Heuristic Rule Learning

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Vom Fachbereich Informatik der
Technischen Universität Darmstadt
zur Erlangung des akademischen Grades eines
Doktor der Naturwissenschaften (Dr. rer. nat.)
genehmigte

Dissertation

von

Dipl.-Inform. Frederik Janssen
(geboren in Bad Schwalbach)

Referent:	Prof. Dr. Johannes Fürnkranz
Korreferent:	Niklas Lavesson, Ph.D. School of Computing, Blekinge Institute of Technology Karlskrona, Sweden
Tag der Einreichung:	29.08.2012
Tag der mündlichen Prüfung:	10.10.2012

D17
Darmstadt 2012

Abstract

The primary goal of the research reported in this thesis is to identify what criteria are responsible for the good performance of a heuristic rule evaluation function in a greedy top-down covering algorithm both in classification and regression. We first argue that search heuristics for inductive rule learning algorithms typically trade off consistency and coverage, and we investigate this trade-off by determining optimal parameter settings for five different parametrized heuristics for classification. In order to avoid biasing our study by known functional families, we also investigate the potential of using metalearning for obtaining alternative rule learning heuristics. The key results of this experimental study are not only practical default values for commonly used heuristics and a broad comparative evaluation of known and novel rule learning heuristics, but we also gain theoretical insights into factors that are responsible for a good performance. Additionally, we evaluate the spectrum of different search strategies to see whether separate-and-conquer rule learning algorithms are able to gain performance in terms of predictive accuracy or theory size by using more powerful search strategies like beam search or exhaustive search. Unlike previous results that demonstrated that rule learning algorithms suffer from oversearching, our work pays particular attention to the interaction between the search heuristic and the search strategy. Our results show that exhaustive search has primarily the effect of finding longer, but nevertheless more general rules than hill-climbing search.

A second objective is the design of a regression rule learning algorithm. To do so, a novel parametrized regression heuristic is introduced and its parameter is tuned in the same way as before. A new splitpoint generation method is introduced for the efficient handling of numerical attributes. We show that this metric-based algorithm performs comparable to several other regression algorithms. Furthermore, we propose a novel approach for learning regression rules by transforming the regression problem into a classification problem. The key idea is to dynamically define a region around the target value predicted by the rule, and considering all examples within that region as positive and all examples outside that region as negative. In this way, conventional rule learning heuristics may be used for inducing regression rules. Our results show that our heuristic algorithm outperforms approaches that use a static discretization of the target variable, and performs en par with other comparable rule-based approaches, albeit without reaching the performance of statistical approaches.

In the end, two case studies on real world problems are presented. The first one deals with the problem of predicting skin cancer and the second one is about decid-

ing whether or not students have to be invited to a counseling session. For reasons of interpretability, rules were perfectly suited to work with in both case studies. The results show that the derived rule-based algorithms are able to find rules that are very diverse, proved to be interesting, and are also sufficiently accurate. All experiments were performed in the SECO-Framework, a new versatile framework for heuristic rule learning, which allows for an easy configuration of a wide range of different components rule learners consist of.

Zusammenfassung

Das hauptsächliche Forschungsziel dieser Dissertation ist, Kriterien zu identifizieren, die für eine gute Performanz von heuristischen Evaluationsfunktionen in einem Greedy Top-Down Covering Algorithmus verantwortlich sind. Dies wurde sowohl für Klassifikation als auch für Regression untersucht. Zu Beginn wird argumentiert, dass Suchheuristiken für induktive Regel-Lern-Algorithmen typischerweise zwischen Konsistenz und Abdeckung abwägen. Es werden Parameter für fünf verschiedene Heuristiken für Klassifikation zur optimalen Abwägung dieser beiden Ziele bestimmt. Diese Parameterwerte sind von praktischer Relevanz da sie als Standardwerte für Regel-Lern-Algorithmen verwendet werden können. Um aber eine Beeinflussung durch bereits bekannte Funktionsfamilien auszuschließen, wird das Potential von Meta-Lernverfahren untersucht, um alternative Regel-Lern-Heuristiken zu erhalten. Hervorzuheben ist, dass theoretische Einblicke in Faktoren, die für eine gute Performanz verantwortlich sind, in beiden Studien gewonnen wurden. Des Weiteren wurde das Spektrum verschiedener Suchstrategien analysiert, um festzustellen ob Separate-and-Conquer Algorithmen von mächtigeren Suchstrategien wie z.B. einer Beam-Suche oder einer vollständigen Suche im Hinblick auf Genauigkeit oder Theoriegröße profitieren können. Im Gegensatz zu bisherigen Resultaten aus der Literatur, in welchen festgestellt wurde, dass Regel-Lern-Algorithmen am sog. „Oversearching“ leiden, wird in dieser Arbeit besonderes Augenmerk auf das Zusammenspiel zwischen der Suchheuristik sowie der Suchstrategie gelegt. Die Ergebnisse verdeutlichen, dass eine vollständige Suche den vorrangigen Effekt hat längere, aber trotzdem generellere Regeln als eine Hill-Climbing Suche zu finden.

Ein zweites Ziel dieser Dissertation ist die Erstellung eines Regel-Lern-Algorithmus für Regression. Um dieses Ziel zu erreichen wird eine parametrisierbare Regressionsheuristik entworfen und der Parameter wird in der gleichen Weise wie zuvor angepasst. Eine effiziente Methode um Splitpoints zu generieren wird ebenfalls eingeführt, da man auf aus der Klassifikation bekannte Verfahren nicht zurückgreifen kann. Es wird gezeigt, dass dieser Metrik-basierte Algorithmus vergleichbar gut wie andere Regressionsalgorithmen funktioniert, obwohl er ein deutlich simpleres Modell für die Regeln verwendet. Im Weiteren wird eine neuartige Methode präsentiert mit welcher Regressionsregeln gelernt werden können indem Regression in ein Klassifikationsproblem transformiert wird. Die Kernidee ist, dynamisch eine Region um den Vorhersagewert der Regel zu definieren, innerhalb dieser alle Beispiele als positiv zu betrachten sind und alle die außerhalb liegen als negativ. So können konventionelle Regel-Lern-Heuristiken verwendet werden, um

Regressionsregeln zu lernen. Die Experimente zeigen, dass der entwickelte heuristische Algorithmus signifikant bessere Ergebnisse als eine statische Diskretisierung liefert. Zudem ist das Verfahren ähnlich gut wie andere regelbasierte Ansätze wenn auch nicht so gut wie statistische Methoden.

Zuletzt werden zwei Fallstudien präsentiert, die auf realen Daten durchgeführt werden. Das Ziel der ersten Studie ist das Hautkrebsrisiko eines Patienten vorherzusagen. In der zweiten Studie geht es um die Entscheidung, ob Studierende entsprechend verschiedener persönlicher Daten zu einem Beratungsgespräch eingeladen werden müssen. Aus Gründen der guten Interpretierbarkeit sind Regeln optimal geeignet um diese beiden Probleme zu lösen. Die Resultate zeigen, dass die in dieser Dissertation eingeführten Regel-Lern-Algorithmen geeignet sind, Regeln zu finden die sehr unterschiedlich, die für Domänenexperten interessant und die trotzdem noch ausreichend genau sind. Alle Experimente wurden im sog. SECO-Framework durchgeführt, einem neuartigen vielseitigen Framework für heuristisches Regel-Lernen welches eine bequeme Konfiguration verschiedenster Komponenten eines Regel-Lern-Algorithmus erlaubt.

Acknowledgements

This thesis was done at the *Knowledge Engineering Group* at the TU Darmstadt. The whole research presented was supported by the *German Science Foundation (DFG)*. Without this support this work would not have been possible.

In the beginning, I would like to thank Prof. Dr. Johannes Fürnkranz for his support during my time at the *Knowledge Engineering Group*. He always had an open ear for all the problems that arose through the years. Not only in scientific topics but also in all other questions he provided a constant support. Without all the discussions with him I never would have had the opportunity to finish such a thesis. Prof. Dr. Johannes Fürnkranz also provided a fruitful environment to do research, namely the *Knowledge Engineering Group*. I am very glad that I had the opportunity to work under his supervision and I could not imagine a better support.

