Bayesian bound can be used to determine analytically, to which degree the queries' inter-correlation influences the overfitting behavior.

In most learning systems, $\vec{q}_k$ and $\vec{c}_k$ depend on all three sources of information, although the actual dependencies vary considerably. Often, performance considerations lead to systems that use certain types of information only implicitly, so that their influence on the output is only marginal. Sometimes there are complicated interdependencies that make it hard to assess the influence of each source of information. However, there are also many learning systems which proceed in a way that can easily be formalized. For example, a very common approach to relational learning is to extend an initially empty classifier step by step until it reaches a sufficient level of proficiency. Often, such a system iterates in a loop: First, it generates a new *batch* of queries and sends it to the database. Then, it filters those queries that are likely to be relevant and adds them to the current (partial) classifier. In the next section we give a formal way to describe this (or similar) approaches to query and classifier construction.

6.3.2 Low-Level View

The level of the framework described in the preceding section focuses on the sequence of queries and models from an abstract point of view. In particular, it does not consider the way the queries in a sequence are generated. In the following, we take a look inside the learning system's "black box" and take a more procedural view of the query and classifier generation process. On this level, we deal with the information effectively used to generate individual queries or batches of queries. For instance, we can express formally whether information about the target class is used directly (by looking it up in the database) or only indirectly (via looking at a partially induced classifier) in this process. To do so, we abstract from single queries and instead deal with batches of queries, where each batch contains the queries generated between two classifier updates. More formally, recall that c_k denotes the current classifier of the system after sending the kth query q_k. If a system does not update the classifier after q_k, then $c_k = c_{k-1}$. For each current classifier c let $Q(c) := \{q_i | c_i = c\}$ be the set of queries that do not lead to an update of c. It is clear that the $Q(.)$ can be used to partition the the query sequence $q_0, \ldots, q_l$ into a sequence of *query batches* $Q_1 = Q(q_0), \ldots, Q_{l'} = Q(q_l)$, so that the Q_k contain exactly those queries that are sent to the database between two classifier updates. We can now investigate, how a system generates each batch of queries. For instance, when modeling a stepwise refinement system (for example, Tilde, Blockeel and De Raedt, 1998), the algorithm obtains the next batch of queries or the next query by maximizing some criterion (for instance with respect to the target class). Another possibility is to generate the next queries depending on the parameters of the decision function d_c of the current classifier.

To allow for a more fine-grained formalization of the classifier update process, we frame the construction of a query batch Q_k as follows: First, the queries in Q_k are (syntactically) generated depending on some information available at the time of generation, then they are filtered, sorted, and finally handed over to a classifier construction procedure, which derives the decision function d_c. For each of these steps, we consider the information that is necessary to perform them: information about the instances, their class labels, the previously generated queries, or the partially learned model. The generated queries are presented to the classifier construction part of the learning system, which selects, weights, or otherwise processes them to combine their information in a decision function d_c for prediction.

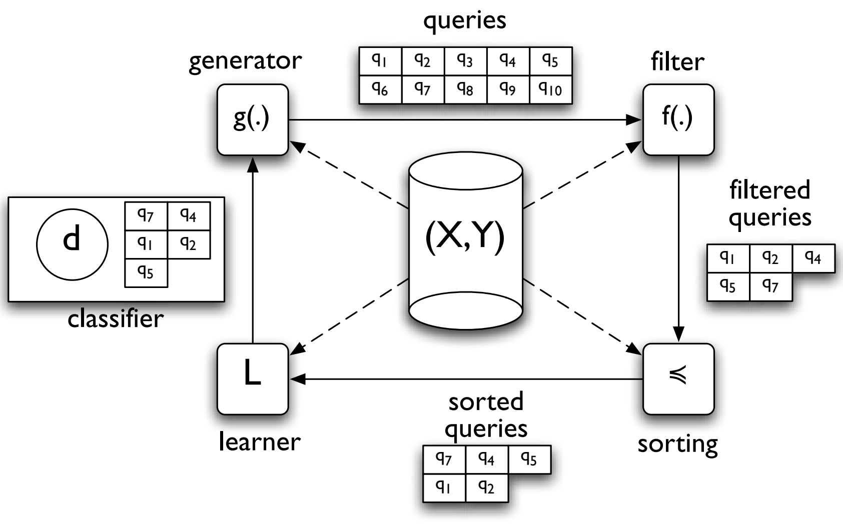

Figure 6.3: The low-level view of a relational learning system in the framework: each batch of queries is generated by $g(.)$, filtered through $f(.)$, then sorted according to $\preceq$ and finally handed over to the classifier construction, where the queries are augmented with the decision function d to form an (intermediate) classifier. Each component can base its actions on previous queries, the training instances, the labels and/or previous classifiers. This procedure is repeated iteratively until a final classifier is output.

To formally define the process of query generation and the information used therein, we have to introduce a few functions and predicates. First, we assume that the system contains a procedure b that generates the next batch of queries. The resulting batch of queries is possibly sorted according to some criterion and presented to the classifier construction procedure. Different implementations of b are conceivable: For instance, b could use information about the queries q_1 to q_k generated so far, X resp. Y, or the current decision function d_c. Zooming in on b, we may find it practical for some systems to distinguish between the process of generating and filtering queries. In other words, we assume the batch of queries is first generated using a so-called generator procedure g and then filtered according to a filter procedure f. Thus, b consists of two parts g and f where each query generated by g is given as input to f. This does not necessarily need to happen in a chronological way (first call g, then f), but it can also happen in a more complex fashion.

Typical generator functions g generate queries syntactically based on a declarative language bias. The filter function f can be thought of as applying an interestingness predicate p, as known from the data mining literature (Mannila and Toivonen, 1997), to a set of generated queries $f(Q) = \{q \in$

	generator	filter	sorting predicate
agnostic	$g()$	$f(Q')$	$\preceq$
syntax-dependent	$g(q_1, ..., q_k)$	$f(Q')$	$\preceq$
instantiation-dependent	$g(X)$	$f_X(Q')$	$\preceq_X$
interaction-depdendent	$g(q_1, ..., q_k, X)$	$f_{q_1,...,q_k,X}(Q')$	$\preceq_{q_1,...,q_k,X}$
class-dependent	$g(q_1, ..., q_k, X, Y)$	$f_{X,Y}(Q')$	$\preceq_{X,Y}$
model-dependent	$g(q_1, ..., q_k, d_c)$	$f_{q_1,...,q_k,d_c}(Q')$	$\preceq_{q_1,...,q_k,d_c}$

Table 6.1: Sources of information for generator function, filter function, and sorting order.

$Q|p(q)\}$. In some instantiations of the framework, f may pick a single query, for instance, by optimizing a scoring function. Splitting the process into generating and filtering, it is possible to reconstruct pattern mining approaches like Warmr (Dehaspe and Toivonen, 1999) or classical ILP systems like FOIL (Quinlan and Cameron-Jones, 1995) in some detail (see below). Optionally, we can sort the resulting queries for the batch according to some order $\preceq$, which in turn may depend on various types of information. Finally, the newly generated, filtered, and sorted batch of queries has to be instantiated with respect to the instances of a database (if it has not already been instantiated by g, f or the sorting procedure), before it can be processed by the classifier construction procedure, which induces d_c.

The generator/filter functions as well as the sorting predicate may be based on different sources of information (see Table 6.1). If the data without the class are accessed, we have X as an argument or as a subscript. If the queries from the sequence up to index k are taken as input, we have q_1 to q_k either as argument or as subscript. Analogously, we have Y or d_c as an argument of g, or as a subscript of f or $\preceq$, if the target class or the current model is required. A schematic illustration of the classifier induction loop in this framework is given in Figure 6.3. Modelling a relational learning system as an iterated query generation, filtering, sorting and learning procedure might appear arbitrary or ad-hoc. However, we feel that the presented framework not only enables theoretical and empirical investigations, but is also flexible and generic enough to apply to existing systems. In the next section we give a short survey of how a selection of existing relational learning systems can be described and classified in the framework.

6.3.3 Instantiations of the Framework

In this section, we briefly discuss known and conceivable instantiations of the above framework. First, many traditional ILP algorithms proceed top-down, iterating query generation and testing. Typically, models consist of queries that are combined conjunctively or disjunctively. Queries that are already part of a model are syntactally refined according to a declarative language bias specification. In this process, existing queries are often replaced by newly generated queries. This scheme applies to many classical ILP algorithms, from FOIL (Quinlan, 1990) to Tilde (Blockeel and De Raedt, 1998). In terms of the framework, the queries are generated depending on previous queries ($g(q_1, ..., q_k)$) and evaluated with respect to the class ($f_{X,Y}(Q')$). The state of the classifier d_c is not included in the generation function, because the queries are combined conjunctively and no additional information has to be considered for the purpose of query generation. Surprisingly, the kFOIL (Landwehr *et al.*, 2006) approach shares the same pattern of necessary information ($g(q_1, ..., q_k)$ and $f_{X,Y}(Q')$).

Some systems use more complex decision functions than just conjunctions, for instance, linear combinations of queries. In these cases further options are available. Structural logistic regression (Popescul *et al.*, 2002; Popescul and Ungar, 2003) refines only those features already incorporated into the logistic regression model ($g(q_1, ..., q_k, d_c)$ and $f_{X,Y}(Q')$). Data-driven approaches access the database already for the generation of new queries. This can be observed in simple extensions of top-down methods, where information about training instances is extracted "on the fly" from the database and directly used in newly generated queries. As an example, consider thresholds for tests on numeric variables in Tilde ($g(q_1, ..., q_k, X)$), subsequently filtered as above with respect to the class ($f_{X,Y}(Q')$).

Progol (Muggleton, 1995) and other bottom-up, data-driven approaches as implemented in Aleph (Železný *et al.*, 2006) generate bottom clauses based on single training instances ($g(X)$), and then search for compressive and predictive clauses above the bottom clause in the lattice ($f_{X,Y}(Q')$). Other ILP approaches based on the generalizations of pairs of instances can be treated similarly. Data-driven propositionalization approaches include Roth and Yih's relation generation functions (RGFs) (Roth and Yih, 2001) and aggregation-based methods (Krogel and Wrobel, 2001). Relation generation functions generate features by the application of general templates to training instances ($g(X)$ and $f(Q') = Q'$). Roth and Yih also investigated the use of frequency-based filtering ($f_X(Q')$) in subsequent publications. Aggregate functions for propositionalization were proposed by Krogel and Wrobel in their RELAGGS system ($g(X)$) and $f(Q') = Q'$). In his PhD thesis (Krogel, 2005), Krogel also discusses the elimination of redundant features whenever there are functional dependencies between two or more predicates (again $f_X(Q')$).

Other propositionalization approaches proceed purely syntactical in the generation of new queries (Železný and Lavrač, 2006) ($g()$). However, syntactic constraints such as the non-decomposability of structural features are known to restrict the search space considerably. Moreover, feature filtering is performed ($f_X(Q')$), making sure that any two features have different extensions. In contrast to stepwise refinement methods, however, features are generated at once and not incrementally. Top-down propositionalization methods like Warmr (Dehaspe and Toivonen, 1999) and AGM (Inokuchi *et al.*, 2003) generate or refine existing queries syntactically ($g(q_1, ..., q_k)$) and filter according to frequency constraints ($f_X(Q')$). The graph mining algorithm gSpan (Yan and Han, 2002) generates new queries by searching for frequent extensions of queries already known to be frequent ($g(q_1, ..., q_k, X)$ and $f(Q') = Q'$).

6.4 Dispersion-Based Rule Generation for Structured Data

In the preceding section we described a framework to rate a relational learning system according to the kind of information it uses for rule generation, filtering and sorting. Before reporting on experiments performed within the framework in the next section, we describe a rule generation procedure for Rumble that is inspired directly by the presented framework. The procedure is motivated by the observation that many approaches to learning from molecular graph data tend to generate large amounts of features. This is the case even for methods that use information about the class labels during feature generation ($g(q_1, ..., q_k, X, Y)$ in the framework). This is in contrast to the following simple lemma, which bounds the number of features that are necessary for linear classifiers.

Lemma 6.4.1. *Let $X \in \{-1, 1\}^{n \times m}$ be a training matrix with rank $d < n$. Then there are at least 2^d weight vectors that induce a different partitioning into positive and negative instances.*

Proof. Since X has rank d, one can find a $d \times d$ submatrix X' of full rank. Consider an arbitrary target vector $y' \in \{-1, 1\}^d$. Since X' has full rank, there is exactly one weight vector $w' \in \mathbb{R}^d$ which is a solution to the linear equation system $X'w' = y'$. Insert zeros in those positions of w', where a column was removed to generate X' from X and denote the resulting weight vector by $w \in \mathbb{R}^m$. It is easy to see that w assigns the values of y' to the instances (rows) that were not removed while building X' from X. The lemma follows, because there are exactly 2^d different y's. $\square$

Thus, if one selects the features in a way that the training matrix has full rank, then one needs only n features to enable the learning algorithm to induce any possible dichotomy and reach 100% training accuracy. Thus, even if the learning algorithm does not use the class labels ($g(q_1, ..., q_k, X)$ in the framework), there is no need to generate more than $n - 1$ rules. Since many existing systems generate significantly more than n features, their feature generation process appears to make inefficient use of the available information in the training data.

In the following we present a feature generation procedure for graph-structured instances that aims at finding small, but complementary and diverse feature sets. For the sake of clarity, we only deal with features, which specify whether or not a subgraph is present in an instance. However, the approach is general enough to also work on more general representations, if desired. Even though subgraph features are popular for graph classification problems, the number of subgraphs grows exponentially with the size of the graphs. Hence, it is clearly infeasible to use all possible subgraphs as features. Therefore, many approaches restrict the set of subgraph features to frequently occurring, frequent closed, or class-correlated subgraphs (Deshpande *et al.*, 2005; Bringmann *et al.*, 2006). In all of these cases, however, there is no guarantee that the resulting feature sets have sufficient coverage over all instances of a data set. Moreover, the resulting features may be only slight alterations of a few subgraphs. As a consequence, the number of features required to reach some level of performance may be unnecessarily high, potentially harming comprehensibility and efficiency. While we focus on graph classification in this study, the same problems occur with other forms of structured data, for instance, in logic-based representations.

Instead of generating a bulk of features blindly and obtaining a well-balanced coverage only incidentally, it may be worthwhile to actively construct favorable structural features in the first place. One way to minimize the number of features required is to explicitly maximize the diversity of subgraph features. Thus, we frame the search for diverse, complementary sets of subgraph features as an optimization problem. We define a scoring function measuring the diversity of a feature set and present a stochastic local search (SLS) procedure to find subgraphs optimizing that score. Stochastic local search is motivated by the NP-hardness of the problem of finding a perfectly complementary subgraph given a set of graphs and optimal features so far. To focus the search for useful structural features even further, it is possible to extend the scoring function by balancing diversity with class correlation. The resulting feature sets are used in linear classifiers. In Section 6.5.3, the effectiveness of the method and its dependence on variations are tested in experiments on three small molecule data sets from cheminformatics.

6.4.1 Finding Expressive Rule Sets

Recall that we deal with the problem of *rule generation* for linear classifiers on first-order data. In this setting, the instances are arbitrary objects and we need to find rules that extract meaningful information from the objects. In particular we are interested in features that are well suited for use by a learning algorithm to construct a predictive classifier. Information theory gives us the tools to quantify what constitues a feature set that is informative about the target class: Let $X \in \{-1, 1\}^{n \times m}$ be the training matrix containing n instances $x_1, \ldots, x_n$, where each instance $x_i = (x_i(1), \ldots, x_i(m))$

ranges over m features. X_i denotes the ith column of the training matrix, that is, the instantiation of the ith feature. Assuming a binary classification problem, we denote the class labels of the instances by $Y \in \{-1, 1\}^n$. Then we are aiming at features $X_1, \ldots X_m$ with high mutual information $I(X; Y) = I(X_1, \ldots, X_m; Y)$ between features and target class. We can write the mutual information as the difference between the entropy of X and the conditional entropy of X given Y: $I(X; Y) = H(X) - H(X|Y)$. Thus, in order to obtain highly informative features, we need to maximize $H(X) := H(X_1, \ldots, X_m)$ and to minimize $H(X|Y) := H(X_1, \ldots, X_m|Y)$. This leads to the following three criteria:

- *High correlation with the class.* Since we would like to minimize $H(X|Y)$, we are looking for features that are highly correlated with Y. This is the criterion that is most prominently applied in most traditional multi-relational learning systems. Theoretically, a single feature X_i agreeing with Y on all instances would be enough to ensure $H(X_i|Y) = 0$. In practice it is rarely possible to find such a perfect feature and often there is only a small number of features with high correlation. In such a setting, the learning algorithm also needs to consider features with comparably low correlation and the two other criteria below become relevant for optimal feature construction.

- *High feature entropy.* The joint entropy can be upper-bounded by the sum of single features: $H(X) = \sum_{i=1}^{m} H(X_i|X_{i-1}, \ldots, X_1) \leq \sum_{i=1}^{m} H(X_i)$. Thus, in order to maximize $H(X)$ we need to maximize the entropy of each single feature. For Boolean features this means that each feature should divide the training instances in two parts of preferably equal size, so that it assigns -1 to roughly the same number of objects as $+1$. This is intuitively intriguing: a set of k features that assign $+1$ to only one training instance and -1 to the others can discriminate only between k different instances, whereas a set of features that divide the instances into two equal-sized parts can discriminate between up to 2^k bins of instances.

- *High inter-feature entropy.* Even if all single features have maximal entropy, it could be the case that the features are highly correlated to each other. In the most extreme case, it could be that all features assign the same labels to all instances $X_1 = \ldots = X_m$. Clearly, we need to ensure that the features complement each other and do not provide the same information all over again. In terms of information theory one can write $H(X) = \sum_{i=1}^{m} H(X_i|X_{i-1}, \ldots, X_1) \leq \sum_{i=1}^{m} H(X_i|X_{i-1})$. Thus we need to maximize $H(X_i|X_{i-1})$ for all $1 < i \leq n$. Since the features can be ordered arbitrarily, this essentially means we need to maximize $H(X_i|X_j)$ for each pair of features.

The first criterion has been dealt with to great extent in the existing literature on relational learning, the second is sometimes addressed by putting minimum frequency constraints on the features, and the third is usually not considered or included only implicitly. This is a problem in particular in the graph learning setting, where a feature indicates the occurrence or absence of a substructure in a graph. Typically, it is easy to construct substructures that appear in only a very small number of graphs, so the entropy of single features tends to be low. To avoid this, one often selects substructures that appear only with a certain minimal frequency in the graph database. Unfortunately, the resulting substructure's instantiations are often very similar so that inter-feature entropy is low. For instance, mining all subgraphs that appear in at least six percent of the NCTRER data set (see Section 6.5.3) yields 83,537 frequent subgraphs. However, when comparing the instantiation X_i with X_j for all pairs of subgraphs (i, j), it turns out, that in 19% of the pairs $X_i = X_j$ and in 77% of the pairs X_i differs from X_j on less than ten instances. Hence, training matrices based on minimum frequency mining tend to be large and exhibit an unnecessarily large degree of redundancy.

On the other hand, it is easy to see that *Hadamard matrices* constitute optimal training matrices with regard to the latter two criteria, because any two columns are orthogonal (so $H(X_i, X_j) = 2$ is maximal for all $i \neq j$), and the number of ones is equal to the number of minus ones in each column (so $H(X_i) = 1$ is maximal for all i). Hadamard matrices of order 2^i can be generated using Sylvester's recursive construction:

$$H_1 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \quad H_i = \begin{pmatrix} H_{i-1} & H_{i-1} \\ H_{i-1} & -H_{i-1} \end{pmatrix}.$$

In the following we propose a method that optimizes the second and third criterion explicitly for *linear classifiers* on subgraph features. A linear classifier is given by a direction vector $w \in [-1, 1]^m$ and an intercept w_0. As usual, the classifier predicts the positive class for an instance x, whenever $w^T x - w_0 > 0$ and the negative class otherwise. Hadamard matrices are especially well suited for linear classifiers, because they have full rank. As shown in Lemma 6.4.1, a training matrix of rank d can give rise to 2^d different linear classifiers, each representing a unique partitioning of the training instances into two classes. Thus, each of the 2^n different ways to split the instances into a positive and a negative class can be represented by a linear classifier on a Hadamard training matrix. This means a rule generation procedure that produces Hadamard matrices needs to generate only n rules to achieve 100% training accuracy, regardless of the actual class labels Y.

While such a training matrix would be optimal for learning, it is certainly impossible to find subgraphs whose instantiations give rise to a Hadamard matrix. Instead, we are faced with the problem of finding subgraphs whose instantiations approximate a Hadamard matrix as good as possible. In the following, we assume a forward selection setting and frame the problem as an iterative combinatorial optimization problem: Let $D = \{g_1, \ldots, g_n\}$ be a set of graphs (the instances), and $F = \{f_1, \ldots, f_m\}$ be a set of subgraphs (the already existing features), and denote by s_i the instantiation vector of the ith subgraph f_i with regard to D, that is, the vector whose jth component is 1 if f_i is a subgraph of g_j and -1 otherwise. We are then looking for a new subgraph f_{m+1} whose instantiation vector s_{m+1} optimizes some score quantifying the criteria explained above. For the experiments in Section 6.5.3, we use what we call a *dispersion score* in the remainder of the chapter:[3]

$$d(s_{m+1}) := \sum_{i=1}^{m} (s_i^T s_{m+1})^2$$

Minimizing this score adresses the criteria in an elegant way: first of all, the dot product between the new instance vector and an existing vector is a measure of the similarity of the two vectors. Its absolute value is large, if the two vectors agree on all instances (or, equivalently, disagree on all instances), and zero, if the two vectors agree and disagree on an equal amount of instances. By summing up the square of the dot products, the score therefore penalizes features that are similar to existing ones. If the score is zero, the new instance vector is orthogonal to all existing vectors and thus increases the rank of the training matrix. Second, the score can be reformulated as a sum over all pairs of instances, where $x(j)$ denotes the jth component of vector x:

$$\begin{aligned} d(s_{m+1}) &= \sum_{i=1}^{m} \Big(\sum_{k=1}^{n} s_i(k) s_{m+1}(k) \Big)^2 \\ &= \sum_{k,l}^{n} s_{m+1}(k) s_{m+1}(l) \sum_{i=1}^{m} s_i(k) s_i(l) \end{aligned} \tag{6.2}$$

The second sum on the right side of (6.2) measures how many features assign the same label to the instances g_k and g_l. It is maximal if all features assign the same label and minimal if all features disagree on those two instances. Thus, for each k, l the summand in the first sum is minimal, if s_{m+1} assigns different labels whenever the existing features assign identical labels, and the same label whenever the majority of features assigns different labels. Minimizing this score therefore promotes features that discriminate between instances that have not been discriminated well by the existing features. Third, the score reaches the global optimum zero precisely for the Hadamard matrix. Of course, the dispersion score does not aim at finding features that correlate well with the target. In order to also incorporate the first criterion, we extend it to not only penalize features that are similar to

[3]In principle one could also apply mutual information instead of the dispersion score, but experiments have shown that this is not effective for the data sets in Section 6.5.3.

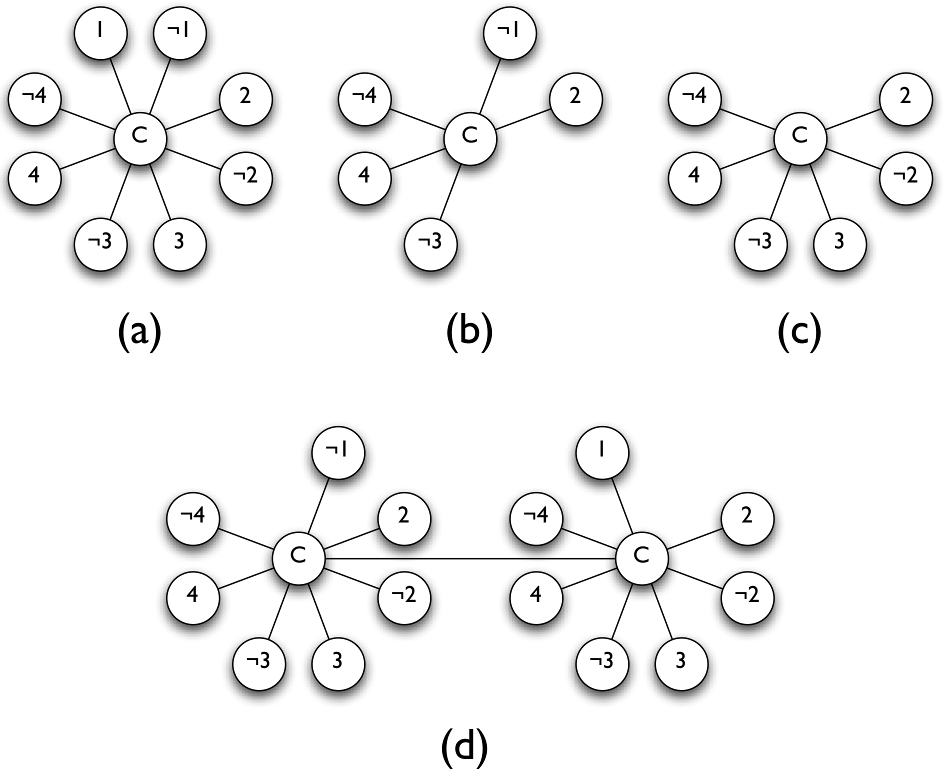

Figure 6.4: The graphs that are generated during the reduction of a 3SAT problem to a dispersion score problem. The graph s for four Boolean variables V_1, V_2, V_3, V_4 is given in (a), the graph from G_C corresponding to the clause $(V_1 \wedge \neg V_2 \wedge V_3)$ in (b), the graph corresponding to V_1 from G_V in (c) and the graph corresponding to V_1 from G_D in (d).

the existing features s_i, but also to reward features that are similar to the target class vector t:

$$d'(s_{m+1}) := \sum_{i=1}^{m} (s_i^T s_{m+1})^2 - m(t^T s_{m+1})^2$$

The modified *class-correlated dispersion score* is designed to value dispersion to the same extent as similarity with the target. In the following section we describe our algorithmic approach to optimizing the dispersion score.

6.4.2 Stochastic Local Search for Optimal Dispersion Features

The dispersion score provides a practical way to formulate the search for features with large discriminative power as a combinatorial optimization problem. Unfortunately, due to the complexity inherent in graph operations, the problem can be extremely difficult to solve. It is clear that computing the instantiation vector for an arbitrary graph involves the repeated computation of solutions to NP-complete graph isomorphism problems. Even if one avoids these subgraph isomorphism tests, the problem can be shown to be NP-hard:

Theorem 6.4.2. *The problem of deciding whether there exists a graph that achieves a dispersion score of zero on a given training matrix and a given graph database is NP-hard.*

Proof. The proof is based on a reduction of 3SAT to the dispersion score problem. Let S be a CNF formula containing k clauses over l Boolean variables $V_1, \ldots, V_l$. Let $n = 2(k + l)$. We construct a training matrix X and a graph database $G = (g_1, \ldots, g_n)$ and show that there exists a subgraph h whose instantiation vector t_h on G has dispersion score $d(t_h) = 0$ if and only if there is a truth value assignment on $V_1, \ldots, V_l$ that satisfies S.

First, let H be a $n \times n$ Hadamard matrix. Denote the last row of H by r and let the training matrix X be H with the last row r removed. The dispersion score of a new feature t is $d(t) = \sum_{i=1}^{n-1}(x_i^T t)^2$, where the x_i are the columns of X. In order to have $d(t) = 0$, a new feature t must satisfy $x_i^T t = 0$ for $1 \leq i \leq n-1$. Since X has rank $n-1$ and there are $n-1$ linear equations for the n components of t, the solution space $\{t | d(t) = 0\}$ is a one-dimensional vector space, that is, a line. However, since we are interested only in instantiation vectors $t \in \{-1, +1\}^n$, the only two solutions with $d(t) = 0$ are r and $-r$. We denote the set of indices where $r_i = +1$ by $R_{+1} := \{i | r_i = +1\}$ and the set of indices where $r_i = -1$ by $R_{-1} := \{i | r_i = -1\}$. Since r is part of a Hadamard matrix, $|R_{+1}| = |R_{-1}| = \frac{n}{2} = k + l$.

Now, we generate a database $G = (g_1, \ldots, g_n)$ depending on the clauses in S. All graphs in G are subgraphs of the star-shaped graph $s = (V_s, E_s, l_s)$ with the vertex set V_s, the edge set E_s, the vertex label function $l_s : V_s \to L_s$ and the set of vertex labels L_s. We define s as follows: $V_s = \{v_c, v_1, v_{\neg 1}, v_2, v_{\neg 2}, \ldots, v_m, v_{\neg m}\}$ so that it contains one center vertex v_c (the center of the star) and two vertices for each variable V_i in S. Intuitively, vertex v_i encodes the truth assignment "variable V_i is true" and $v_{\neg i}$ encodes "variable V_i is false". $E_s = \{\{v_c, v_i\} | 1 \leq i \leq n\} \cup \{\{v_c, v_{\neg i}\} | 1 \leq i \leq n\}$ contains only edges between the star center v_c and the truth assignment vertices v_i and $v_{\neg i}$. Finally, we set $L_s = \{c, 1, \neg 1, \ldots, n, \neg n\}$ and $l_s : v_x \mapsto x$, so that each vertex is labeled with its subscript. Figure 6.4 (a) gives s for a 3SAT problem with four variables V_1, V_2, V_3, V_4.