Secondly, I like to thank my second supervisor Associate Prof. Niklas Lavesson, PhD. Niklas and I met at the *SDM* conference in 2009 and quickly recognized that we are both interested in very similar research topics. I am glad that he accepted to review my thesis and I am thankful for all the high quality comments he made. He clearly participated in improving the thesis considerably.

I am also thankful to Prof. Dr. Mira Mezini, Prof. Dr. Chris Biemann, and Prof. Dr. Ulf Brefeld for being part of the board of examiners.

During my time at the *Knowledge Engineering Group* I had the opportunity to work with very funny but also very smart colleagues. I never had the feeling that I am not totally satisfied to be able to work in this group. We always had fun, even in times when there was a lot to do and the atmosphere was not as pleasant as usual. In particular, I like to thank Dr. Eneldo Loza Mencía for many fruitful discussions (also during several vacations), Dr. Sang-Hyeun Park for being my cigarette partner during the time I was a smoker (I quit smoking now) and the interesting conversations, Dr. Heiko Paulheim for introducing me to the Semantic Web and for writing papers with me, Jan-Nikolas Sulzmann for working on the same project as I, and Lorenz Weizsäcker for interesting discussions (from every point of view). I also wish to thank all the student workers who supported my thesis by implementing the main parts of the SeCo-Framework or by working on several other important topics. During the years Raad Bahmani, David Schuld, and Markus Zopf supported the framework and my work. Dirk große Osterhues and Kilian Kiekenap provided working hardware so that all experiments could be conducted very conveniently. I never could forget to mention the secretary Gabriele Ploch. I would like to thank her for all the nice chats during breaks and all the good tips for a whole bunch of different things.

In the end, I would like to thank my parents Uta Janssen and Peter Janssen. Without them nothing at all could have been possible in my whole life. They always supported me in every aspect a child could wish for. My father even introduced me to Computer Science in the first place. Without him I would have never found such joy in working on topics concerned with computational problems. My father and I had the chance of being part of the early days of the information age and even loaded the first programs by playing a cassette. I am very glad that my mother always was there for me and supported me with everything one can imagine. I never had to bother with anything related to my education because it was always clear that a constant support is ensured. I also like to thank my brother Arik Janssen for all the interesting discussions away from all the computer science topics. Life has much more to offer than sitting in front of a computer which my brother constantly highlighted. Walter Untermann was there for me any time I needed him, so I am also very grateful for that. I like to thank Golriz Chehrazi who supported me most of the time during my job as a PhD-guy.

Many thanks go to my girlfriend Martina Hochstatter. She never hesitated to proof-read the thesis even when I tried to convince her that this really is not necessary. The mistakes and typos she found in my thesis are uncountable as her support for me and my love for her also is.

Contents

List of Figures

List of Algorithms

List of Tables

1 Introduction

With the emergence of the Internet and computer systems that have more and more storage space, the available information also grows drastically. Nowadays, there are estimates that 15 Petabyte of new information are generated every day [74]. Clearly, the process of extracting valuable knowledge, whatever form it may have, becomes unfeasible for a human. Consequently, computer systems are necessary that are able to extract important knowledge out of the mass of information. The task of finding relationships in datasets is called *data mining*. Data mining involves methods of *machine learning*. Machine learning focuses on the design and construction of algorithms that are able to learn. The discipline of machine learning, however, is a branch of *artificial intelligence*, which dates back to 1956 when John McCarthy coined the term when he wrote a proposal for the Darthmouth Summer Research Project on Artificial Intelligence[1]. Today, there are many different subfields of machine learning [117] such as neural networks, support vector machines, or rule learning.

The latter is the main topic of this thesis. The discipline of rule learning dates back to the early days of the field of machine learning. It has various subfields such as association rule mining (see for example [1, 77, 102]), inductive logic programming (see [120, 121, 136]), and propositional rule learning. We are mostly concerned with the task of *propositional* or *attribute-value* rule learning, where the rules are learned from a single table. The most famous strategy for learning such decision rules is the so-called *separate-and-conquer* or *covering* strategy. This strategy was proposed in 1969 by Ryszard S. Michalski [113]. The discipline of rule learning had its up and downs but research on this topic never completely vanished. The main reason to foster research on models that are composed by simple, interpretable rules is that they are easy to understand. Hence, a human has the ability to comprehend what a rather incomprehensible algorithm has learned on a given set of information. This property is not evident for most of the machine learning algorithms. In fact, one might argue that only decision trees are also interpretable by humans.

Especially in domains where the experts are unfamiliar with machine learning, algorithms that yield interpretable models are beneficial. Often, humans also think in a more or less rule-based way [140] which makes it easy to compare knowledge, because, at least to some extent, it is built in a similar manner. The advantages then are that these simple models are more trustful in the eyes of an expert, that they

[1] The full proposal can be found under `http://www-formal.stanford.edu/jmc/history/dartmouth/dartmouth.html` (visited on 2011-08-22).

can be compared to the assumptions of the humans, and that faulty parts of the model can be detected easily. In this sense, rules provide an excellent testbed for interacting with fields different from machine learning.

Through the years considerable progress had been made in the field of rule learning. Various authors proposed an amazing number of different algorithms. Nevertheless, rule learning is hard to analyze from a statistical point of view due to its combinatorial complexity or the heuristic search (cf. [144]). Following from this, there are different aspects of rule learning that are still not well researched. For example, the requirements of rule learning heuristics are still not framed in a unifying way. Despite, there is some work on providing theoretical frameworks for analyzing heuristics [60]. But still, some fundamental issues of heuristics are not well known. For example, it is unclear which parameter setting works best for heuristics that feature a parameter. While there are some suggestions (cf. [180, 96, 97] for the Klösgen-Measure), a solid empirical evaluation is yet missing for other measures. Presumably, a single parameter setting that works best in all situations can also not be found because of the no free lunch theorem [179]. Essentially, the theorem states that an optimal setting always is domain-dependent. It is also an open question how heuristics behave when different search algorithms are employed. Albeit there is some work (cf. [134]), this topic needs to be further examined. For the task of regression, some heuristics were proposed but there is also a lack of clear formulations, which requirements heuristics should met in this field. How different heuristics behave when the same algorithm is used has yet to be investigated in detail. Often, only some heuristics are evaluated for a certain algorithm. A comprehensive overview still needs to be done. This thesis can be seen as an approach to answer some of these questions.

Interestingly, most of the rule learning algorithms share several concepts. For example, the majority of decision rule learning algorithms are based on the separate-and-conquer strategy. In spite of these similarities, there is no unifying framework in which, for example, different algorithms based on the same basic strategy could be implemented. Often, new work about an algorithm can be characterized as changing some of the fundamental elements of the algorithm (cf., e.g., the improvements of CN2 [20, 19]). Thus, such a unifying framework would be of great value for the rule learning community. It is also a necessary first step to ease the experiments conducted in this work.

1.1 Contributions

In this thesis the following contributions are made.

General framework for rule learning
To unify different rule learning algorithms, the SeCo-Framework for rule learning is introduced (note that SeCo stands for Separate-and-Conquer). The framework al-

lows to configure rule learning algorithms by specifying different building blocks. In this way, it is easy to interchange the components of a given algorithm. The modular design allows to implement many existing rule learning algorithms and simplifies the process of changing components of one algorithm such as the heuristic or the search algorithm. Due to an evaluation package that is included in the SECO-Framework, comparing several algorithms is also a simple task.

Broad understanding of heuristics for classification
The work reported in this thesis enriches the general understanding of the heuristic component most rule learning algorithms consist of. For some parametrized heuristics for classification, optimal parameters are suggested based on an empirical optimization. Beyond that, two entirely new heuristics are derived and compared to existing ones. It is shown that their performance is comparable to the best heuristics for classification. Another important insight is that the preference structure of those heuristics is also quite similar to existing state-of-the-art heuristics.

Rule learning algorithms for regression
For the task of regression, on the one hand, a novel heuristic to solve regression directly is derived and evaluated. On the other hand, by the usage of a reduction scheme, the performance of classification heuristics for regression is reported. Thus, two algorithms that are able to deal with numerical target attributes are developed. The evaluation shows that their performance is comparable to other regression algorithms. Besides, the limitations that come with the rule based approach to regression are also discussed in detail.