We now construct the graph database G. The graphs in G ensure that the vertices of a solution graph (if it exists) represent the literals of a solution to the 3SAT problem. G is partitioned into four disjoint parts: The graphs in G_S ensure that the solution graph is a subgraph of s, so that the vertices represent literals. For each variable V_i, the G_V part of the database encodes the constraint that the solution graph contains either the vertex v_i or $v_{\neg i}$, while the G_D part guarantees that it does not contain both, v_i and $v_{\neg i}$. Finally, the graphs in G_C ensure that the truth value assignment induced by the solution graph satisfies all clauses of S. To specify the graphs in G, we describe how each graph g_i in the database is derived from s. First of all, for the indices $i \in R_{-1}$, we set the g_i to subgraphs of s, where specific vertices and their incident edges are removed. If one removes vertices $v_{x_1}, v_{x_2}, \ldots, v_{x_p}$ and the corresponding incident edges from s, we denote the resulting graph by $\delta(s, x_1, x_2, \ldots, x_p)$. For instance $\delta(s, 1, \neg 2, 3)$ is the graph s with the vertices $v_1, v_{\neg 2}$, and v_3 removed. Recall that there are $k + l$ indices in R_{-1}. For the first k indices we set the corresponding graphs g_i to $\delta(s, v_{x_1}, v_{x_2}, v_{x_3})$, where x_1, x_2 and x_3 are the literals of the corresponding clause of S. For example, if S contains the clause $(V_1 \wedge \neg V_2 \wedge V_3)$, G contains the graph $\delta(s, 1, \neg 2, 3)$, that is, the star graph s with the vertices $v_1, v_{\neg 2}, v_3$ removed. This graph is illustrated in Figure 6.4 (b). This scheme describes the form of the graphs with the first k indices in R_{-1}. Let G_C denote the set of these k graphs. For each of the l remaining indices, we set the corresponding graph to $\delta(s, i, \neg i)$, where $1 \leq i \leq l$. In other words, we set the ith of the remaining graphs to s with the two vertices v_i and $v_{\neg i}$ removed. We call the set of graphs that are built in this way G_V. Figure 6.4 (c) depicts the graph $\delta(s, 1, \neg 1) \in G_V$.

For the indices in R_{+1}, we also distinguish between the first k graphs and the remaining l graphs. The first k graphs are all set to the unmodified star graph s. Let G_S denote the resulting multiset of graphs that contains only copies of s. For the remaining l graphs, we need to join two subgraphs of s by adding an edge between the two center vertices. More formally, if u and u' are subgraphs of s, then $\gamma(u, u')$ is the graph $(V_u \cup V_{u'}, E_u \cup E_{u'} \cup \{v_c, v'_c\}, l_s)$. For each variable V_i, we add the graph $\gamma(\delta(s, v_i), \delta(s, v_{\neg i}))$ to the database. As an example, Figure 6.4 (d) shows the graph corresponding to V_1. Since there are l variables, this yields exactly the remaining l graphs with indices in R_{+1}. For later reference we also keep these graphs in the set G_D.

Now, we show that solving the dispersion score problem with the training matrix X and graph database G is equivalent to solving the 3SAT problem for the CNF formula S. First of all, if the dispersion score problem has a solution, there exists a subgraph h whose instantiation vector is $t \in \{-1, +1\}^n$ with $d(t) = 0$. First of all, $t \neq -r$, because otherwise h is a subgraph of the graphs in $G_C \cup G_V$, and therefore also a subgraph of the graphs in $G_S \cup G_D$. This would mean that all components of t are $+1$, a clear contradiction to $d(t) = 0$. Apparently $t = r$ is the only possible solution. This means that the solution graph h is a subgraph of the graphs in $G_S \cup G_D$. Since G_S contains only s, h must be a subgraph of s, that is, a star with vertices expressing variable truth assignments. For an arbitrary variable V_i, G_D contains a subgraph of s which lacks v_i and a subgraph, which lacks $v_{\neg i}$, but no star, which contains both, v_i and $v_{\neg i}$. Since h must be a subgraph of all graphs in G_D, it can contain either v_i or $v_{\neg i}$, but not both for all $1 \leq i \leq l$.

On the other hand, h must not be a subgraph of any graph in $G_C \cup G_V$. Thus, h must differ from each $g \in G_C \cup G_V$ by at least one vertex, or, more precisely, for each graph $g \in G_C \cup G_V$, there must be at least one vertex v_x of s which is contained in h, but not in g. Thus, a graph in G_V enforces that h contains at least either v_i or $v_{\neg i}$ for an $i \in \{1, \ldots, l\}$. Thus, h must contain exactly one vertex per variable V_i, namely either v_i or $v_{\neg i}$. Thus, the vertices assign a truth value to each variable. Now it is easy to see that each graph g in G_C enforces that h contains at least one vertex v_x so that the literal x satisfies the corresponding clause c in S. Therefore the truth value assignment induced by h is a solution to the 3SAT problem given by S. If there is no h with $d(h) = 0$, then there is also no truth value assignment that satisfies S.

On the other hand, if $a : \{V_1, \ldots, V_l\} \to \{\text{true}, \text{false}\}$ is a truth assignment satisfying S, one can compute the graph $g_S := \delta(s, a'(V_i), \ldots, a'(V_i))$, where $a'(V_i)$ is v_i, if $a(V_i) = \text{true}$ and $v_{\neg i}$ otherwise. It is easy to see that g_S is a subgraph of the graphs in $G_S \cup G_D$, but no subgraph of the graphs in $G_C \cup G_V$. If there is no truth value assignment that satisfies S, there can also be no subgraph h that gives rise to the instantiation vector r.

Finally, it is clear that the computation of G can be done in time polynomial in k and l, because it just involves the construction of $2(k + l)$ graphs, where each graph contains at most $4l + 1$ edges. The computation of H for an arbitrary size $n = 2(k + l)$ is difficult. However, if one chooses a larger $n' \geq n$ of the form $n' \in \{2^i | i \in \mathbb{N}\}$, then Sylvester's recursive construction can be applied to generate a H of size $n' \times n'$ in polynomial time. Since the proof above remains valid for a larger H, and since there is an $n' \in \{2^i | i \in \mathbb{N}\}$ with $n < n' \leq 2n$, the computation of an H of appropriate size is possible in polynomial time. Thus, the construction procedure of X and G above constitutes a polynomially computable reduction of 3SAT to the dispersion score problem. □

There is no generally applicable approach to solving such a combinatorial optimization problem. However, in recent years stochastic local search (SLS) algorithms have been shown to be remarkably successful on similar NP-hard combinatorial problems. In particular, they are among the best algorithms available to solve hard satisfiability problems, see for example Hirsch and Kojevnikov (2005). Essentially, stochastic local search can be described as a randomized greedy walk in the space of solution candidates. As described in Section 4.2.2, an SLS algorithm starts by generating a random solution candidate. It then iterates in a loop over two steps: in the first step, it calculates "neighboring" solution candidates according to some predefined neighborhood relation. For each neighbor, it computes a score function indicating to which degree the candidate is optimal. In the second step, it randomly selects a new candidate among the neighbors with the best score.

Algorithm 9 An SLS algorithm for dispersion based feature induction. D is the graph database, *maxSteps* specifies the maximal length of search, p is the probability for random steps.

```
procedure DispersionSLS($D$)
    $c \leftarrow$ a random subgraph of $D$
    $steps \leftarrow 0$
    while $score(c) \neq 0$ and $steps < maxSteps$ do
        $steps \leftarrow steps + 1$
        $n \leftarrow$ set of general and specific neighbors of $c$
        $c \leftarrow \begin{cases} \text{with probability } p\text{:} & \text{a random graph in } n \\ \text{otherwise:} & \text{a random graph from } n \text{ that} \\ & \text{achieves the best score within } n. \end{cases}$
    end while
    return the best $c$ found so far
end procedure
```

As such a pure greedy algorithm can easily be trapped in local optima, the second step is from time to time (that is, with a predefined noise probability p) replaced by a step, where a completely random neighbor is selected as new candidate. Finally, the algorithm keeps track of the best candidate found so far and outputs this candidate as a solution after a maximum number of iterations. While modern SLS algorithms often use more sophisticated decision functions, the basic principle has been shown to be effective on a range of NP-hard problems.

The SLS framework can be easily adjusted to the optimization problem presented in this section. A solution candidate is simply a graph, and the scoring function is the dispersion score explained in the preceding section. For the neighborhood relation we generate two different kinds of neighbors: more *specific neighbors* are built by extending the current candidate graph with an edge so that the resulting subgraph occurs in at least one graph of the graph database. This avoids generating neighbors that do not occur in the database at all. More *general neighbors* are built by removing one edge from the current candidate. If the removal of the edge separates the graph into two unconnected components, we keep the larger of the two as neighbor and discard the smaller one. Since such a breakup can remove a large part of the current candidate, the SLS algorithm is able to proceed from one part of the candidate space to a different part in only a few steps. This ensures that the local search is not restricted to the proximity of the start candidate. Algorithm 9 gives the pseudocode for the SLS algorithm.

It is crucial for the effectivity of an SLS algorithm that the calculation of the score function and the neighbors is as fast as possible. Unfortunately, both tasks are exceptionally expensive in our case. To compute the dispersion score, one needs to identify the instantiation vector and that implies a

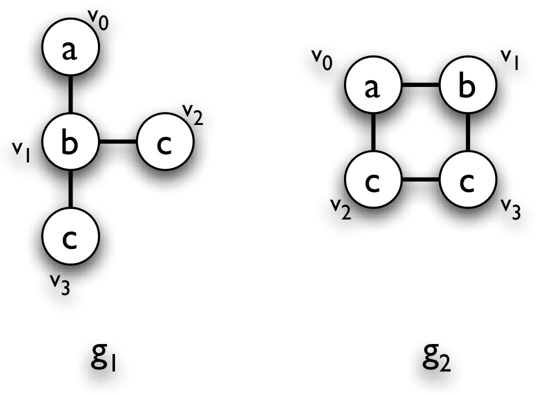

Figure 6.5: A graph database, containing two graphs g_1 and g_2. The vertices in the graphs are ordered according to a breadth-first traversal $v_0, \ldots, v_4$. These traversals give rise to the canonical code string "a(0,b,1)(1,c,2)(1,c,3)" for graph g_1 and "a(0,b,1)(0,c,2)(1,c,3)(2,c,3)" for graph g_2.

subgraph isomorphism test with each graph in the database. While computing more general neighbors is straightforward, identifying more specific neighbors involves finding the occurrences of the current candidate in the database and to collect the edges that extend those occurrences. Again, this requires a subgraph isomorphism test of the candidate with each graph in the database. To overcome this performance bottleneck, we pre-compute an index structure that stores all subgraphs up to a maximum size that occur in the database. The calculation of a candidate's instantiation vector and more specific neighbors is then just a lookup or a limited search operation in the index structure. As there is a huge number of subgraphs in a typical database, it is important to design the index structure to be space efficient, yet fast to access.

We achieve this by associating each (sub-)graph with a *canonical code string*, that is, a string that uniquely determines the graph, and storing those canonical strings in a graph trie. More precisely, let $G = (V, E, l)$ be a graph, where $V = \{v_1, \ldots, v_n\}$ is a set of vertices, $E \subseteq \{\{a, b\} \mid a, b \in V, a \neq b\}$ the set of edges, and $l : V \cup E \to L$ is a function assigning labels to the vertices and edges. Typically, a program stores the vertices of such a graph in arbitrary order and uses pointers to encode the edges. In this way, each graph can be represented in many different ways and looking up a particular graph in a list of graphs involves expensive graph isomomorphism tests. To avoid this overhead, we compute a unique representation of each graph and store only this canonical string. We follow the scheme by Borgelt (2005) and compute a graph's canonical string as follows: first of all, we assume an order $\preceq_l$ on the vertex and edge labels. We select this order so that infrequent

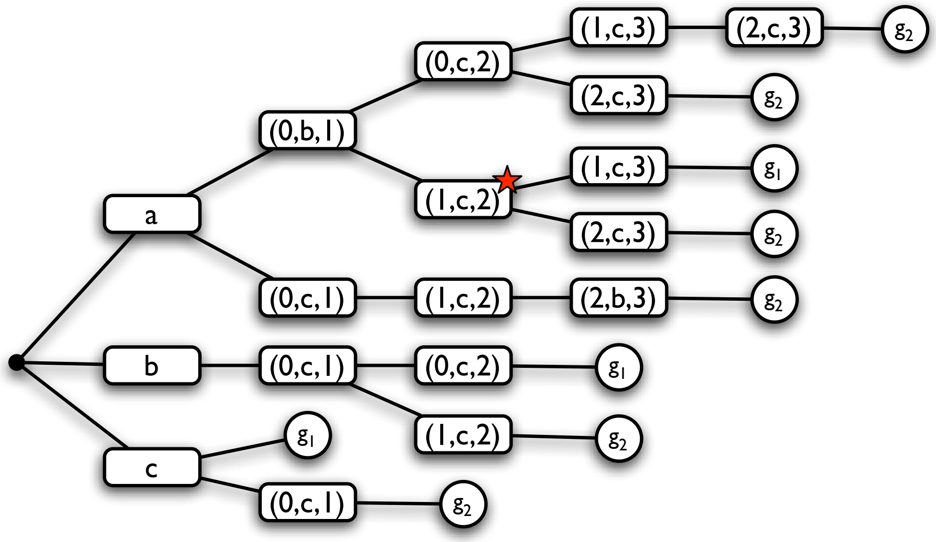

Figure 6.6: The graph trie for the database in Figure 6.5. Each node represents the canonical code string of a subgraph. The leaves below a node n are exactly the IDs of the database graphs that are supergraphs of n.

labels precede frequent ones. Then, we traverse the graph in breadth-first order starting from vertices with minimal label. This is illustrated in Figure 6.5, where a database contains two graphs g_1 and g_2. The order $v_0, \dots, v_4$, by which the two graphs are traversed is a breadth-first traversal. Of course, there are many possible traversals and the order in which a traversal visits the graph's vertices determines an index number for each vertex. Given such an index numbering scheme, an edge in the graph can be conveniently represented by a code $(i_1, l_v, i_2,)$, where i_1 is the index of the edge's start vertex, i_2 is the index of the end vertex and l_v is the end vertex's label.[4] We define the order $\preceq_c$ over such codes to be the lexicographic order with $\preceq_l$ for the labels and $\leq$ for the indices. For each traversal, a graph can now be represented by a string $l_s c_1 \dots c_m$, where l_s is the label of the start vertex and the c_i are the codes for each edge. For the canonical code string, we consider only ordered strings where $c_i \preceq_c c_j$ whenever $i \leq j$. It is easy to see that the codes in such a string are ordered according to the succession in which the traversal visits the vertices. As each traversal gives rise to a different ordered code string, we define an order $\preceq_s$ on all ordered code strings, again by taking the lexicographic order. Finally, the *canonical code string* is defined to be the minimal code string according to this order among all ordered code strings. In Figure 6.5 the two canonical code strings are

[4]For simplicity of representation, we give the code for graphs without edge labels. In the case of labeled edges, the code is (i_1, l_e, l_v, i_2), where l_e is the edge label.

given for each of the two graphs. We refer to Borgelt (2005) for an algorithm to check for canonicity.

This canonical code has a couple of nice properties. First, due to the choice of $\preceq_l$, vertices with infrequent labels appear early in the string. Thus, when comparing the code strings of two graphs highly discriminative node labels get tested early. Second, each prefix of a code string is a canonical string on its own and represents a subgraph of the original graph. This enables the use of a trie to store subgraphs efficiently. Instead of storing all subgraphs of a graph, we only store the maximal canonical subgraphs, that is, the subgraphs whose code string is not a prefix of a larger subgraph's code string. Figure 6.6 gives the graph trie which contains all subgraphs of the graph database in Figure 6.5. One can determine the instantiation of a graph g by looking up the canonical code of g in the trie and collecting all leaves below the node representing g. For instance, consider a simple graph g with three vertices labeled a, b and c and an edge between the first and the second and the second and the third vertex. Its canonical code string is "a(0,b,1)(1,c,2)". Traversing this code in the trie leads to the node marked with a star. The leaves below this node are g_1 and g_2 and these are exactly the graphs of which g is a subgraph. In the next section we present experiments with Rumble and dispersion-based feature generation.

6.5 Experiments

In the preceding sections we extended Rumble towards the multi-relational setting. To do so, we presented a framework to assess and categorize first-order learning approaches according to their use of information, and put the framework to use to derive a dispersion-based rule generation procedure that aims at small, but diverse and complementary rule sets. With this, we are in the position to evaluate the proposed methods empirically. We perform three experiments. For the first experiment, the goal is to test whether Rumble in general is competitive with existing margin-based first-order rule learning systems. The second experiment is designed to demonstrate how the framework can be applied to empirically compare and rate typical design decisions in first-order rule learners. For the third experiment, we investigate the utility of dispersion-based rule generation on three small molecule data sets from cheminformatics.

6.5.1 Experiments with Rumble on First-Order Data

Here, we compare the predictive accuracy of MMV optimization to those of other margin-based approaches. Following the discussion in Section 6.2, we set p to two and b to zero for all experiments. In a recent paper, Landwehr *et al.* (2006) present kFOIL, a version of FOIL modified to use kernels and SVMs in the rule induction step. The authors give a comparison of kFOIL

	kFOIL	nFOIL	Aleph	c-ARMR + SVM	RUMBLE
Mutagenesis	81.3%	75.4%	73.4%	73.9%	84.0%
Alzh. amine	88.8%	86.3%	70.2%	81.2%	91.1%
Alzh. toxic	89.3%	89.2%	90.9%	71.6%	91.2%
Alzh. acetyl	87.8%	81.2%	73.5%	72.4%	88.4%
Alzh. memory	80.2%	72.9%	69.3%	68.7%	83.2%
NCTRER	77.6%	78.0%	50.9%	65.1%	79.3%

Table 6.2: Predictive accuracy according to tenfold cross-validation for several systems. Four systems are compared on six data sets with MMV optimization.

with nFOIL, Aleph and the propositionalization approach c-ARMR+SVM on six data sets. Mutagenesis (Srinivasan *et al.*, 1996) is a popular ILP benchmark data set. To warrant a fair comparison we used atom and bond information only and give results for the regression friendly part, as the regression unfriendly part is too small to warrant precise estimation of predictive accuracy. On the Alzheimer data set (King and Srinivasan, 1995) the task is to predict the ranking of certain compounds with regard to four quantities that are known to influence Alzheimer's disease. Following Landwehr *et al.* (2006), we give results for amine reuptake inhibition (686 instances), low toxicity (886 examples), high acetyl cholinesterase inhibition (1,326 instances) and reversal of memory deficiency (642 instances). We used a Prolog-based bias that resembles the original GOLEM bias (Srinivasan *et al.*, 1996). The NCTRER data set deals with the prediction of binding activity of small molecules at the estrogen receptor. We used the FreeTree plugin to build substructure rules for the 232 examples. Table 6.2 gives the results for the four systems (Landwehr *et al.*, 2006) and RUMBLE. As can be seen, RUMBLE outperforms the other approaches on all data sets. For NCTRER, one can improve the predictive accuracy of RUMBLE even further to 82.76% by setting p to 1.5.

A different approach to relational margin based learning has been taken by Muggleton *et al.* (2005), where the output of CProgol5.0 is used in a linear kernel SVM. The authors compare their algorithm with partial least squares, multi-instance kernels and an RBF kernel on three chemical features on the DSSTox data set (576 instances). The goal is to predict whether or not the toxicity of a compound is above average. We used a Prolog refinement operator to generate 300 rules that check for the existence of substructures in the molecules that appear at least three times in the data set. Then, we apply a Meta refinement operator that calculates all pairs of the first 100 existing rules and combines them disjunctively. The results in Table 6.3 show that RUMBLE outperforms SVILP on this data set. The results

Method	Pred. Acc.
CProgol5.0	55%
CHEM	58%
PLS	71%
MIK	60%
SVILP	73%
Rumble	76%

Table 6.3: Predictive accuracy according to tenfold cross-validation for several systems. Five systems are compared on the DSSTox data set.

demonstrate that margins are useful in first-order learning without the need for kernels.

6.5.2 An Empirical Investigation of Rule Learning in the Framework

The framework outlined in Section 6.3 allows to rate a relational learning system according to the information it is using to generate queries and classifiers. With this, we have a convenient method to compare different learning systems and make justified statements about the contributions of certain design decisions. For example, we could answer questions such as: On this particular kind of data, does it make more sense to generate queries of form A or form B? Is it worthwhile to keep all the features or will filtering those features, which meet condition C, help overfitting avoidance? If the goal is to pass the informative queries earlier to the learning system than non-informative queries, should one sort the generated queries according to sorting order E or F? Each of these questions could be answered by keeping parts of the learning system fixed while varying only the part under investigation. Other experiments could shed light on the interdependence between certain design decisions. For instance, one could ask: What is the best sorting order for each of three different feature generation methods? Which learning algorithms works particularly well with certain filters? Those questions are often hard to answer when comparing existing methods, because most existing systems are built in an integrated fashion so that it is difficult to rate the contribution of single design decisions.

In the following we perform two studies. In the first, we primarily compare different query generation procedures while keeping the filter and sorting order fixed. In the second study, we keep the query generation and filter stage fixed and use different sorting criteria to investigate which sorting order works best. We deal with two data sets where the goal is to predict the biological activity of small molecules, given as molecular graphs. The

Data Set & Number of Queries		Agnostic		Minimum Frequency		Dispersion		Dispersion + Class Corr.	
		Trg	Tst	Trg	Tst	Trg	Tst	Trg	Tst
	20	45.2	45.7	56.4	53.2	66.2	58.9	72.9	64.9
	40	46.6	45.7	61.8	55.5	75.8	66.4	78.3	67.5
	60	51.1	47.9	63.3	55.8	78.2	68.3	80.4	66.0
Yoshida	80	53.2	50.2	64.6	57.4	79.8	68.7	80.0	67.5
	100	53.7	49.8	66.4	56.2	81.9	68.3	81.4	67.5
	150	53.8	48.7	70.5	57.0	83.6	68.3	82.5	63.4
	200	54.9	50.2	73.4	60.0	85.1	66.0	82.8	63.4
	20	61.1	63.6	63.9	62.7	71.7	67.7	76.6	71.6
	40	64.4	61.2	61.2	55.7	76.8	70.8	78.4	73.5
	60	65.2	62.4	66.7	60.2	78.0	71.8	79.5	72.3
Bloodbarr	80	59.4	56.9	66.2	61.4	78.4	71.3	80.6	71.8
	100	62.4	60.7	66.1	59.0	80.7	73.5	82.0	74.0
	150	68.8	65.5	67.6	59.8	81.8	73.0	82.0	69.6
	200	70.7	66.0	74.1	64.8	84.4	72.8	83.9	73.0

Table 6.4: Training and test set accuracies for agnostic, minimum frequency, dispersion-based and dispersion with class correlation based feature generation.

yoshida data set (Yoshida and Topliss, 2000) consists of 265 molecules classified according to their bio-availability. The second data set classifies 415 molecules according to the degree to which they can cross the blood-brain barrier (Li *et al.*, 2005).

For the first study, we chose the following four query generating procedures, sorted by the amount of information that is used:

- *Agnostic.* Here we simply generate all subgraphs with up to ten edges ($g()$).
- *Based on a minimum frequency constraint.* Here, we apply a frequent subgraph mining tool to identify all subgraphs that appear in at least 6% of the graphs in the database ($g(X)$). The implementation is based on a depth-first search.
- *Based on dispersion.* Dispersion based query generation (compare Section 6.4) aims at a diverse set of queries, so that each query's instantiation is as different as possible from the instantiations of the other queries. To this end, we use the dispersion scoring function (6.2) that measures the dissimilarity (dispersion) of a query set and apply the stochastic local search algorithm described in Section 6.4 to find subgraph queries whose instantiation optimizes this score ($g(q_1, ..., q_k, X)$).

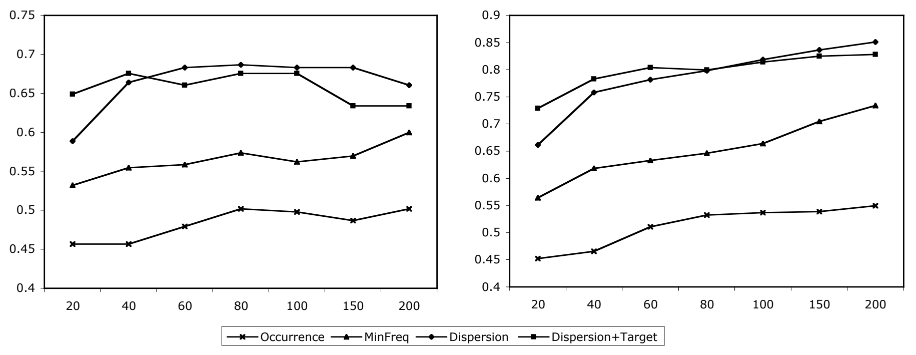

Figure 6.7: Predictive accuracy (left) and training set accuracy (right) for linear classifiers on the yoshida data set plotted against on the number of queries posed to the database.

This ensures that each new query contributes a different piece of information.

- *Based on dispersion and correlation to the target class.* This is the same as the preceding strategy, except that the scoring function is modified so that dispersion and correlation with the target contribute in equal parts ($g(q_1, ..., q_k, X, Y)$).

In each of the four cases we use a simple filter that discards all queries whose instantiations are duplicates of already existing features ($f(X)$). We do not sort the generated features, but hand them to the learning algorithm in the order they were generated. A good strategy finds relevant queries first and less informative queries only later so that the system can stop early without compromising predictive accuracy. Stopping as early as possible is desirable because it leads to fast learning systems that induce small and compact classifiers. Thus, to investigate the performance of each strategy, we generate a fixed number of queries and use those as features in a linear classifier. We apply Margin Minus Variance (MMV) with $b = 0$ and $p = 2$ to learn the linear classifier. We give the results in Table 6.4. Then, in Figure 6.7 and 6.8 we plot the predictive accuracy and the training set accuracy for each strategy on the two data sets. It can be seen that the first two strategies do not make efficient use of the information provided by the database. For both strategies, predictive accuracy and training set accuracy increase only slowly and are significantly lower than the corresponding quantities for the second two strategies. For the two dispersion based strategies there is surprisingly little difference. The correlation to the target, which is used in the fourth strategy, causes the predictive and training set accuracy to be better for

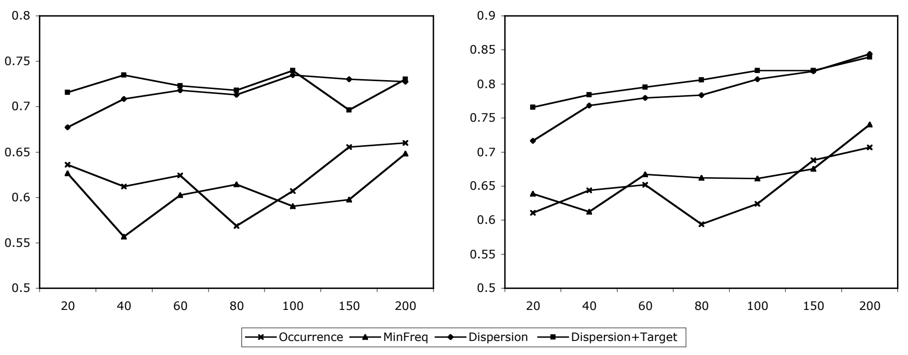

Figure 6.8: Predictive accuracy (left) and training set accuracy (right) for linear classifiers on the Blood Barrier data set plotted against the number of queries posed to the database.

small numbers of queries. However, after a certain number of queries is generated, the "dispersion only" strategy not only catches up, but also does not overfit as strongly as the fourth strategy. This indicates that – at least for the two investigated data sets – finding queries that complement each other is more important than finding queries that are informative about the target on their own. The visualization of training and test accuracy in Figure 6.7 illustrates this phenomenon on the yoshida data set.

For the second study, we keep the feature generation and filtering fixed, but investigate different sorting criteria. Sorting is particularly interesting, if one is aiming at small and comprehensible feature sets and wishes to stop query generation as early as possible. We generate all queries by mining for subgraphs that are contained in at least 6% of the database graphs ($g(X)$). The queries are then filtered by the same filter as above, that is, by discarding all features that give rise to the same instantiation as an already existing query. We investigate the following four sorting criteria:

- *By size.* Here we sort the subgraph queries according to the number of edges in the subgraph ($\preceq$). The sorting order is from few edges to many edges.
- *By balance.* In this case, queries are sorted according to how evenly the +1 and -1 are assigned in a query instantiation ($\preceq_X$). Queries, which assign +1 to the same number of instances than -1 are ranked first, while queries that assign +1 (or -1) to only a single instance are ranked last. The idea is to prefer queries whose instantiations split the training set into parts of preferably equal size, because those provide more information and can be better used to scatter the instance space.

Data Set & Number of Queries		By Size		By Balance		By Class Correlation		By a χ^2 Test	
		Trg	Tst	Trg	Tst	Trg	Tst	Trg	Tst
Yoshida	20	65.2	58.9	65.2	59.6	71.0	61.1	72.5	62.3
	40	68.6	57.0	68.5	60.0	72.7	64.5	74.1	65.3
	60	73.5	63.0	75.7	66.4	73.5	64.2	74.5	66.4
	80	74.3	61.5	78.9	65.3	75.1	67.2	75.4	67.5
	100	80.6	67.5	81.6	68.3	75.8	66.8	76.4	67.5
	150	84.9	68.7	82.8	67.5	77.1	67.5	78.7	65.3
	200	86.1	69.1	83.7	68.7	77.9	66.4	80.3	66.0
Bloodbarr	20	73.0	70.8	73.3	71.1	77.3	74.9	76.3	74.5
	40	75.3	69.9	73.7	72.5	77.4	75.9	77.3	74.7
	60	77.5	72.0	76.2	72.8	77.8	74.7	77.8	74.2
	80	80.7	74.7	78.9	73.3	78.3	72.3	78.1	73.0
	100	82.0	74.7	79.6	72.5	79.4	73.5	78.3	72.8
	150	83.9	74.9	80.0	71.8	80.1	72.8	80.0	72.3
	200	84.8	75.2	81.1	72.0	80.9	73.0	82.2	73.7

Table 6.5: Training and test set accuracies for sorting by size, by balance, by class correlation and by a class χ^2 test.

- *By class correlation.* This simply sorts the queries according to the Pearson correlation coefficient between query instantiation and target class vector ($\preceq_{X,Y}$).

- *By* χ^2. This is similar to the preceding sorting criterium, except for the use of a χ^2 test on the 2x2 contingency table instead of a simple correlation coefficient ($\preceq_{X,Y}$).