Schemes for reducing regression to classification
A new mechanism to dynamically reduce the regression problem to classification is introduced. Among the interesting observations when using the reduction is that heuristics that are known to find many rules in classification end up in considerably fewer rules when used in regression. The scheme is not restricted to rule learning but can also be employed whenever an algorithm is based on positive and negative coverage statistics (as, e.g., decision tree algorithms are).

Behavior of heuristics in different search scenarios
Most of the research about classification heuristics deals with scenarios where the search algorithm of the rule learner is fixed. The majority of systems only use a simple hill-climbing search instead of complex search algorithms. The work reported here contributes to the understanding of how the requirements of heuristics change when they are used in more complex search algorithms. The performance of algorithms that employ these elaborate search algorithms, scaled even up to a true exhaustive search, is evaluated with a special focus on the phenomenon of oversearching.

1.2 Contents

In Chapter 2 the separate-and-conquer strategy, which forms the basis for most algorithms used in this thesis, is illustrated on the basis of two simple algorithms. Separate-and-conquer rule learning is used to solve the concept learning problem which is also described. When more than two classes are present, means for reducing the data to two-class problems are discussed. The main tool for a graphical analysis of heuristics, the so-called coverage space, is introduced. Subsequently, some general concepts for the design of rule learning heuristics are summarized. Then, the different rule learning heuristics that are used in this thesis are introduced. For some of them, examples of their isometrics in coverage space are also given. The chapter concludes with a descriptionn of evaluation methods for classification and regression rule learning algorithms.

Chapter 3 gives an overview of the SECO-Framework for rule learning, a new versatile and extendable framework. The architecture and the components are described. The chapter is completed by showing how to implement some existing algorithms in the framework and by a brief description of an additional evaluation package that is part of the framework.

In Chapter 4 work on classification heuristics is reported. It starts with a summary on the research on parameter tuning of heuristics. The parameters of five heuristics are optimized. The heuristics are compared by using the coverage space framework. In the following, new heuristics are derived that are learned by a meta-learning approach. The chapter also includes an extensive empirical evaluation.

Chapter 5 deals with the search algorithm of the rule learning algorithm. Based on a previous work on that topic [134], in this chapter the experiments are extended. The focus lies on the behavior of different classification heuristics, including those tuned in the previous chapter, when the search algorithm is changed. In particular, simple hill-climbing, beam search, a true exhaustive search, and a bidirectional search are compared against each other and against state-of-the-art algorithms.

Heuristics for regression are described in Chapter 6. The chapter summarizes work on a direct adaption of the classification algorithm to regression. Consequently, a new heuristic for regression had to be designed and some additional changes to the basic algorithm are described. The new algorithm is compared against other well-known algorithms.

Chapter 7 deals with a different approach to regression. Here, a mechanism to dynamically reduce the regression problem to classification is presented. As a consequence of the reduction, the heuristics for classification from Chapter 4 can be used again. In the end of the chapter, a comparison of the two regression algorithms is given.

In Chapter 8 experiments on real-world data are shown. The main purpose here is to demonstrate that the derived algorithms also perform well on actual

data. All previous experiments were done on datasets that are publicly available in repositories. These datasets may not meet the conditions that are observed in real-world data. For this reason, two domains were picked and experiments on the derived datasets are reported.

Chapter 9 focuses on providing a unified view of separate-and-conquer rule learning algorithms. Here, all results and observations are put into context following some identified criteria a good rule learning heuristic has to fulfill.

Chapter 10 concludes the thesis. Directions for future work are also given here.

2 Inductive Rule Learning

Inductive learning is the process of deriving a general description from special cases in an empirical way. Its direct counterpart is deduction where special cases are derived from general given knowledge. Generalizations from specialized cases are constructed in the form of a model. This model can then be applied on new previously unseen cases and is able to categorize them. Often, when some form of assignments of cases to certain categories or classes is been employed in an automated way, this is referred to as *classification*. Classification is the process of constructing a definition of a class. Among others, this can be achieved by the *concept learning* approach.

Concept learning was introduced by Bruner, Goodnow, and Austin [10] in 1967 and is defined there as "the search for and listing of attributes that can be used to distinguish exemplars from non exemplars of various categories". In machine learning the exemplars and non exemplars are called *examples* or *instances*. Sometimes these two terms are distinguished. Then, an instance has no class label whereas an example has a class label, i.e., the category is known. The term category is called *class* or *label*. Whenever the class for an example is given the task is referred to as *supervised learning*. *Unsupervised learning* describes the opposite, i.e., situations where the class is unknown to the algorithm. In this thesis, we are only concerned with supervised learning problems. Essentially, in a concept learning problem, examples are given from which some belong to the concept and some do not belong to the concept. The concept is one predefined class. The exemplars (examples that belong to the concept) are called *positive examples* and the non-exemplars, i.e., those that do not belong to the concept, are called *negative examples*. Then, a learning algorithm is applied to these examples to find a description of the concept. Its output is a *model*, a *hypothesis*, or a *theory* that is, by some predefined means, able to decide whether new examples are part of the concept or not. There are many different types of models, such as decision trees [7], neural networks [41, 72, 76], statistical methods (e.g., support vector machines, e.g., [24]), or rule-based theories given in the literature. This thesis is focused on rule-based theories.

In the beginning of this chapter, some definitions and notations are presented that form the basis for the following sections. These include a definition of classification and regression. Then, separate-and-conquer rule learning is introduced. Here, the model class of rules is explained in detail. In Chapter 3 a generic algorithm that implements this strategy is explained. For this reason, the basic concepts of rule learning in this fashion are illustrated here, before a concrete

implementation is given later. In Section 2.3 different mechanisms to handle multiclass classification problems are explained. Then, methods to avoid overfitting are shown. In Section 2.5 coverage spaces are introduced which are our main means for analyzing heuristics graphically. Section 2.6 is dedicated to heuristics. Here, four different types of heuristics are shown and for some of them isometrics in coverage space are given. In the last section, evaluation methods for rule learning algorithms are introduced. The section summarizes some general evaluation methods, some that are used in classification, and some for regression. In the end, statistical tests are described.

2.1 Foundations of Machine Learning and Inductive Rule Learning

Inductive rule learning aims at building rule sets from sets of examples. Attributes are used to form an example. An *attribute* A can either be nominal or numerical. There are also other types of attributes including, e.g., hierarchical attributes. These types are not considered in this thesis. A *nominal attribute* encodes situations that can be described by adjectives. An example for a nominal attribute is "outlook" which can have values like "sunny", "overcast", or "rainy". Values of nominal attributes cannot be ordered. The values of a *numerical attribute* are numbers that are comparable. The numerical attribute "temperature" may have values in certain ranges. For example, the temperature can be defined to reside in the range $[-20, 50]$. Each instance $\boldsymbol{x}$ assigns all attributes a concrete value x_i. The attributes form the instance space or data space $\mathbb{D}$. An example for a collection of such instances can be found later in Table 2.1. Note that parts of the notation are based on [81].

$$\mathbb{D} \stackrel{\text{def}}{=} A_1 \times \cdots \times A_k$$

An instance $\boldsymbol{x}$ then is defined as given below.

$$\boldsymbol{x} \stackrel{\text{def}}{=} (x_{1,j}, \ldots, x_{k,j}) \in \mathbb{D} \tag{2.1}$$

The index j identifies the j-th value of an attribute where k is the total number of attributes.

Usually, a complete instance is formed by adding another special attribute, called the *class attribute* (cf. Equation 2.2). It encodes the label or class of the instance in label space $\mathbb{L}$. This attribute allows to assign each instance to a class or a category by adding a label $y \in \mathbb{L}$ to the example. The class attribute can be nominal (nominal class attributes are considered in Chapter 4, 5, and 8) or numerical (Chapter 6 and 7). In the first case the label space is defined by $\mathbb{L} = \{y_1, \ldots, y_u\}$, where u is the number of classes. Note that in a concept learning scenario $m = 2$, in other words it is a binary classification task. One of the two labels refers to is "part of

the concept" whereas the other label states that the example is "not part of the concept". The class attribute can also be numerical. In this case the label space changes. Consequently, it is defined by $\mathbb{L} \subseteq \mathbb{R}$ then. When the class attribute is numerical the learning task is usually referred to as regression.

Attribute values can also be missing. This means that the true value of the attribute for the given instance is unknown or cannot be determined exactly. For instance, if the values of an attribute are measured by a sensor, it may happen that in some situations the sensor fails. In these cases the attribute value cannot be retrieved and remains missing. How algorithms cope with such situations is described in Section 3.2.1.

A set of instances is called *dataset* or *database*. A dataset is defined by

$$D \stackrel{\text{def}}{=} \{(\boldsymbol{x_1}, y_1), \ldots, (\boldsymbol{x_t}, y_t)\} \subseteq \mathbb{D} \times \mathbb{L} \tag{2.2}$$

where t is the number of examples. Note that each instance now has an additional label marking it as an example, i.e., the label is known to the algorithm during the training phase. In the remainder of the thesis the target value always is the last attribute. This convention stems from *Weka* [177]. We also do not differentiate between the term instance or example any more because in the remainder of the thesis we are only dealing with examples, i.e., with instances for which the class is known.

One of the main tools we used in this thesis for the handling of datasets as well as for employing learning algorithms is *Weka* [177]. The abbreviation *Weka* stands for *Waikato Environment for Knowledge Analysis*. It is a framework where a huge number of different algorithms is implemented and it is developed at the University of Waikato[1].

Datasets as defined in Equation 2.2 can be of arbitrary size, have a different number of attributes and the attributes can be of different types. They come from various different domains that have numerous requirements. The attributes of a medical domain, e.g., are completely dissimilar to those of a dataset that encodes game playing situations. We do not focus on a particular domain, the goal of this thesis is to provide high quality algorithms for a wide variety of different domains. Note that datasets can include duplicate examples or inconsistent examples. In the latter case two examples have the same characteristics but different class values, i.e., the values of the last attribute differ.

Note that *Weka* is also able to pre-process datasets. The term *pre-processing* means that the dataset is modified before the learning starts, e.g., by removing these inconsistent examples.

Table 2.1: A sample classification task

	attribute				class attribute
example	outlook	temperature	humidity	windy	play
$\mathbf{x}_1$	sunny	75	high	FALSE	yes
$\mathbf{x}_2$	sunny	80	high	TRUE	yes
$\mathbf{x}_3$	overcast	83	medium	FALSE	no
$\mathbf{x}_4$	rainy	70	high	FALSE	no
$\mathbf{x}_5$	rainy	68	medium	FALSE	no
$\mathbf{x}_6$	sunny	65	low	TRUE	yes

2.1.1 Classification

Classification is a popular task in machine learning. In classification, a function (or a *classifier*) is induced by a learning algorithm given training examples. The class of these examples is known to the algorithm, i.e., a label y_i is present. In the end, the classifier is able to assign a class to each of the test examples for whom the class is unknown. Such a classification problem can be rather simple and solvable by a human. For example, the task of deciding whether or not someone should play golf based on weather conditions (cf. Table 2.1) can be tackled by a human. Note that this example is an excerpt of the popular weather example used in many different sources (e.g., [117] or *Weka* [177]). The dataset contains four attributes, one of them is a numerical one ("temperature" measured in Fahrenheit) whereas the others are nominal attributes. It has a binary class, thus having only two values (*yes* and *no*). The relation between the attributes or one attribute and the class is quite obvious. For example, whenever the outlook is "sunny" one should play. Nevertheless, datasets can also be quite complex, thus showing multivariate coherences, containing millions of examples, and thousands of attributes. Those classification problems can hardly be solved by a human.

In regular classification a learning algorithm is trained on a training dataset that is composed by attributes and **one nominal** class attribute. There are other cases as, e.g., *multi-target learning* where more than one class is predicted simultaneously, but these are not considered here. The output of such an algorithm is a classifier that is able to predict the class attribute for an instance where the class is unknown (a test instance). Formally, a discrete function f should be learned that maps instances to class values.

[1] For more information about *Weka* see `http://www.cs.waikato.ac.nz/ml/weka/` (visited 2012-04-24)

Table 2.2: A sample regression task

	attribute				regression value
example	outlook	play	humidity	windy	temperature
$\mathbf{x}_1$	sunny	yes	high	FALSE	75
$\mathbf{x}_2$	sunny	yes	high	TRUE	80
$\mathbf{x}_3$	overcast	no	medium	FALSE	83
$\mathbf{x}_4$	rainy	no	high	FALSE	70
$\mathbf{x}_5$	rainy	no	medium	FALSE	68
$\mathbf{x}_6$	sunny	yes	low	TRUE	65

$$f : \mathbb{D} \rightarrow \mathbb{L}$$

In the remainder of the thesis the prediction of such a function f is called $y' \in \mathbb{L}$ whereas the true value is conveniently written by $y \in \mathbb{L}$.

2.1.2 Regression

In Regression, however, the target value is numeric and therefore called *regression value* or *numerical target value*. Thus, the task switches from learning a discrete function to finding a continuous one because $\mathbb{L} \subseteq \mathbb{R}$. Hence, there is no direct way to derive a notion of positive and negative examples as known from classification. In essence, there are two ways to deal with this problem, either by

- the use of different metrics to measure the quality of the model (e.g., the *mean absolute error* (cf. Section 2.7.3) or similar ones) or by
- reducing the regression problem to classification.

In Table 2.2 a sample regression dataset is presented. It is an adaption of the dataset of Table 2.1. In essence, the attributes “temperature” and “play” were exchanged. It has four nominal attributes. The regression value is the attribute “temperature”. Usually, the class values of such a regression dataset are to a large extent disjunct. As a result, it is harder to observe a relation between the attributes and the class. Where it was possible for a human to manually find a classifier for the dataset of Table 2.1, it is nearly impossible to do so for the regression dataset. For this reason, it usually is more complicated to deal with regression problems.

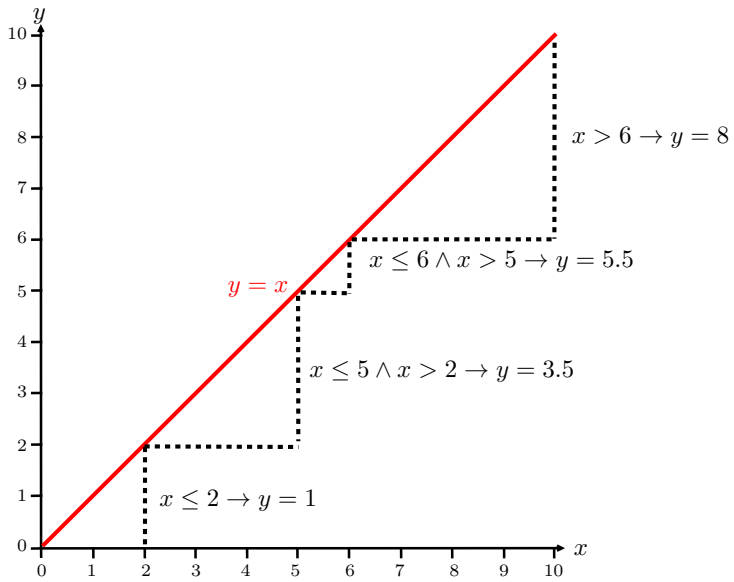


Figure 2.1: Comparison of a rule set and a linear regression

2.2 Separate-and-Conquer Rule Learning

There are many strategies to induce a set of rules. The most popular one is the so-called *separate-and-conquer* or *covering* strategy. In the following merely a brief overview of the strategy is given which is illustrated by a straight-forward separate-and-conquer algorithm. The focus is to give an overview of the separate-and-conquer strategy without paying so much attention to implementational details. Indeed, actual implementations and a detailed discussion of a framework called SeCo-Framework that makes use of this strategy are postponed here and discussed in Chapter 3.

The goal of an inductive rule learning algorithm is to automatically learn rules that allow to map the examples of a domain to their respective classes. Algorithms differ in the way they learn individual rules, but most of them employ a separate-and-conquer strategy for combining rules into a rule set [55]. This means that a rule is learned, the examples covered by the rule are removed from the dataset and the next rule is learned as long as examples are left. The origin of this strategy is the famous AQ algorithm [113], but it is still used in many algorithms, most notably Ripper [22] arguably still one of the most accurate rule learning algorithms today. The term separate-and-conquer was coined by Pagallo and Haussler [126] and stands in contrast to the *divide-and-conquer* strategy that is usually used to build decision trees [145, 143]. Where in the latter the example space is divided into

different regions, in separate-and-conquer learning this space itself is modified by removing covered examples.