We give training and test accuracies (estimated by tenfold cross validation) in Table 6.5. Again, sorting dependent on the class information works better than the first two criteria only for a small number of features. For a large number of features (150 or 200 queries), sorting by subgraph size outperforms the other methods on both data sets with regard to training and test accuracy. It is a surprising fact that an agnostic sorting criterium that uses neither information about the training set nor the target class performs best. This demonstrates that empirical investigations similar to the two studies performed above can lead to interesting insights. In particular the visualization scheme in Figure 6.7 and 6.8 can be a valuable tool to compare existing and novel methods and biases.

6.5.3 Experiments With Dispersion-Based Rule Generation

In order to evaluate dispersion-based rule generation, we implemented the dispersion optimizing SLS algorithm and applied it to three data sets. We use the same data sets as before: The NCTRER data set (Fang *et al.*, 2001) deals with the prediction of binding activity of small molecules at the estrogen receptor. It contains 232 molecules. The yoshida data set (Yoshida and Topliss, 2000) consists of 265 molecules classified according to their bio-availability. The third data set classifies 415 molecules according to the degree to which they can cross the blood-brain barrier (BBB) (Li *et al.*, 2005). These data sets were found to be the most useful for studying overfitting and related phenomena in the previous experiments.

For the experiments, we set the noise probability of taking a purely random step in the SLS loop to 0.2, the maximum size of subgraphs stored in the graph trie to fifteen edges and the maximal number of iterations for the SLS loop to 2000. To evaluate dispersion-based feature generation independently from the MMV optimization criterion, we chose two different learning algorithms to induce linear classifiers from the resulting training sets. The first one is a support vector machine with soft margin and the C parameter set to one. We also experimented with different values for C but could not yield significantly better results. The second learner is Margin Minus Variance (MMV) optimization as described in Section 6.2. MMV optimization offers the parameter p to adjust how evenly the weights should be distributed among the features in the induced linear classifier. We set this parameter to two. As before, we build the classifier in an iterative fashion: we start with an empty feature set and then add the features one by one according to the optimal dispersion criterion. Whenever the number of features exceeds one hundred, we compute the linear classifier and remove the feature with the smallest weight before adding a new feature. The time to generate the trie is typically a few minutes. As it is generated once per data set (like an index structure in a database), it does not influence the runtimes of subsequent SLS runs.

For the first experiment, we investigate to what extent SLS with the dispersion score and the class-correlated dispersion score is able to generate training sets that are well suited for classification. To do so, we apply SLS with the two scores to construct four feature sets containing 25, 50, 150 and 300 features. We apply MMV and the SVM to induce classifiers on those training sets and report the training set accuracy and test accuracy as estimated by tenfold cross-validation in Table 6.6. The results point out some interesting insights. First of all, the dispersion score with class correlation is on average able to obtain a better *training accuracy* than the pure dispersion score, in particular with larger feature sets. This is not very surprising given the fact that the pure dispersion score does not consider the class labels. However, the improvement in training accuracy does not always

Data Set & Nr. Features		SLS (Dispersion)				SLS (Class Corr.)			
		MMV		SVM		MMV		SVM	
		Trg	Tst	Trg	Tst	Trg	Tst	Trg	Tst
Yo-shida	25	70.1	61.5	72.4	60.0	75.3	65.3	76.7	60.0
	50	75.6	64.5	79.2	62.3	78.0	67.2	80.5	66.4
	150	76.6	67.2	81.5	62.3	78.8	68.3	81.7	64.2
	300	77.2	66.4	81.5	63.8	85.9	66.4	96.1	65.7
NCT RER	25	84.2	82.8	83.1	77.6	82.6	81.5	82.9	76.3
	50	84.1	81.9	85.2	78.4	83.8	82.3	84.5	78.0
	150	84.2	81.0	84.5	77.2	84.3	82.3	86.1	82.8
	300	84.4	80.2	84.5	77.2	88.6	81.5	96.5	78.4
Blood barr	25	72.7	69.6	76.8	66.5	77.1	72.5	78.2	68.2
	50	77.5	71.3	80.0	67.2	77.5	73.5	79.5	68.4
	150	77.3	69.9	81.1	69.9	78.0	74.9	80.8	68.0
	300	77.7	71.1	81.2	70.4	84.7	73.7	95.2	74.5

Table 6.6: Results: percentage of correct classifications for SLS with the original dispersion score and SLS with class-correlated dispersion score on training and test set according to tenfold cross validation.

translate to an improvement in predictive accuracy. Generally, it does so for small feature sets up to 50 features, but for larger feature sets the difference in predictive accuracy is small even though the training accuracy is way larger for the class-correlated dispersion score. Also, MMV tends to perform better with regard to predictive accuracy than the SVM, even though its training accuracy is generally inferior to the SVM. Overall, MMV with the class-correlated dispersion score achieves good predictive performance for all feature set sizes.

As the SLS-based method should be particularly well-suited for obtaining small (for example, size 25 or 50) useful feature sets, we set up an experiment comparing it to minimum-frequency and class-correlation feature generation within this range and beyond (size 150 and 300). First, we apply a subgraph mining tool to identify all subgraphs that occur in more than six percent of the data set's graphs. Then, we sort the subgraphs by size (that is, number of edges) or by the correlation with the target according to a χ^2 test on the 2x2 contingency table. Finally we derive four feature sets with 25, 50, 150 and 300 features from those two sorted feature sequences. Hence, the first sorting order is essentially an unsupervised propositionalization approach (similar to the one by Deshpande *et al.*, 2005), while the second resembles the class-correlation based approach by Bringmann *et al.* (2006). Table 6.7 gives the training and test accuracies for MMV and the SVM. Generally, sorting

Data Set & Nr. Features		MinFreq (Sorted by Size)				MinFreq (Sorted by Corr.)			
		MMV		SVM		MMV		SVM	
		Trg	Tst	Trg	Tst	Trg	Tst	Trg	Tst
	25	65.7	58.5	68.6	60.0	70.9	61.9	71.2	60.0
Yo-	50	67.7	57.0	73.4	60.0	72.2	60.4	73.1	60.0
shida	150	80.8	66.4	91.3	68.7	73.5	64.9	76.4	60.0
	300	85.8	66.8	96.4	68.7	76.4	66.8	83.4	63.4
	25	80.2	76.3	80.7	59.9	79.2	78.4	79.6	59.9
NCT	50	83.4	79.7	84.2	69.4	80.6	80.2	79.3	59.9
RER	150	87.1	82.3	91.3	78.0	80.8	79.7	82.2	79.3
	300	87.6	80.6	92.5	75.9	81.1	79.7	82.5	77.6
	25	72.9	70.4	73.6	66.5	76.2	73.7	77.3	67.5
Blood	50	76.8	71.3	81.0	70.4	76.7	74.2	79.7	67.5
barr	150	83.2	75.7	90.0	75.9	78.4	72.0	85.6	70.1
	300	85.7	75.2	95.1	74.2	81.3	73.7	87.9	74.0

Table 6.7: Results: percentage of correct classifications for minimum frequency mining sorted by size and minimum frequency sorted by class correlation on training and test set according to tenfold cross validation.

by correlation appears to be better than sorting by size for small feature sets, but the opposite is the case for larger feature sets. On larger feature sets (150 and 300 features), the differences between dispersion-based and minimum frequency approaches are only marginal. However, in the target range of small feature sets, the SLS optimization of dispersion outperforms other approaches on two of the three data sets (yoshida and NCTRER) in almost all pairwise comparisons.

Overall, the results are competitive with those of other methods. Table 6.8 compares the presented results with those of an SVM with optimal assignment kernel (Fröhlich *et al.*, 2005) for the yoshida and bloodbarrier data sets and, for the NCTRER data set, with those of kFOIL, an extension of FOIL incorporating an SVM in a novel evaluation function (Landwehr *et al.*, 2006). Both, MMV optimization and the SVM outperform these algorithms.

6.6 Summary and Related Work

In this chapter we dealt with data in first-order representation. In order to induce predictive classifiers from such data, a learning system needs to extract the relevant pieces of information from the training set. We address this problem from two perspectives. First, we deal with the pragmatic issues and describe how RUMBLE can be extended towards the multi-relational

	Yoshida	NCTRER	Bloodbarr
kFOIL	n/a	77.6	n/a
OA Kernel	67.8	n/a	57.97
Dispersion MVV	68.3	82.3	74.9
Dispersion SVM	65.7	82.8	74.5

Table 6.8: Results: comparison of the dispersion based approach with kFOIL and a SVM with optimal assignment kernel.

setting in a modular and flexible manner (also published in Rückert and Kramer, 2006a, 2007a). This allows for a customizable and efficient way to extract meaningful information from the data. Also, using arbitrary p-norms instead of the 1-norm in RUMBLE accommodates better to the first-order setting, where the relevant information is often spread uniformly among the rules.

Margin-based first-order learning is a comparably new field of research. Popescul and Ungar (2003) extend logistic regression to the relational setting. They apply refinement operators to generate new features from relational data and add those features to a logistic regression model until the model overfits according to the Bayesian information criterion (BIC). Landwehr *et al.* (2005) propose replacing the scoring function of FOIL with a criterion based on Naive Bayes. Most of the work on margin-based multi-relational learning builds on kernel methods: Muggleton *et al.* (2005) proposed Support Vector Inductive Logic Programming, where the clauses that are induced by CProgol5.0 are used as features to represent the training data propositionally and thus allow the application of a linear SVM. This is similar to the feature construction procedure already proposed by Srinivasan and King (1999). A different approach is taken by Landwehr *et al.* (2006). They replace the rule evaluation function in FOIL with a novel scoring function, which rates the quality of a set of clauses as the accuracy of a SVM that is built on a propositional data representation, where the clauses are again used as features. Passerini *et al.* (2006) propose kernels on Prolog proof trees, whereas Woźnica *et al.* (2005) extend convolution kernels to work on relational data. The main idea is to represent each instance in the training set as a tree whose edges are connections between the tuples in different relations. This instance representation can be dealt with convolution kernels so that standard SVM methods apply.

On the theoretical side, we formulate a framework to compare methods on how well they use the information they get and to assess which kind of information is well suited to generate predictive classifiers. The framework (introduced in Rückert and Kramer, 2007c) is designed to focus primarily on the information that is actually retrieved by the learning system from the available data. This enables statistical analyses of various learning systems and design issues. In contrast, the traditional ILP framework (Nienhuys-Cheng and Wolf, 1997; Lavrač and Džeroski, 1994; Lloyd, 2003) is based primarily on (first-order) logic and deals more with the problem of finding consistent theories than with predictive accuracy. In recent years, research on statistical relational learning has attracted considerable interest. Most of the foundational work is not tailored towards classification, but the practical problems of combining logic and probability theory, see for instance Richardson and Domingos (2006) and Getoor and Taskar (2007) and the references therein. Neville *et al.* (2003) provide a short survey over recent work in statistical relational learning.

Finally, we propose dispersion-based feature generation (published also in Rückert and Kramer, 2007b). Here, the goal is to find rules that are not only correlated with the target label, but that also represent a diverse set of properties and complement each other, instead of providing redundant information. To this end, we rate feature sets using the novel dispersion score and devise an SLS algorithm to optimize this score. This is related to propositionalization (Kramer *et al.*, 2001) and general feature generation procedures such as the ones by Popescul and Ungar (2004a,b). Dispersion-based feature generation can also be viewed as a contribution to the field of feature selection (Guyon and Elisseeff, 2003). According to one of the taxonomies (Liu and Motoda, 1998), it belongs to the family of non-deterministic methods with a consistency measure. However, the non-determinism of search stems from the complexity of the pattern language and not from the combinatorial problem of choosing a suitable subset.

Chapter 7

Summary and Outlook

In the final chapter of the thesis, we summarize the main contributions and give an outlook on possible directions for further research.

7.1 Summary

In this thesis we investigated rule learning from a statistical perspective. In particular, we framed rule learning as a classification problem, where the learning system is given a training set and has to select a rule set from a predefined class of possible rule sets. Finding a "good" rule set is a non-trivial task for many reasons. First of all, one can rate a rule learner's performance according to various criteria. Among the most popular ones are predictive accuracy, simplicity and time complexity. In our studies we generally focus on predictive accuracy as the most important goal and aim for simple rule sets and scalable algorithms as second criteria. Unfortunately, there is no simple scheme to achieve good predictive accuracy in all settings. From a statistical perspective finding predictive rule sets depends on three things: first of all, a learner has to select a rule set from the hypothesis space that explains the training set well. This empirical risk minimization is a NP-hard problem for most interesting rule set hypothesis spaces. Second, the representation language and the hypothesis class from which the final classifier is chosen determine a learner's bias. A learner is predictive if the bias matches well with the learning setting. There is no single bias that works well in all settings, but some biases (for example, a bias towards large margins) appear to work well in most practical settings. Third, the size of the chosen hypothesis class (or, equivalently, the strength of the bias) needs to be adjusted to avoid under- and overfitting. This is called capacity control and can be done based on heuristics or through analytically derived methods.

Within this thesis, we deal with empirical risk minimization and capacity control for two rule set representations: DNF rule sets and weighted rule sets. DNF rule sets are covered in Chapter 4: First, in Section 4.2.1 we propose a randomized algorithm for empirical risk minimization for k-term DNF rule sets in the noise-free setting. Then, for the noisy setting, we propose an approach based on stochastic local search in Section 4.2.2. For capacity control, we present a structural risk minimization approach based on cross validation to estimate the structural risk in Section 4.3. The resulting system, SL^2 aims explicitly at predictive, but small rule sets and is therefore well suited as a benchmark system. In Section 4.3.2 we investigate how such a pure bias compares to existing rule learners. It turns out that SL^2 achieves the same level of predictive accuracy, but induces smaller and simpler rule sets.

It is well known that ensemble methods such as bagging are quite successful at improving the predictive accuracy of rule learning systems. Thus, in Section 4.4 we take an ensemble-based approach to capacity control for DNF rule learning. Instead of selecting a fixed hypothesis class size, we build a weighted ensemble of rule sets with varying sizes. This approach resembles Random Forests and proves to be on par with bagged PART. If one allows abstaining classifiers, a modified PAC-Bayesian theorem can be used to gain tight upper-bounds of the prediction error.

For weighted rule sets, we investigate three different margin-based optimization criteria in Section 5.2. It turns out that Margin Minus Variance performs best. For capacity control, we modify the Rademacher and PAC-Bayesian bounds to provide structural risk estimates in terms of margin and MMV instead of the discrete error measure. In Section 5.3.3 we also present novel bounds that depend only on the size of the rule repository. Finally, in Section 5.4 we put together the described building blocks to implement the new weighted rule learning system RUMBLE. The experiments in Section 5.5 indicate that RUMBLE is competitive with PART and SLIPPER, and outperforms an 1-norm SVM.

Finally, we turn to data in first-order representation. After extending RUMBLE to the multi-relational setting by adding a modular rule generation system, we deal with the fundamental issues encountered in first-order rule learning. In Section 6.3 we describe a theoretical framework to assess and compare rule learning systems according to the way they extract information from the available data. Motivated by this framework we present a novel rule generation approach for graph-structured data based on stochastic local search optimizing the dispersion score. Experiments indicate that RUMBLE outperforms other margin-based multi-relational learners and that the framework can be used to gain interesting empirical results. Furthermore, dispersion-based rule generation appears to induce compact and simple, yet diverse and predictive rule sets.