Note that algorithms of that kind are used to solve the concept learning problem, i.e., they try to find a description (or namely a rule set) that is able to decide whether an example is a part of the concept or not. Consequently, they operate only on binary problems. How problems with more than two classes are tackled is described in Section 2.3.

2.2.1 Introduction of Rules

The output of such a separate-and-conquer learning algorithm are simple **if-then** rules. These rules are easy to interpret in comparison to more complex models such as support vector machines or neural networks [68]. Because of their simplicity, rules are often not as accurate as complex models because they are restricted. Consider, e.g., Figure 2.1. The target function here is $y = x$. Rules are only able to approximate this function in a discrete, piecewise way whereas, e.g., a linear regression is able to learn the exact function[2]. The figure is meant as an example, the rule set of course can be different. Nevertheless, in this setting, a rule set is never able to be as exact as more complex models. Whenever interpretability is more important than accuracy, rules are a natural choice. The implementation of learning algorithms which yield simple rules that go hand in hand with an acceptable performance is of importance since often interpretable models are favored over complex ones.

A rule consists of a *body* and a *head*. There are many different types of rules depending on what elements are used in the rule's body, how they are combined, and what the head of the rule consists of. We are concerned with *propositional* rules. A propositional rule's body is formed by attribute-value tests. Each attribute-value test is called a *condition*. For nominal attributes a condition is build by checking for equality or inequality against a value of the attribute that is present in the data ($A_i = x_{i,j}$ or $A_i \neq x_{i,j}$) and numerical attributes are compared by using $<$, $\leq$, $>$, or $\geq$. Thus, a numerical condition is of the form $A_i < v$, $A_i \leq v$, $A_i > v$, or $A_i \geq v$. The value v is a threshold. Note that v not necessarily has to be present in the dataset. Usually, it is a generated value. In the remainder, conditions are referred to by the letter g. A rule has the following generic form

$$\text{body} \rightarrow \text{head}$$

where the body is formed by a conjunction of conditions and the head is a single condition setting the class to one of the class labels. It is to be read as "*if* body *then* head". Given that all conditions that are present in the rule are either equal to the value of the example or suffice the comparators given above for numerical

[2] The linear regression model would be $y' = \beta \cdot x + \epsilon$, where $\beta = 1$ and $\epsilon = 0$

attributes, the rule is said to *cover* the example. Note that there are also other types of rules. For example, the conditions of a rule can also be combined in a disjunctive way by a logical **OR** ($\vee$). Then, the rule covers an example as soon as one condition is true. Another example is a rule that has more than one condition in the head. These rules predict more than one class and are typically used in association rule mining [129, 1].

As noted above, we are only concerned with propositional rules. Other types as relational rules, where the body is formed in first-order logic, are not considered here. Furthermore, the conditions of a rule are combined with a logical **AND** ($\wedge$) in the remainder of the thesis. A sample rule r_{nom} for the dataset displayed in Table 2.1 is given below.

$$r_{nom}\text{: outlook} = \text{sunny} \wedge \text{temperature} < 81 \rightarrow \text{play} = \text{yes}$$

The rule r_{nom} covers all examples whose "outlook" is "sunny" and whose "temperature" is below 81 and classifies them as "play = yes". In this case it covers the examples $\mathbf{x}_1$, $\mathbf{x}_2$, and $\mathbf{x}_6$. The other three examples are not covered by the rule meaning they are not part of the concept "play = yes".

Rules can also be used to describe regression data. Then, in the head of the rule a numerical value is predicted instead of a nominal value. The way a rule is build is the same as in classification but the search for a good rule is based on different metrics. An example for a regression rule for the dataset of Table 2.2 is given below.

$$r_{num}\text{: windy} = \text{FALSE} \rightarrow \text{temperature} = 74$$

The rule r_{num} covers all examples where it was not windy (examples $\mathbf{x}_1$, $\mathbf{x}_3$, $\mathbf{x}_4$, and $\mathbf{x}_5$). Then, it classifies their temperature to be 74. Contrarily to classification, where rule r_{nom} covered three examples correctly, rule r_{num} does not cover a single example correctly (because the temperature is never 74). While there may also be rules in classification that do not classify a single example correctly, this situation is much more typical for regression as the number of distinct values usually is much higher compared to classification. For example, the dataset with the biggest number of classes used in this thesis has 24 classes (cf. Table 4.1). The biggest number of distinct values for the regression datasets is 845 (cf. Table 6.1). Consequently, different metrics to measure the error of the rule are necessary as the probability that a regression rule exactly matches the correct value is much less than that a classification rule will predict the right class. In general, compared to classification, regression rules are harder to find and usually are not as accurate.

Typically, a dataset cannot be described sufficiently by using only a single rule. For example, if an additional example is added to the dataset displayed in Table 2.1 that has the form

<outlook=overcast, temperature=69, humidity=low, wind=TRUE, play=yes>

Algorithm 2.1 SEPARATEANDCONQUER*(Examples)*

```
Theory ← ∅
while POSITIVE(Examples) ≠ ∅ do
    # the conquer step: find a "good" rule
    Rule = FINDBESTRULEWITHTOPDOWNSEARCH(Examples)
    Covered = COVER (Examples)
    # the separate step: remove the covered examples
    Examples = Examples \ Covered
    Theory = Theory ∪ Rule
return (Theory)
```

a single rule is unable to explain all the positive examples (those where "play=yes") anymore. How the rules can be combined if a single rule is insufficient to cover all positive examples is described later (cf. Section 2.3.4).

2.2.2 A straight-forward Separate-and-Conquer Algorithm

Separate-and-conquer rule learning can be divided into two main steps:

1. learn a "good" rule from the given data
2. add the rule to the rule set and remove all examples covered by the rule; if positive examples are left **GOTO** 1

Algorithm 2.1 shows pseudo-code that implements these two steps.

First, a single rule is learned by the method FINDBESTRULEWITHTOPDOWNSEARCH (the *conquer* step). Then this rule is added to a set of rules (the *Theory*) and all examples covered by the rule are removed from the *Examples* (the *separate* step). The two steps are repeated until no more positive examples are left. In the simplest case this ensures that every positive example is covered at least by one rule (*completeness*) and no negative example is included (*consistency*). A rule that does not cover a negative example is called a *consistent rule*.

The procedure FINDBESTRULEWITHTOPDOWNSEARCH is shown in detail in Algorithm 2.2. It implements a straight-forward version of a *top-down* algorithm, where rules are searched in a general-to-specific way (each refinement makes the rule more specific). The procedure starts by refining the empty rule (the most general rule r_g). It is defined by

$$\text{most general rule } r_g : \text{TRUE} \rightarrow \text{class} = \text{class value}. \tag{2.3}$$

This rule covers all examples. A *specialization* of a rule is defined by adding a new condition to it while a *generalization* means that a condition is deleted from

Algorithm 2.2 FINDBESTRULEWITHTOPDOWNSEARCH*(Examples)*

```
BestRule = r_g
BestRefinement = BestRule
BestRuleEvaluation = EVALUATERULE (BestRule, Examples)
BestRefinementEvaluation = BestRuleEvaluation
# loop as long as refinements left
while Refinements ≠ ∅ do
    # add all possible conditions, if any, to the best refinement
    Refinements = REFINERULE (BestRefinement, Examples)
    for Refinement ∈ Refinements do
        # evaluate the current refinement
        Evaluation = EVALUATERULE (Refinement, Examples)
        # find the best refinement
        if Evaluation > BestRefinementEvaluation then
            BestRefinement = Refinement
            BestRefinementEvaluation = Evaluation
    # if best refinement is better than the current best rule, use it as best rule
    if BestRefinementEvaluation > BestRuleEvaluation then
        BestRule = BestRefinement
        BestRuleEvaluation = BestRefinementEvaluation
return (BestRule)
```

the rule. After a rule is specialized it covers less examples and has more conditions than before. We then call this rule more special as the rule was before. On the other hand, if a rule is generalized, a condition is removed from it so that it covers more examples but has less conditions than before. The most specific rule is defined by

$$\text{most specific rule } \; r_s : \text{FALSE} \rightarrow \text{class} = \text{class value}. \tag{2.4}$$

The rule r_s does not cover a single example. Note that r_g cannot be generalized any more and another specialization is impossible for r_s.

In the first step of the refinement procedure FINDBESTRULEWITHTOPDOWNSEARCH, all possible *refinements* of a given rule (*Refinements*) are built from the data. In this algorithm, a refinement of a *candidate rule* r_c is the addition of a condition g, i.e. a specialization. The refinement thus is defined as a conjunction ($r_c \wedge g$). Refining a candidate rule is also called *refinement step*. Note that, in general, a refinement can also be a generalization. Candidate rules are those that are potential candidates for a good rule. If no such refinement is possible any more the set returned by the REFINERULE method will be empty. This is usually the case when the rule tests all available attributes. In previous steps, REFINERULE will return a set of all possible refinements of a given candidate rule. The quality of all those refinements

Table 2.3: The confusion matrix

	predicted positive	predicted negative	
class is positive	p (*true positives*)	$P - p$ (*false negatives*)	P
class is negative	n (*false positives*)	$N - n$ (*true negatives*)	N
	rule covers the examples	rule does not cover the examples	$P + N$

is evaluated by the EvaluateRule method, i.e., it is decided which of them should be used to yield a "good" rule.

After the method RefineRule has computed all possible refinements, the best one is determined by the heuristic. This is done by evaluating all the refinements and storing the best one in *BestRefinement*. In the next step it is checked whether this best refinement is better than the current best rule and, if so, the new best rule becomes the refinement. These steps are repeated as long as it is possible to build new refinements. Note that in this straight-forward algorithm refinements are computed even if it is clear that the current best refinement cannot get better in further refinement steps. Importantly, the best rule that is returned in the end has not to be the last refinement, but the refinement that yielded the highest heuristic value during all refinement steps. A concrete example of the different refinement steps, namely the refinement process, is given in Chapter 3 (Figure 3.1).

After the best rule is found, it is added to the theory, the examples covered by it are removed and a new best rule is searched given that not all positive examples are already covered. Note that the rules that are induced with such a separate-and-conquer algorithm may overlap. This is because the examples covered by the rule are removed. Therefore, subsequent rules may also cover examples that were removed in previous steps. For this reason, the rules can be only used in the context of all previous rules. For example, a rule that states that all animals with two legs can fly may be correct because previous rules excluded all animals with two legs that cannot fly.

One of the most important points of the whole algorithm is how "good" is defined in the conquer step of Algorithm 2.1. Clearly, the best rule that can be induced from the given dataset is unknown. Additionally, there is no oracle at hand that tells which condition will be the best one to be added to the current rule. Consequently, some means of deciding which condition will be the best choice is necessary.

2.2.3 Searching for good Rules

Usually, a *heuristic* is used to evaluate such candidate rules. Heuristics are denoted by $h_{identifier}$, where the identifier is usually an abbreviation of the name of the

heuristic. We adopt a notation that was previously used [60]. The objective of a heuristic is to rank the rules based on their coverage statistics (cf. Table 2.3). Among others, the most important objectives are: cover as many positive examples and exclude as many negative examples as possible. The more positive examples and the less negative examples are covered by the candidate rule, the higher the evaluation (the value returned by the heuristic) will be.

In a concept learning problem, statistics based on positive and negative examples can be gathered. These statistics are then used inside the algorithm mostly for evaluating the candidate rules that are build during the process of learning a rule set (cf. Algorithm 2.2). Table 2.3 shows a so-called confusion matrix. All positive examples that are covered by the rule are called p, all negatives that are covered are referred to as n. The total number of positive/negative examples is P and N. Hence, the rule r_{nom} presented in Section 2.2.2 has a coverage of $p = 3$ and $n = 0$ among the total three positive and three negative examples. Usually these coverage statistics are denoted by $[n, p]$ at the end of the rule.

A simple example for a heuristic is *MaxPosCoverage* $h_p = p$, where the coverage on the positive examples is computed without regarding the negative coverage. In this sense, only one of the two objectives is met. Analogously, the heuristic *MinNegCoverage* $h_n = -n$ only tries to minimize the negative coverage without using the positive coverage. These two heuristics are rather simple and clearly they do not reach the two objectives for a solid heuristic. A simple method to meet both requirements is to combine these two heuristics. The resulting heuristic *accuracy* is defined by $h_{acc} = p - n$, accordingly. What means are used to implement more complex heuristics is discussed in detail in Section 2.6. At this point, the three heuristics presented above are also revisited. Here, it is only important to recognize that a heuristic evaluates a candidate rule and based on this value a best rule can be determined.

2.2.4 Discussion of the straight-forward Algorithm

Clearly the presented algorithm has some limitations. For example, it may be unnecessary to refine the candidate rule anymore (e.g., when it covers no negative instances). To circumvent this, usually a so-called stopping criterion is used. The way how the best rule is searched also is fixed. In Algorithm 2.1 a top-down strategy is used, i.e., the process starts with an empty rule and iteratively adds conditions to it. The empty rule is defined as a rule with the body `TRUE`, thus covering all examples. This rule is the most general rule r_g because when conditions are removed from the rule it will never cover more examples (cf. Equation 2.3).

Note that the length l of this rule is zero (the conditions `TRUE` for r_g and `FALSE` for r_s given in Equation 2.4 do not count). The length l is defined by

$$l \equiv \text{number of conditions.}$$

By adding more and more conditions, the rule covers less and less examples until, eventually, only a single example is covered. By adding conditions the length of the rule also increases. Thus, a different strategy to reach a good rule would be to start with a very specific rule (usually with a rule that has length k, i.e., the number of attributes) and remove conditions from it. This strategy is called bottom-up search. The type of strategy used to search for a rule is called *search bias*.

Assuming that a heuristic is used that minimizes the covered negative examples (e.g., *MinNegCoverage*), a rule set induced by this straight-forward algorithm is consistent. In situations where we deal with noisy data[3] or where the found rules tend to be overly specific, it may be beneficial to allow certain degrees of inconsistency. A rule set that is adapted too strongly to the given example set is said to *overfit* the data. Usually, these specific rule sets often classify new examples erroneously. Hence, an important factor for a good rule learning algorithm is to avoid overfitting. Algorithmically, this can be assured by selecting an appropriate heuristic, by stopping to refine a rule, or by using an additional method that prevents the algorithm from adding a rule to the theory.

2.2.5 Characterizing Separate-and-Conquer Algorithms

There are many different separate-and-conquer algorithms but nearly all of them can be distinguished by the following three basic dimensions [55]. First, a definition of these dimensions is given, then each of them is discussed briefly.

Language Bias: The hypothesis language of the algorithm, i.e., the type of rules that are used.

Search Bias: The search method that is used to guide through the search space.

Overfitting Avoidance Bias: The mechanism used to avoid overfitting. This could either be a mechanism to remove conditions from a learned rule, i.e., making it more general or whole rules are not added to the theory if they do not fulfill predefined requirements.

In each implementation of a rule learning algorithm that is based on the separate-and-conquer strategy these three basic dimensions are defined in a different way. For example, such a rule learning algorithm can use conditions that check nominal attributes for equality or inequality (the language bias of the algorithm). Other algorithms are employing a dynamic language that can be adapted in each step of the algorithm. Typically the languages are ordered by increasing expressive power. When a sufficient theory cannot be learned in the current language, it is switched to the next (more expressive) one (cf., e.g., CLINT [27, 26]).

[3] Noisy data commonly are data that have errors in the measurements of the attribute values. For example, the value of an attribute may be incorrect or the class value may be altered.

The search algorithm sketched in Algorithm 2.2 is a top-down search where conditions are iteratively added to an initially empty rule (the FINDBESTRULEWITHTOPDOWNSEARCH procedure). One could also reverse the procedure by removing conditions from a maximal specific rule. This results in a bottom-up search. A different choice is to combine these two search mechanisms resulting in a bidirectional search. Consequently, a refinement can be the addition or deletion of a condition then. There are many other alternatives including a beam search where many candidate rules are refined simultaneously or an exhaustive search where all possible candidates are generated. The search bias defines what actual method is used to find a promising rule.

Inauspiciously, rule sets are often strongly adapted to the given training data, which hinders their ability to classify new unseen examples. Without question, a good rule learning algorithm has to find a general rule set that is valid for the domain rather than a specific training dataset. The training data can be seen as an excerpt from the domain. In this sense, it is never complete. For example, the weather dataset (cf. Table 2.1) only includes observations of six different days. Clearly, future weather observations are not present in the dataset yet. In machine learning in general and in rule learning as a special case, overfitting is an important problem. Whenever a rule set or some other model overfits the data, it is suboptimal.