7.2 Outlook

In this thesis we tackled the challenges in rule learning using the tools and concepts from statistical machine learning. Rule learning and statistical machine learning are broad and lively research areas, and current research in these fields continues to contribute new and relevant results. It is therefore impossible to cover the statistical aspects of rule learning in an extensive or even detailed manner. We therefore concentrated on algorithms for empirical risk minimization and on methods for capacity control. While simplicity and time complexity were important considerations, we focused on predictive accuracy as the main objective. Of course, this choice is somewhat subjective, and research on simplicity and time complexity of rule learning remains an interesting area.

But even when one is only concerned with the statistical issues in predictive rule learning, there are lots of poorly understood issues and promising directions for further research. From a practical perspective, an exceptionally hard, but important challenge is to better understand and deal with the bias (or approximation) part of the prediction error. Since this part depends on a learner's bias, the actual question is how to select a suitable learning bias when given a particular learning problem. Rule learning is notably well positioned to contribute to this question, because the rule generation process can be adjusted in a flexible way, so that different biases can be easily compared and analyzed. The variance (or estimation) part of the prediction error is better understood than the bias part. Still, there is no silver bullet, and the advantages and disadvantages of certain methods for capacity control are not as well understood as one would like.

When it comes to the issues and methods presented in this thesis, it is easy to find unanswered questions and promising directions for further research. For instance, the SL^2 system presented in Section 4.3 uses a time-consuming and rather coarse scheme for capacity control. It would be interesting to evaluate more fine-grained approaches (possibly based on the number of literals rather than rules) with better time complexity. Using ensemble methods instead of structural risk minimization has been proven to improve predictive accuracy in Section 4.4, albeit at the expense of generating large and incomprehensible classifiers. Research on the extraction of relevant information from these ensembles or on the generation of small and comprehensible, yet predictive ensembles would be certainly most fruitful. While we dealt with DNF rule sets and weighted rule sets, we did not treat decision lists. Decision lists are simpler than weighted rule sets, but harder to comprehend than DNF formulae. On the other hand, they lack the concept of a margin and are therefore somewhat less robust against noise when compared to weighted rule sets. Still, they seem to be more robust than DNF rule sets. Hence, in many regards decision lists appear to be an interesting compromise between DNF and weighted rule sets.

Our empirical study on margin-based empirical risk minimization for weighted rule sets in Section 5.2 was limited to only a few approaches. While MMV performed well, other optimization criteria based on second (or even higher order) moments might work equally well or even better. For instance, margin divided by variance can be framed as a convex optimization problem, but it appears to be harder to analyze analytically. The bounds in Section 5.3.3 provide an elegant way to perform capacity control, but they depend only on the number of rules and ignore other relevant information. It would be interesting to extend them to also accommodate for inter-feature correlation, sparsity of the weight vector, or the use of information about the class labels in the rule generation procedure. As mentioned above, Rumble's rule generation procedure allows for a flexible way to adjust the bias. While our experiments give first indications, a more extensive study on which rule generation bias works well under which circumstances could lead to practically very relevant results.

Our investigation of first-order rule learning in Chapter 6 is probably the part of the thesis which gives rise to the most opportunities for further research. The framework in Section 6.3 provides an elegant way to categorize and assess multi-relational learning systems, but its real value depends on how well it facilitates the derivation of novel and practically relevant results. We gave two examples of how the framework can provide interesting empirical results, but while we demonstrated its practical utility, its theoretical foundation could certainly be investigated further. Various extensions are conceivable, for instance, a more quantitative measurement of the used information in bits, or an investigation of exploration versus exploitation when extracting information from the input space. Similar considerations can be made about the dispersion-based approach to feature generation. Also, the use of stochastic local search is by no means the only way to optimize for diverse rule sets. Approximation schemes or randomized algorithms might prove to work better and faster. Finally, it would be interesting to extend dispersion-based rule generation from graph-structured data towards more general first-order representations.

In any case, we hope that the results in this thesis prove to be useful for further work and encourage future research on statistical rule learning.

Bibliography

Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *Proceedings of the Twentieth International Conference on Very Large Data Bases, VLDB*, pages 487–499. Morgan Kaufmann, 12–15 1994.

Grant Anderson and Bernhard Pfahringer. Random relational rules. In S. Muggleton and R. Otero, editors, *ILP '06, Sixteenth International Conference on Inductive Logic Programming, Short Papers, Santiago de Compostela, Spain, August 2006*, pages 10–12, Santiago de Compostela, Spain, 2006.

Cosimo Anglano, Attilio Giordana, Giuseppe Lo Bello, and Lorenza Saitta. An experimental evaluation of coevolutive concept learning. In *ICML '98: Proceedings of the Fifteenth International Conference on Machine Learning*, pages 19–27, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

Dana Angluin and Philip Laird. Learning from noisy examples. *Machine Learning*, 2(4):343–370, 1988.

Martin Anthony, Graham Brightwell, and John Shawe-Taylor. On specifying boolean functions by labelled examples. *Discrete Applied Mathematics*, 61(1):1–25, 1995.

A. Asuncion and D.J. Newman. UCI machine learning repository [`http://www.ics.uci.edu/~mlearn/MLRepository.html`], 2007.

Maria-Florina Balcan and Avrim Blum. On a theory of learning with similarity functions. In William W. Cohen and Andrew Moore, editors, *Machine Learning, Proceedings of the Twenty-Third International Conference (ICML 2006), Pittsburgh, Pennsylvania, USA, June 25-29, 2006*, pages 73–80. ACM, 2006.

Peter L. Bartlett and Shahar Mendelson. Rademacher and gaussian complexities: risk bounds and structural results. *Journal of Machine Learning Research*, 3:463–482, 2003.

Peter L. Bartlett, Stéphane Boucheron, and Gábor Lugosi. Model selection and error estimation. *Machine Learning*, 48(1-3):85–113, 2002.

Peter L. Bartlett, Peter J. Bickel, Peter Bühlmann, Yoav Freund, Jerome Friedman, Trevor Hastie, Wenxin Jiang, Michael J. Jordan, Vladimir Koltchinskii, Gabor Lugosi, Jon D. McAuliffe, Yaacov Ritov, Saharan Rosset, Robert E. Schapire, Robert Tibshirani, Nicolas Vayatis, Bin Yu, Tong Zhang, and Ji Zhu. Discussions of boosting papers, and rejoinders. *The Annals of Statistics*, 32(1):85–134, 2004.

Peter L. Bartlett, Olivier Bousquet, and Shahar Mendelson. Local rademacher complexities. *The Annals of Statistics*, 33(4):1497–1537, 08 2005.

Shai Ben-David, Nadav Eiron, and Philip M. Long. On the difficulty of approximately maximizing agreements. *Journal of Computer and System Sciences*, 66(3):496–514, 2003.

Gilles Blanchard and François Fleuret. Occam's hammer. In Nader H. Bshouty and Claudio Gentile, editors, *Learning Theory, 20th Annual Conference on Learning Theory, COLT 2007, San Diego, CA, USA, June 13-15, 2007, Proceedings*, volume 4539 of *Lecture Notes in Computer Science*, pages 112–126. Springer, 2007.

Hendrik Blockeel and Luc De Raedt. Top-down induction of first order logical decision trees. *Artificial Intelligence*, 101(1-2):285–297, June 1998.

Marko Bohanec and Ivan Bratko. Trading accuracy for simplicity in decision trees. *Machine Learning*, 15(3):223–250, 1994.

Christian Borgelt. On canonical forms for frequent graph mining. In *Proceedings of the Third International Workshop on Mining Graphs, Trees, and Sequences*, pages 1–12, 2005.

Henrik Boström. Covering vs. divide-and-conquer for top-down induction of logic programs. In C.S. Mellish, editor, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1194–1200, Montreal, Canada, 1995. Morgan Kaufmann.

Paul S. Bradley and Olvi L. Mangasarian. Feature selection via concave minimization and support vector machines. In Jude W. Shavlik, editor, *Proceedings of the Fifteenth International Conference on Machine Learning (ICML 1998), Madison, Wisconson, USA, July 24-27, 1998*, pages 82–90. Morgan Kaufmann, 1998.

Ivan Bratko. Refining complete hypotheses in ilp. In Saso Dzeroski and Peter A. Flach, editors, *Inductive Logic Programming, 9th International*

Workshop, ILP-99, Bled, Slovenia, June 24-27, 1999, Proceedings, volume 1634 of *Lecture Notes in Computer Science*, pages 44–55. Springer, 1999.

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

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

Björn Bringmann, Albrecht Zimmermann, Luc De Raedt, and Siegfried Nijssen. Don't be afraid of simpler patterns. In Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou, editors, *Proceedings of the Tenth PKDD*, volume 4213 of *Lecture Notes in Computer Science*, pages 55–66. Springer, 2006.

Clifford A. Brunk and Michael J. Pazzani. An investigation of noise-tolerant relational concept learning algorithms. In L. Birnbaum and G. Collins, editors, *Proceedings of the Eighth International Workshop on Machine Learning*, pages 389–393. Morgan Kaufmann, 1991.

Peter Bühlmann and Bin Yu. Explaining bagging. Technical Report 92, Seminar für Statistik, ETH Zürich, 2000.

Jadzia Cendrowska. Prism: An algorithm for inducing modular rules. *International Journal of Man-Machine Studies*, 27(4):349–370, 1987.

Olivier Chapelle, Bernhard Schölkopf, and Alexander Zien. *Semi-Supervised Learning.* MIT Press, Cambridge, MA, 2006.

Michael Chisholm and Prasad Tadepalli. Learning decision rules by randomized iterative local search. In *Machine Learning, Proceedings of the Nineteenth International Conference (ICML 2002)*, pages 75–82. Morgan Kaufmann, 2002.

C. Chow. On optimum recognition error and reject tradeoff. *IEEE Transactions on Information Theory*, 16(1):41–46, 1970.

Peter Clark and Tim Niblett. The cn2 induction algorithm. *Machine Learning*, 3:261–283, 1989.

William W. Cohen and Yoram Singer. A simple, fast, and effective rule learner. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, pages 335–342. AAAI Press, 1999.

William W. Cohen. Fast effective rule induction. In Armand Prieditis and Stuart Russell, editors, *Proceedings of the Twelfth International Conference on Machine Learning (ICML 1995)*, pages 115–123. Morgan Kaufmann, 9–12, 1995.

Michael Collins, Robert E. Schapire, and Yoram Singer. Logistic regression, adaboost and bregman distances. In *COLT '00: Proceedings of the Thirteenth Annual Conference on Computational Learning Theory*, pages 158–169, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

Vitor Santos Costa and Ricardo Lopes. Yet Another Prolog [`http://www.ncc.up.pt/~vsc/Yap/`], July 2007.

Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods.* Cambridge University Press, Cambridge, 2000.

B. V. Dasarathy, editor. *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques.* IEEE Computer Society Press, Los Alamitos, CA, 1991.

Luc Dehaspe and Hannu Toivonen. Discovery of frequent datalog patterns. *Data Mining and Knowledge Discovery*, 3(1):7–36, 1999.

Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, January 2006.

Mukund Deshpande, Michihiro Kuramochi, Nikil Wale, and George Karypis. Frequent substructure-based approaches for classifying chemical compounds. *IEEE Transactions on Knowledge and Data Engineering*, 17(8):1036–1050, 2005.

Luc Devroye, Laszlo Györfi, and Gabor Lugosi. *A Probabilistic Theory of Pattern Recognition (Stochastic Modelling and Applied Probability).* Springer, New York, February 1996.

Luc Devroye. Necessary and sufficient conditions for the pointwise convergence of nearest neighbor regression function estimates. *Zeitschrift für Wahrscheinlichkeitstheorie und verwandte Gebiete*, 61(4):467–481, 1982.

A. Miguel Dias. CxProlog [`http://ctp.di.fct.unl.pt/~amd/cxprolog/`], October 2006.

Federico Divina and Elena Marchiori. Evolutionary concept learning. In William B. Langdon, Erick Cantú-Paz, Keith E. Mathias, Rajkumar Roy, David Davis, Riccardo Poli, Karthik Balakrishnan, Vasant Honavar, Günter Rudolph, Joachim Wegener, Larry Bull, Mitchell A. Potter, Alan C. Schultz, Julian F. Miller, Edmund K. Burke, and Natasa Jonoska, editors, *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference, New York, USA, 9-13 July 2002*, pages 343–350. Morgan Kaufmann, 2002.

Federico Divina. Evolutionary concept learning in first order logic: An overview. *AI Communications*, 19(1):13–33, 2006.

Pedro Domingos. The RISE system: Conquering without separating. In *Sixth International Conference on Tools with Artificial Intelligence*, pages 704–707, New Orleans, Louisiana, USA, 1994.

Pedro Domingos. Unifying instance-based and rule-based induction. *Machine Learning*, 24(2):141–168, 1996.

Pedro Domingos. The role of Occam's razor in knowledge discovery. *Data Mining and Knowledge Discovery*, 3(4):409–425, 1999.

Pedro Domingos. Bayesian averaging of classifiers and the overfitting problem. In Pat Langley, editor, *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000), Stanford University, Standord, CA, USA, June 29 - July 2, 2000*, pages 223–230. Morgan Kaufmann, 2000.

Pedro Domingos. A unified bias-variance decomposition for zero-one and squared loss. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on on Innovative Applications of Artificial Intelligence, July 30 - August 3, 2000, Austin, Texas, USA.*, pages 564–569. AAAI Press / The MIT Press, 2000.

Andrzej Ehrenfeucht, David Haussler, Michael Kearns, and Leslie Valiant. A general lower bound on the number of examples needed for learning. In *COLT '88: Proceedings of the first annual workshop on Computational learning theory*, pages 139–154, San Francisco, CA, USA, 1988. Morgan Kaufmann Publishers Inc.

Thomas Eiter, Toshihide Ibaraki, and Kazuhisha Makino. Decision lists and related boolean functions. *Theoretical Computer Science*, 270(1-2):493–524, 2002.

Hong Fang, Weida Tong, Leming M. Shi, Robert Blair, Roger Perkins, William Branham, Bruce S. Hass, Qian Xie, Stacy L. Dial, Carrie L. Moland, and Daniel M. Sheehan. Structure-activity relationships for a large diverse set of natural, synthetic, and environmental estrogens. *Chemical Research in Toxicology*, 14(3):280–294, 2001.

Ronald A. Fisher. The use of multiple measurements in taxonomic problems. *The Annals of Eugenics*, 7(2):179–188, 1936.

Eibe Frank and Ian H. Witten. Generating accurate rule sets without global optimization. In Jude W. Shavlik, editor, *Proceedings of the Fifteenth International Conference on Machine Learning (ICML 1998), Madison,* *Wisconson, USA, July 24-27, 1998*, pages 144–151. Morgan Kaufmann, 1998.

Yoav Freund and Robert E. Schapire. Game theory, on-line prediction and boosting. In *COLT '96: Proceedings of the Ninth Annual Conference on Computational Learning Theory*, pages 325–332, New York, NY, USA, 1996. ACM Press.

Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

Yoav Freund, Yishay Mansour, and Robert E. Schapire. Generalization bounds for averaged classifiers. *Annals of Statistics*, 32(4):1698–1722, 2004.

Caroline C. Friedel, Ulrich Rückert, and Stefan Kramer. Cost curves for abstaining classifiers. In *Third Workshop on ROC Analysis in ML (ROCML-2006), Pittsburgh, USA, 29 June, 2006*, 2006.

Caroline C. Friedel. On abstaining classifiers. Master's thesis, Ludwig-Maximilians-Universität München, 2005. [`http://www.bio.ifi.lmu.de/~friedel/material/OnAbstainingClassifiers.pdf`].

Jerome Friedman and Peter Hall. On bagging and nonlinear estimation. Technical report, Department of Statistics, Stanford University, 2000.

Jerome Friedman and Bogdan Popescu. Importance sampled learning ensembles. Technical report, Stanford University, Department of Statistics, 2003.

Jerome Friedman and Bogdan Popescu. Predictive learning via rule ensembles. Technical report, Stanford University, 2005.

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Additive logistic regression: a statistical view of boosting. *The Annals of Statistics*, 28(2):337–374, 2000.

Holger Fröhlich, Jörg K. Wegner, Florian Sieker, and Andreas Zell. Optimal assignment kernels for attributed molecular graphs. In Luc De Raedt and Stefan Wrobel, editors, *Machine Learning, Proceedings of the Twenty-Second International Conference (ICML 2005)*, pages 225–232. ACM, 2005.

Johannes Fürnkranz and Peter A. Flach. Roc 'n' rule learning - towards a better understanding of covering algorithms. *Machine Learning*, 58(1):39–77, 2005.

Johannes Fürnkranz and Gerhard Widmer. Incremental reduced error pruning. In W.W. Cohen and H. Hirsh, editors, *Machine Learning, Proceedings of the Eleventh International Conference*, pages 70–77. Morgan Kaufmann, 1994.

Johannes Fürnkranz. Pruning algorithms for rule learning. *Machine Learning*, 27(2):139–172, 1997.

Johannes Fürnkranz. Separate-and-conquer rule learning. *Artificial Intelligence Review*, 13(1):3–54, 1999.

Johannes Fürnkranz, editor. *Proceedings of the Workshop Advances in Inductive Rule Learning, ECML/PKDD 2004*, 2004.

Lise Getoor and Benjamin Taskar. *Introduction to Statistical Relational Learning (Adaptive Computation and Machine Learning).* The MIT Press, August 2007.

E. Mark Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967.

Carla P. Gomes, Bart Selman, and Henry Kautz. Boosting combinatorial search through randomization. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference, AAAI 98, IAAI 98, July 26-30, 1998, Madison, Wisconsin, USA*, pages 431–437, Menlo Park, 26–30 1998. AAAI Press.

Yves Grandvalet. Bagging equalizes influence. *Machine Learning*, 55(3):251–270, 2004.

Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003.

Trevor Hastie, Robert Tibshirani, and Jerome H. Friedman. *The Elements of Statistical Learning.* Springer, August 2001.

David Haussler. Quantifying inductive bias: AI learning algorithms and Valiant's learning framework. *Artificial Intelligence*, 36(2):177–221, 1988.

David Haussler. Decision theoretic generalizations of the PAC model for neural net and other learning applications. *Information and Computation*, 100(1):78–150, 1992.

Simon Haykin. *Neural Networks: A Comprehensive Foundation.* Prentice Hall, 1999.

Christoph Helma, Tobias Cramer, Stefan Kramer, and Luc De Raedt. Data mining and machine learning techniques for the identification of mutagenicity inducing substructures and structure activity relationships of noncongeneric compounds. *Journal of Chemical Information and Computer Sciences*, 44(4):1402–1411, 2004.

Ralf Herbrich and Thore Graepel. A PAC-Bayesian margin bound for linear classifiers. *IEEE Transactions on Information Theory*, 48(12):3140–3150, 2002.

Ralf Herbrich. *Learning Kernel Classifiers: Theory and Algorithms.* MIT Press, Cambridge, MA, USA, 2001.

Edward A. Hirsch and Arist Kojevnikov. Unitwalk: A new SAT solver that uses local search guided by unit clause elimination. *Annals of Mathematics and Artificial Intelligence*, 43(1-4):91–111, 2005.

Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58:13–30, 1963.

Klaus-U. Höffgen, Hans-U. Simon, and Kevin S. van Horn. Robust trainability of single neurons. *Journal of Computer and System Sciences*, 50(1):114–125, 1995.

John H. Holland. Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach: Volume II*, pages 593–623. Kaufmann, Los Altos, CA, 1986.

Robert C. Holte, Liane Acker, and Bruce W. Porter. Concept learning and the problem of small disjuncts. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 813–818, Detroit, MI, 1989.

Robert C. Holte. Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, 11(1):63–90, 1993.

Holger H. Hoos. *Stochastic Local Search - Methods, Models, Applications.* PhD thesis, TU Darmstadt, 1998.

David W. Hosmer and Stanley Lemeshow. *Applied logistic regression.* Wiley, New York, second edition, 2000.

Akihiro Inokuchi, Takashi Washio, and Hiroshi Motoda. Complete mining of frequent patterns from graphs: Mining graph data. *Machine Learning*, 50(3):321–354, 2003.

Ross Quinlan J. [www.rulequest.com], 2003.

Manfred Jäger. Probabilistic classifiers and the concepts they recognize. In Tom Fawcett and Nina Mishra, editors, *Machine Learning, Proceedings of the Twentieth International Conference (ICML 2003), August 21-24, 2003, Washington, DC, USA*, pages 266–273. AAAI Press, 2003.

Thorsten Joachims. Estimating the generalization performance of an svm efficiently. In Pat Langley, editor, *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000), Stanford University, Standord, CA, USA, June 29 - July 2, 2000*, pages 431–438, 2000.

Matti Kääriäinen, Tuomo Malinen, and Tapio Elomaa. Selective rademacher penalization and reduced error pruning of decision trees. *Journal of Machine Learning Research*, 5:1107–1126, 2004.

Matti Kääriäinen. Relating the Rademacher and VC bounds. Technical Report C-2004-57, Department of Computer Science, University of Helsinki, 2004.

Anil P. Kamath, Narendra K. Karmarkar, K. G. Ramakrishnan, and Mauricio G. C. Resende. A continuous approach to inductive inference. *Mathematical Programming*, 57:215–238, 1992.

Michael Kearns and Ming Li. Learning in the presence of malicious errors. *SIAM Journal on Computing*, 22(4):807–837, 1993.

Michael Kearns and Umesh V. Vazirani. *An Introduction to Computational Learning Theory.* The MIT Press, Cambridge, Massachusetts, 1994.

Michael Kearns, Yishay Mansour, Andrew Y. Ng, and Dana Ron. An experimental and theoretical comparison of model selection methods. In *COLT '95: Proceedings of the Eighth Annual Conference on Computational Learning Theory*, pages 21–30, New York, NY, USA, 1995. ACM Press.

Ross D. King and Ashwin Srinivasan. Relating chemical activity to structure: An examination of ILP successes. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):411–434, 1995.

Scott Kirkpatrick and Bart Selman. Critical behavior in the satisfiability of random Boolean expressions. *Science*, 264(5163):1297–1301, 27 1994.

Vladimir Koltchinskii. Rademacher penalties and structural risk minimization. *IEEE Transactions on Information Theory*, 47(5):1902–1914, 2001.

Stefan Kramer, Nada Lavrač, and Peter Flach. Propositionalization approaches to relational data mining. In Saso Dzeroski and Nada Lavrač, editors, *Relational Data Mining*, pages 262–291. Springer-Verlag, September 2001.

Stefan Kramer. Structural regression trees. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 812–819, Cambridge/Menlo Park, 1996. AAAI Press/MIT Press.

Mark-A. Krogel and Stefan Wrobel. Transformation-based learning using multirelational aggregation. In Céline Rouveirol and Michèle Sebag, editors, *Inductive Logic Programming, Proceedings of the Eleventh International Conference, ILP 2001, Strasbourg, France, September 9-11, 2001*, volume 2157 of *Lecture Notes in Computer Science*, pages 142–155. Springer, 2001.

Mark-A. Krogel. *On propositionalization for knowledge discovery in relational databases.* PhD thesis, Otto-von-Guericke Universität Magdeburg, 2005.

Niels Landwehr, Kristian Kersting, and Luc De Raedt. nFOIL: Integrating naïve Bayes and FOIL. In Manuela M. Veloso and Subbarao Kambhampati, editors, *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*, pages 795–800. AAAI Press / The MIT Press, 2005.

Niels Landwehr, Andrea Passerini, Luc De Raedt, and Paolo Frasconi. kFOIL: Learning simple relational kernels. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference, July 16-20, 2006, Boston, Massachusetts, USA*. AAAI Press, 2006.

John Langford. Tutorial on practical prediction theory for classification. *Journal of Machine Learning Research*, 6:273–306, 2005.

Nada Lavrač and Sašo Džeroski. *Inductive Logic Programming: Techniques and Applications.* Ellis Horwood, 1994.

Nada Lavrač, Branko Kavsek, Peter A. Flach, and Ljupco Todorovski. Subgroup discovery with CN2-SD. *Journal of Machine Learning Research*, 5:153–188, 2004.

Wenmin Li, Jiawei Han, and Jian Pei. Cmar: Accurate and efficient classification based on multiple class-association rules. In *ICDM '01: Proceedings of the 2001 IEEE International Conference on Data Mining*, pages 369–376, Washington, DC, USA, 2001. IEEE Computer Society.

Hu Li, Chun Wei Yap, Choong Yong Ung, Ying Xue, Zhi Wei Cao, and Yu Zong Chen. Effect of selection of molecular descriptors on the prediction of blood-brain barrier penetrating and nonpenetrating agents by statistical learning methods. *Journal of Chemical Information and Modeling*, 45(5):1376–1384, 2005.

Nick Littlestone and Manfred K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108:212–261, 1994.

Huan Liu and Hiroshi Motoda. *Feature Selection for Knowledge Discovery and Data Mining.* Kluwer Academic Publishers, Norwell, MA, USA, 1998.

Bing Liu, Wynne Hsu, and Yiming Ma. Integrating classification and association rule mining. In Rakesh Agrawal, Paul E. Stolorz, and Gregory Piatetsky-Shapiro, editors, *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD-98)*, pages 80–86. AAAI Press, 1998.

John W. Lloyd. *Logic for learning: learning comprehensible theories from structured data.* Springer-Verlag, 2003.

Heikki Mannila and Hannu Toivonen. Levelwise search and borders of theories in knowledge discovery. *Data Mining and Knowledge Discovery*, 1(3):241–258, 1997.

Mario Marchand and John Shawe-Taylor. Learning with the set covering machine. In Carla E. Brodley and Andrea Pohoreckyj Danyluk, editors, *Proceedings of the Eighteenth International Conference on Machine Learning (ICML 2001), Williams College, Williamstown, MA, USA, June 28 - July 1, 2001*, pages 345–352. Morgan Kaufmann, 2001.

Mario Marchand and John Shawe Taylor. The set covering machine. *Journal of Machine Learning Research*, 3:723–746, 2003.

Mario Marchand, Mohak Shah, John Shawe-Taylor, and Marina Sokolova. The set covering machine with data-dependent half-spaces. In Tom Fawcett and Nina Mishra, editors, *Machine Learning, Proceedings of the Twentieth International Conference (ICML 2003), August 21-24, 2003, Washington, DC, USA*, pages 520–527. Morgan Kaufmann, 2003.

Llew Mason, Jonathan Baxter, Peter L. Bartlett, and Marcus R. Frean. Boosting algorithms as gradient descent. In Sara A. Solla, Todd K. Leen, and Klaus-Robert Müller, editors, *Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]*, pages 512–518. The MIT Press, 1999.

David McAllester, Bart Selman, and Henry Kautz. Evidence for invariants in local search. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Innovative Applications of Artificial Intelligence Conference, AAAI 97, IAAI 97, July 27-31, 1997, Providence, Rhode Island*, pages 321–326, Providence, Rhode Island, 1997. AAAI Press / The MIT Press.

David McAllester. PAC-Bayesian model averaging. In *COLT '99: Proceedings of the Twelfth Annual Conference on Computational Learning Theory*, pages 164–170, New York, NY, USA, 1999. ACM Press.

Colin McDiarmid. On the method of bounded differences. In *Surveys in combinatorics*, volume 141 of *London Mathematical Society Lecture Note Series*, pages 148–188. Cambridge University Press, Cambridge, 1989.

Ryszard S. Michalski. On the quasi-minimal solution of the covering problem. In *Proceedings of the Fifth International Symposium on Information Processing (FCIP-69)*, volume A3 (Switching Circuits), pages 125–128, Bled, Yugoslavia, 1969.

Ryszard S. Michalski. A theory and methodology of inductive learning. *Artificial Intelligence*, 20(2):111–161, 1983.

Ryszard S. Michalski. Attributional calculus: A logic and representation language for natural induction. Technical Report MLI 04-2, Machine Learning and Inference Laboratory, George Mason University, Fairfax, VA, 2004. [`http://hdl.handle.net/1920/1487`].

Raymond J. Mooney. Encouraging experimental results on learning CNF. *Machine Learning*, 19(1):79–92, 1995.

Stephen Muggleton and Cao Feng. Efficient induction in logic programs. In S. Muggleton, editor, *Inductive Logic Programming*, pages 281–298. Academic Press, 1992.

Stephen Muggleton, Huma Lodhi, Ata Amini, and Michael J. E. Sternberg. Support vector inductive logic programming. In A. G. Hoffmann, H. Motoda, and T. Scheffer, editors, *Proceedings, Discovery Science, Eighth International Conference, DS 2005, Singapore, October 8-11, 2005*, pages 163–175. Springer, 2005.

Stephen Muggleton. Inverse entailment and Progol. *New Generation Computing*, 13(3&4):245–286, 1995.

Jennifer Neville, Matthew Rattigan, and David Jensen. Statistical relational learning: Four claims and a survey. In *Proceedings of the Workshop on Learning Statistical Models from Relational Data, Eighteenth International Joint Conference on Artificial Intelligence.*, 2003.

Shan-Hwei Nienhuys-Cheng and Ronald De Wolf. *Foundations of Inductive Logic Programming*, volume 1228 of *Lecture Notes in Artificial Intelligence.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1997.

Manfred Opper. On the annealed VC entropy for margin classifiers: a statistical mechanics study. In *Advances in kernel methods: support vector learning*, pages 117–126. MIT Press, Cambridge, MA, USA, 1999.

Aline Paes, Železný Filip, Gerson Zaverucha, David Page, and Ashwin Srinivasan. ILP through propositionalization and stochastic k-term DNF learning. In S. Muggleton, R. Otero, and A. Tamaddoni-Nezhad, editors, *Inductive Logic Programming, Proceedings of the Sixteenth International Conference, ILP 2006, Santiago de Compostela, Spain, August 2006*, volume 4455 of *Lecture Notes in Computer Science*, pages 379–393, 2007.