There are different ways to implement a bias to avoid overfitting. Often, rules are only accepted as candidates when they are general enough. This can either be assured directly by the heuristic or by an external method that checks every refinement whether it meets some predefined generality constraints or not. On the other hand, a general rule set can also be reached after the theory is learned by trying to reduce it then. Hence, either conditions are removed from the complete rules or whole rules are deleted from the rule set. Because overfitting avoidance is a severe factor when building rule sets it is described in more detail in Section 2.4.

2.3 Handling Multi-Class Problems

So far, we were concerned with concept learning problems, namely with datasets that have only two classes. Real-world problems often have more than two classes. The separate-and-conquer algorithm that was used in this thesis, as most other rule learners, is restricted to concept learning problems. Hence, a mechanism is necessary to adapt the algorithm to be able to deal with more than two classes. Essentially, this involves converting a *multi-class* problem to a binary (or two-class) one. For this reason, schemes that do so are called *binarization techniques*. The most common ones are described below.

As before, let u be the number of classes and y_i the i-th class. When a dataset has more than two classes ($u > 2$), a method is needed to decompose this *multi-class* problem into u binary ones. The most frequent method for such a conversion is a

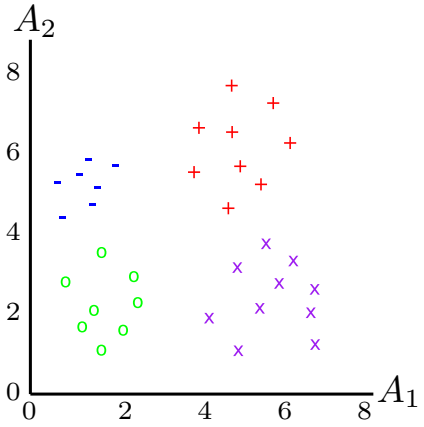


Figure 2.2: A 4-class classification problem

1-vs-all class binarization. It can be ordered or unordered. Other methods include the *pairwise approach* [56] which is explained later.

2.3.1 (Unordered) 1-vs-all Class Binarization

The goal of a 1-vs-all class binarization is to decompose the multi-class problem into u binary problems. The classes are not ordered in any way, hence this scheme is also called unordered class binarization. To perform such a binarization, examples of the first class (y_1) are defined to be the positive examples (i.e., those that belong to the concept). The examples of the remaining classes ($y_2, ..., y_u$) are defined as negative examples thus do not belong to the concept. Then a rule set for class y_1 is learned. After its induction a new binary dataset is built by defining the second class (y_2) as the positive one and all others, including the first class, as negatives ($y_1, y_3, ..., y_u$). This procedure is repeated until each class once was the positive one. Figure 2.2 displays a classification problem with four classes. The class that is decoded by blue minus signs has seven examples. The one that has green circles has eight examples, the class with the red plus signs has nine examples, and the class decoded by purple crosses has ten instances. The dataset has two numerical attributes A_1 and A_2, both residing in $[0, 8]$.

In an unordered class binarization, given that the first class to learn is the one that is depicted with the minus signs (the smallest class), this class is separated from all other classes. This situation is displayed in Figure 2.3 (a). Here, the examples with the minus signs are the positive examples and all others are the negative ones. Assuming that the next class to learn is the one with circles, Figure 2.3 (b) shows the situation for the next step when this class is now positive and again all other classes are negative.

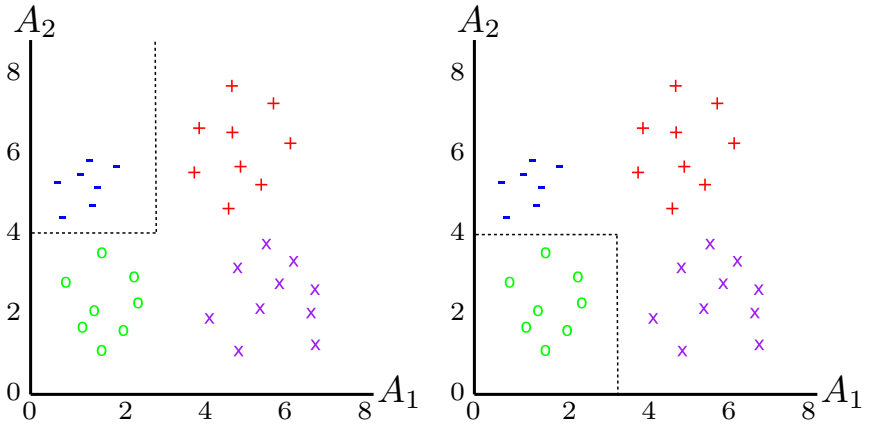


Figure 2.3: 1-vs-all class binarization

In the end of a 1-vs-all class binarization, one rule set is yielded that contains rules for all classes. The rule set is unordered, hence it is unclear how to handle ties, i.e., if more than one rule covers a test example but predict different classes. How algorithms cope with such situations is described in Section 2.3.4. Note that usually the number of negative examples is much larger than the number of positives because the negative examples are formed by the union of all examples of all the remaining classes.

An example rule for the second step of the unordered class binarization (Figure 2.3 (b)) would be:

$$r_{u_2}\colon A_1 < 3 \wedge A_2 < 4 \rightarrow \text{class=circles}$$

2.3.2 Ordered 1-vs-all Class Binarization

The basic decomposition in this scheme is the same as before but in a previous step the classes are ordered by their frequency. Then the smallest class is defined to be the positive one and all other classes form the set of negative examples just as before. The key difference, however, is that once a ruleset for one class is learned, all examples of this class are removed from the original dataset. In consequence, no rules for the biggest class are learned because it can simply be covered by a default rule that is located at the end of the list and which predicts exactly this class. In comparison to the unordered decomposition, the negative examples, given that y_2 is defined as positive, do not contain the examples from class y_1 any more (the negative examples are those from classes $y_3, ..., y_u$). Due to the imposed order, this method is referred to as *ordered class binarization*.

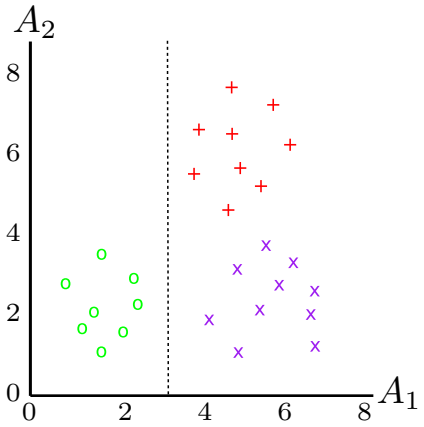


Figure 2.4: The second step of an ordered class binarization (step one is displayed in Figure 2.3 (a))

Figure 2.3 (a) displays the first step of an ordered class binarization which is the same as for a 1-vs-all class binarization. Figure 2.4 shows the second step, which is different because all examples from the first class are removed already. For this reason, the separation of the classes is also different compared to the second step of the 1-vs-all class binarization. Consequently, a different rule is learned in the ordered class binarization. The found rule r_{o_2} is:

$$r_{o_2}\colon A_1 < 3 \rightarrow \text{class=circles}$$

As can be seen, the rule was learned on a dataset where the examples covered by rule r_{o_1} were removed already. Thus, all examples that have the class "minus sign" are removed from the original dataset. When we compare rule r_{u_2} and r_{o_2} it is obvious that the first one has to check two conditions where the second one only needs a single condition, because the classes now can be separated without paying attention to the examples that have class "minus sign" which was necessary in an unordered class binarization (cf. Figure 2.3 (b)).

The ordered class binarization results in a rule set where rules predicting the same class appear blockwise. Note that once all rules predicting class y_1 have been included in the list the following rules never predict y_1 again. Additionally, rules now carry the context of all previous rules because the examples that were covered by these were removed. Usually, to classify a new example, the rules are used as decision list (cf. Section 2.3.4).