Andrea Passerini, Paolo Frasconi, and Luc De Raedt. Kernels on Prolog proof trees: Statistical learning in the ILP setting. *Journal of Machine Learning Research*, 7:307–342, 2006.

Ramamohan Paturi, Pavel Pudlák, Michael E. Saks, and Francis Zane. An improved exponential-time algorithm for k-sat. *Journal of the ACM*, 52(3):337–364, 2005.

Bernhard Pfahringer, Geoffrey Holmes, and Richard Kirkby. Optimizing the induction of alternating decision trees. In *Proceedings of the Fifth Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining (PAKDD2001)*, pages 477–487, 2001.

Bernhard Pfahringer, Geoffrey Holmes, and Cheng Weng. Millions of random rules. In *Workshop on Advances in Inductive Rule Learning, Fifteenth European Conference on Machine Learning (ECML 2004)*, pages 123–131, Pisa, Italy, 2004.

Tadeusz Pietraszek. Optimizing abstaining classifiers using ROC analysis. In Luc De Raedt and Stefan Wrobel, editors, *Machine Learning, Proceedings of the Twenty-Second International Conference (ICML 2005), Bonn, Germany, August 7-11, 2005*, pages 665–672. ACM, 2005.

Leonard Pitt and Leslie G. Valiant. Computational limitations on learning from examples. *Journal of the ACM*, 35(4):965–984, 1988.

G.D. Plotkin. A further note on inductive generalization. In *Machine Intelligence*, volume 6, pages 101–124. Edinburgh University Press, 1971.

Alexandrin Popescul and Lyle H. Ungar. Statistical relational learning for link prediction. In *IJCAI03 Workshop on Learning Statistical Models from Relational Data*, 2003.

Alexandrin Popescul and Lyle H. Ungar. Cluster-based concept invention for statistical relational learning. In Won Kim, Ron Kohavi, Johannes Gehrke, and William DuMouchel, editors, *KDD '04: Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 665–670, New York, NY, USA, 2004. ACM Press.

Alexandrin Popescul and Lyle H. Ungar. Dynamic feature generation for relational learning. In *Proceedings of the Third International Workshop on Multi-Relational Mining (MRDM-2004), at the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2004.

Alexandrin Popescul, Lyle H. Ungar, Steve Lawrence, and David M. Pennock. Towards structural logistic regression: Combining relational and statistical learning. In *Proceedings of the Workshop on Multi-Relational Data Mining (MRDM-2002) at KDD-2002*, pages 130–141, Edmonton, Canada, 2002.

Ariel D. Procaccia and Jeffrey S. Rosenschein. Exact VC-dimension of monotone formulas. *Neural Information Processing — Letters and Reviews*, 10(7):165–168, July 2006. Research letter.

J. Ross Quinlan and R. Mike Cameron-Jones. Induction of logic programs: FOIL and related systems. *New Generation Computing*, 13(3 & 4):287–312, 1995.

J. Ross Quinlan. Learning logical definitions from relations. *Machine Learning*, 5(3):239–266, 1990.

J. Ross Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

J. Ross Quinlan. MDL and categorial theories (continued). In A. Prieditis and S. Russell, editors, *Proceedings of the Twelfth International Conference on Machine Learning (ICML 1995)*, pages 464–470, Lake Tahoe, CA, 1995.

J. Sunil Rao and Robert Tibshirani. The out-of-bootstrap method for model averaging and selection. Technical report, Department of Statistics, University of Toronto, 1996.

Gunnar Rätsch, Manfred K. Warmuth, Sebastian Mika, Takashi Onoda, Steven Lemm, and Klaus-Robert Müller. Barrier boosting. In Nicolò Cesa-Bianchi and Sally A. Goldman, editors, *Proceedings of the Thirteenth Annual Conference on Computational Learning Theory (COLT 2000), June 28 - July 1, 2000, Palo Alto, California*, pages 170–179, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

Matthew Richardson and Pedro Domingos. Markov logic networks. *Machine Learning*, 62(1-2):107–136, 2006.

Patricia Riddle, Richard Segal, and Oren Etzioni. Representation design and brute-force induction in a boeing manufactoring domain. *Applied Artificial Intelligence*, 8:125–147, 1994.

Ryan Rifkin and Aldebaro Klautau. In defense of one-vs-all classification. *Journal of Machine Learning Research*, 5:101–141, 2004.

Jorma Rissanen. Modelling by the shortest data description. *Automatica*, 14:465–471, 1978.

Ronald L. Rivest. Learning decision lists. *Machine Learning*, 2(3):229–246, 1987.

Dan Roth and Wen–tau Yih. Relational learning via propositional algorithms: An information extraction case study. In Bernhard Nebel, editor, *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001, Seattle, Washington, USA, August 4-10, 2001*, pages 1257–1263. Morgan Kaufmann, 2001.

Ulrich Rückert and Luc De Raedt. An experimental evaluation of simplicity in rule learning. Accepted for publication in *Artificial Intelligence*, 2007.

Ulrich Rückert and Stefan Kramer. Stochastic local search in k-term DNF learning. In Tom Fawcett and Nina Mishra, editors, *Machine Learning, Proceedings of the Twentieth International Conference (ICML 2003), August 21-24, 2003, Washington, DC, USA*, pages 648–655. AAAI Press, 2003.

Ulrich Rückert and Stefan Kramer. Frequent free tree discovery in graph data. In H. Haddad, A. Omicini, R. L. Wainwright, and L. M. Liebrock, editors, *Proceedings of the 2004 ACM Symposium on Applied Computing*, pages 564–570. ACM, 2004.

Ulrich Rückert and Stefan Kramer. Towards tight bounds for rule learning. In *Machine Learning, Proceedings of the Twenty-first International Conference (ICML 2004), Banff, Alberta, Canada, July 4-8, 2004*. ACM, 2004.

Ulrich Rückert and Stefan Kramer. Margin-based first-order rule learning. In S. Muggleton, R. Otero, and A. Tamaddoni-Nezhad, editors, *Inductive Logic Programming, Proceedings of the Sixteenth International Conference, ILP 2006, Santiago de Compostela, Spain, August 2006*, number 4455 in Lecture Notes in Artificial Intelligence, pages 46–48. Springer, 2006.

Ulrich Rückert and Stefan Kramer. A statistical approach to rule learning. In *Machine Learning, Proceedings of the Twenty-Third International Conference (ICML 2006), Pittsburgh, Pennsylvania, USA, June 25-29, 2006*, pages 785–792. ACM Press, 2006.

Ulrich Rückert and Stefan Kramer. Margin-based first-order rule learning. To appear in *Machine Learning*, 2007.

Ulrich Rückert and Stefan Kramer. Optimizing feature sets for structured data. Accepted for publication in *Machine Learning: ECML 2007, Eighteenth European Conference on Machine Learning, Warsaw, Poland, September 17-21, 2007, Proceedings*, 2007.

Ulrich Rückert and Stefan Kramer. Towards a framework for relational learning and propositionalization. Accepted for publication in *Sixth Workshop on Multi-Relational Data Mining at the Eighteenth European Conference on Machine Learning, Warsaw, Poland, September 17-21, 2007*, 2007.

Ulrich Rückert, Stefan Kramer, and Luc De Raedt. Phase transitions and stochastic local search in k-term DNF learning. In H. Toivonen T. Elomaa, H. Mannila, editor, *Machine Learning: ECML 2002, Thirteenth European Conference on Machine Learning, Helsinki, Finland, August 19-23, 2002, Proceedings*, volume 2430 of *Lecture Notes in Computer Science*, pages 405–417. Springer, 2002.

Ulrich Rückert. Machine learning in the phase transition framework. Master's thesis, Ludwig-Maximilians-Universität München, 2002. [`http://wwwkramer.in.tum.de/rueckert/da.pdf`].

Cynthia Rudin, Robert E. Schapire, and Ingrid Daubechies. Boosting based on a smooth margin. In John Shawe-Taylor and Yoram Singer, editors, *Learning Theoryro, Seventeenth Annual Conference on Learning Theory, COLT 2004, Banff, Canada, July 1-4, 2004, Proceedings*, volume 3120 of *Lecture Notes in Computer Science*, pages 502–517. Springer, 2004.

Daniil Ryabko. Pattern recognition for conditionally independent data. *Journal of Machine Learning Research*, 7:645–664, 2006.

Robert E. Schapire, Yoav Freund, Peter L. Bartlett, and Wee Sun Lee. Boosting the Margin: A New Explanation for the Effectiveness of Voting Methods. *The Annals of Statistics*, 26(5):1651–1686, 1998.

Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 5:197–227, 1990.

Robert E. Schapire. The boosting approach to machine learning: An overview. In D. D. Denison, M. H. Hansen, C. Holmes, B. Mallick, and B. Yu, editors, *Nonlinear Estimation and Classification*, pages 149–172. Springer, 2003.

Richard Segal and Oren Etzioni. Learning decision lists using homogeneous rules. In *Proceedings of the Twelfth National Conference on Artificial Intelligence, Volume 1, Seattle, WA, USA, July 31 - August 4, 1994*, pages 619–625, Menlo Park, CA, USA, 1994. American Association for Artificial Intelligence.

Bart Selman, Henry A. Kautz, and Bram Cohen. Local search strategies for satisfiability testing. In D. S. Johnson and M. A. Trick, editors, *Cliques, Coloring, satisfiability: the second DIMACS implementation challenge*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 521–532. AMS Series in Discrete Mathematics and Theortical Computer Science 26, 1996.

Stephen Frederick Smith. *A learning system based on genetic adaptive algorithms.* PhD thesis, Department of Computer Science, University of Pittsburgh, 1980.

Marina Sokolova, Mario Marchand, Nathalie Japkowicz, and John Shawe-Taylor. The decision list machine. In Suzanna Becker, Sebastian Thrun, and Klaus Obermayer, editors, *Advances in Neural Information Processing Systems 15 [Neural Information Processing Systems, NIPS 2002, December 9-14, 2002, Vancouver, British Columbia, Canada]*, pages 921–928. MIT-Press, Cambridge, MA, USA, 2003.

Ashwin Srinivasan and Ross D. King. Feature construction with inductive logic programming: A study of quantitative predictions of biological activity aided by structural attributes. *Data Mining and Knowledge Discovery*, 3(1):37–57, 1999.

Ashwin Srinivasan, Stephen Muggleton, Michael J. E. Sternberg, and Ross D. King. Theories for mutagenicity: A study in first-order and feature-based induction. *Artificial Intelligence*, 85(1-2):277–299, 1996.

Charles J. Stone. Consistent nonparametric regression. *Annals of Statistics*, 5(4):595–645, 1977.

Ljupčo Todorovski, Peter Flach, and Nada Lavrač. Predictive performance of weighted relative accuracy. In Djamel A. Zighed, Jan Komorowski, and Jan Zytkow, editors, *Principles of Data Mining and Knowledge Discovery, 4th European Conference, PKDD 2000, Lyon, France, September 13-16, 2000, Proceedings*, volume 1910 of *Lecture Notes in Computer Science*, pages 255–264. Springer, 2000.

L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, November 1984.

Vladimir Vapnik and Alexey Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2):264–280, 1971.

Vladimir Vapnik and Alexey Chervonenkis. *Theory of Pattern Recognition [in Russian].* Nauka, Moscow, 1974. (German Translation: W. Wapnik & A. Tscherwonenkis, *Theorie der Zeichenerkennung*, Akademie-Verlag, Berlin, 1979).

Vladimir Vapnik. *The nature of statistical learning theory.* Springer, New York, 1995.

Vijay V. Vazirani. *Approximation Algorithms.* Springer, March 2003.

Liwei Wang and Jufu Feng. Rademacher margin complexity. In Nader H. Bshouty and Claudio Gentile, editors, *Learning Theory, 20th Annual Conference on Learning Theory, COLT 2007, San Diego, CA, USA, June 13-15, 2007, Proceedings*, volume 4539 of *Lecture Notes in Computer Science*, pages 620–621. Springer, 2007.

Geoffrey I. Webb. Further experimental evidence against the utility of Occam's razor. *Journal of Artificial Intelligence Research*, 4:397–417, 1996.

Gary M. Weiss and Haym Hirsh. A quantitative study of small disjuncts. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pages 665–670. AAAI Press / The MIT Press, 2000.

Sholom M. Weiss and Nitin Indurkhya. Optimized rule induction. *IEEE Expert*, 8(6):61–69, 1993.

Sholom M. Weiss and Nitin Indurkhya. Lightweight rule induction. In Pat Langley, editor, *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000), Stanford University, Standord, CA, USA, June 29 - July 2, 2000*, pages 1135–1142. Morgan Kaufmann, 2000.

Sholom M. Weiss, Robert S. Galen, and Prasad Tadepalli. Maximizing the predictive value of production rules. *Artificial Intelligence*, 45(1-2):47–71, 1990.

Gerhard Widmer and Miroslav Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23(1):69–101, 1996.

Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations.* Morgan Kaufmann, 1999.

David H. Wolpert. *The Mathematics of Generalization: The Proceedings of the SFI/CNLS Workshop on Formal Approaches to Supervised Learning.* Perseus Publishing, 1994.

David H. Wolpert. The lack of a priori distinctions between learning algorithms. *Neural Computation*, 8(7):1341–1390, 1996.

Adam Woźnica, Alexandros Kalousis, and Melanie Hilario. Kernels over relational algebra structures. In Tu Bao Ho, David Cheung, and Huan Liu, editors, *Advances in Knowledge Discovery and Data Mining, Ninth Pacific-Asia Conference, PAKDD 2005, Hanoi, Vietnam, May 18-20, 2005, Proceedings*, volume 3518 of *Lecture Notes in Computer Science*, pages 588–598. Springer, 2005.

Xifeng Yan and Jiawei Han. gSpan: Graph-based substructure pattern mining. In Vipin Kumar and Shusaku Tsumoto, editors, *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM 2002), 9-12 December 2002, Maebashi City, Japan*, pages 721–724. IEEE Computer Society, 2002.

Xiaoxin Yin and Jiawei Han. Cpar: Classification based on predictive association rules. In Daniel Barbará and Chandrika Kamath, editors, *Proceedings of the Third SIAM International Conference on Data Mining, San Francisco, CA, USA, May 1-3, 2003.* SIAM, 2003.

Fumitaka Yoshida and John Topliss. QSAR model for drug human oral bioavailability. *Journal of Medicinal Chemistry*, 43:2575–2585, 2000.

Filip Železný and Nada Lavrač. Propositionalization-based relational subgroup discovery with RSD. *Machine Learning*, 62(1-2):33–63, 2006.

Filip Železný, Ashwin Srinivasan, and C. David Page. Randomised restarted search in ILP. *Machine Learning*, 64(1-3):183–208, 2006.

Ji Zhu, Saharon Rosset, Trevor Hastie, and Robert Tibshirani. 1-norm support vector machines. In Sebastian Thrun, Lawrence K. Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems 16 [Neural Information Processing Systems, NIPS 2003, December 8-13, 2003, Vancouver and Whistler, British Columbia, Canada].* MIT Press, 2004.

Xiaojin Zhu. Semi-supervised learning literature survey. Technical Report 1530, Computer Sciences, University of Wisconsin-Madison, 2005. [`http://www.cs.wisc.edu/~jerryzhu/pub/ssl_survey.pdf`].

Index