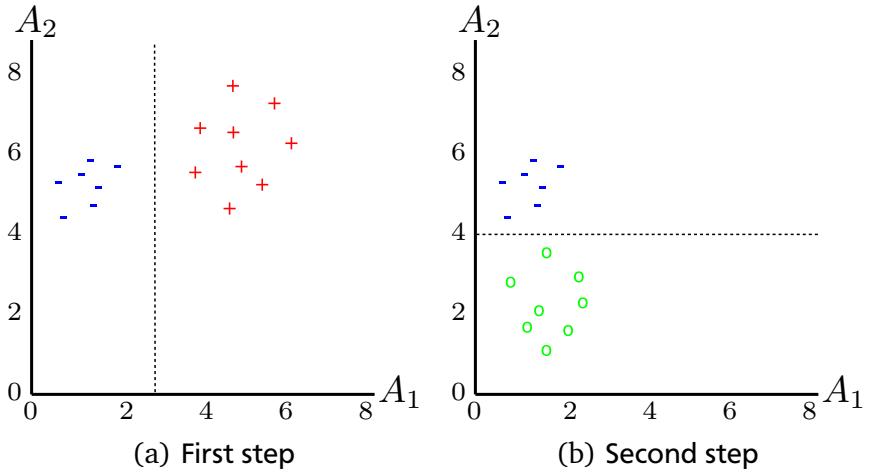


Figure 2.5: Two steps of a pairwise class binarization

2.3.3 Pairwise Class Binarization

The *pairwise class binarization* is often also called *1-vs-1*, *all-vs-all*, or *round robin* [56]. The key idea is to learn a total of $u \cdot (u-1)/2$ classifiers $c_{i,j}$, $i < j$, one for each **pair** of classes. So, in turn a classifier is learned to separate y_1 and y_2 ($c_{1,2}$), y_1 and y_3 ($c_{1,3}$), ..., and y_1 and y_u ($c_{1,u}$). This is done for all u classes. Figure 2.5 (a) displays the situation for learning the class decoded with minus signs against the class with plus signs. Figure 2.5 (b) shows the situation when the second classifier ($c_{1,3}$) is learned by separating the minus signs against the circles.

The motivation behind such an approach is that it may be easier to separate two classes than to separate one class against all others. The situation becomes obvious when the rules for the first step of an unordered or ordered class binarization are compared to the rule for the pairwise approach. Where the rules learned in step one of the first two binarization types have two conditions, the rule for the pairwise approach has only one condition[4]. At first sight, it seems to be rather inefficient to decompose a multi-class problem in this way. However, there is some research on how to implement the pairwise approach in an efficient way [127]. It was shown that this approach can have better performance than the (ordered) 1-vs-all class binarization [56]. Nonetheless, it is not considered in this thesis. The reason is that the number of rules for such an approach is much higher. The interpretability is also lost in a way because the rules are only separating exactly two classes. Rules learned with an ordered class binarization may be favored over the other sketched

[4] The rule would be $A_1 < 2.4 \rightarrow$ class=minus sign

approaches due to reasons of compactness albeit they may not be as accurate as those of the other binarization methods.

There are some other pairwise methods. Best-known are *ternary error-correcting output codes* [36, 3]. Even here, efficient algorithms are already derived [128]. In this thesis, we also do not consider these techniques, but to give a broad overview of binarization methods, they are nevertheless mentioned for reasons of completeness.

2.3.4 Ordered and Unordered Lists of Rules

The method used for binarization is related to the type of rule set. Depending on what type of rule set is used for classification, an appropriate binarization method has to be chosen. Using an inappropriate binarization technique may result in a rule set that fails to cover all classes which in turn often leads to a bad performance.

Rule sets can be ordered or unordered. The first type is a decision list [142] where rules are typically learned using an ordered class binarization as explained in the previous section. Whenever the rule set is unordered, more than one rule may cover an example. Hence, some means are necessary to handle these conflicts.

A mechanism to classify examples is to use such an unordered list of rules. Here, all rules that cover the example are used to obtain a prediction. Conflicts are solved by some kind of voting mechanism (i.e., simple voting or weighted voting). Rules are learned by a 1-vs-all class binarization as shown in Figure 2.3 (a) and (b). In the case of simple voting, each vote is weighted equally (i.e., a weight w_i is set to one for each rule r_i). Nevertheless, the heuristic value of the rules can also be interpreted as a confidence score for the prediction. In this case, information would be lost when a simple majority voting approach is utilized. Usually, the heuristic value of the rules has to be re-computed using the total number of training examples. The scores assigned during the process of learning are reflecting only the counts on parts of the training data because examples are beeing removed already. After the scores are re-computed, the heuristic value is used as weight or confidence score of each rule and the votes are weighted by the new w_i. Hence, the prediction may be altered when few rules that predict the same class have high weights assigned and there are many rules that vote for another class, but all of them have rather low weights. Note that an advantage of unordered rule sets is that usually more than one rule is used to classify an example which makes the prediction more confident. The main disadvantage is that all rules in the rule set have to be examined whether they cover the example or not.

A different approach is to use a decision list. Then, to classify an unseen examples, each rule in the list is tested from top to bottom whether it covers the example or not. But, in contrast to an unordered rule set, whenever a rule “fires”, i.e., covers the example, it is directly used to classify the example. Then, all other rules in the list are skipped. If no rule in the list covers the example a special rule is used which

$$
\begin{aligned}
r_1&: \quad A_1 = x_{1,1} \wedge A_2 < v_1 \rightarrow \text{class} = y_1 \\
r_2&: \quad A_1 = x_{1,2} \rightarrow \text{class} = y_2 \\
r_3&: \quad A_2 \geq v_2 \rightarrow \text{class} = y_2 \\
r_d&: \quad \text{TRUE} \rightarrow \text{class} = y_3
\end{aligned}
$$

Figure 2.6: A simple decision list

usually predicts the majority class in the dataset. Because no rule is learned for the majority class, there is an empty rule covering all remaining examples. Commonly, this rule is called *default rule*[5]. An example of a simple decision list is given in Figure 2.6. Here, the dataset contains two attributes where A_1 is a nominal attribute and A_2 is a numerical one. The dataset has three classes called y_1, y_2, and y_3. Note that the nominal attribute A_1 can have the categorical values $x_{1,1}$ and $x_{1,2}$ (analogously to equation 2.1) whereas the numerical attribute is compared against the thresholds v_1 and v_2.

In the remainder of the thesis, whenever decision lists are used, we refer to ordered rule sets that are learned by an ordered class binarization. One of the main advantages of such decision lists is that the classification phase usually is more efficient compared to an unordered rule set where all rules have to be examined. In contrast, a single rule may have diminished confidence in the prediction.

As an alternative type of decision lists some algorithms also employ a version where the class of the current rule is not fixed in advance. Thus, the decision list is not learned by an ordered class binarization. Albeit these decision lists are not used in the rule learners derived later, there are algorithms that build rule sets in such a different way. One example for such an algorithm would be CN2 [20]. The difference here is that the class is determined for each rule separately. Where the rules predicting a certain class appear blockwise in a regular decision list, there is no such restriction in this so-called *multiclass covering*.

To illustrate this situation consider the decision list given in Figure 2.6. Here, the first rule predicts class y_1 and in the next block each of the two rules predicts class y_2. In a decision list learned by multiclass covering the third rule (r_3) may also predict class y_1 again what can never happen when the rules are learned in a blockwise fashion for the different classes. While the ordered class binarization also learns decision lists, they are less flexible in that the order of the classes in the list is fixed and rules of different classes may not alternate. Nevertheless, there are some problems that come with multiclass covering. These are discussed in Section 3.4.1.

[5] The default rule is the most general rule r_g using the majority class as prediction (cf. Section 2.2.2).

r_1: outlook=sunny $\wedge$ $temp. < 76 \wedge temp. \geq 74$ $\wedge$ humidity=high $\wedge$ windy=FALSE $\rightarrow$ play=yes
r_2: outlook=sunny $\wedge$ $temp. < 81 \wedge temp. \geq 79$ $\wedge$ humidity=high $\wedge$ windy=TRUE $\rightarrow$ play=yes
r_3: outlook=overcast $\wedge$ $temp. < 84 \wedge temp. \geq 82$ $\wedge$ humidity=medium $\wedge$ windy=FALSE $\rightarrow$ play=no
r_4: outlook=rainy $\wedge$ $temp. < 71 \wedge temp. \geq 69$ $\wedge$ humidity=high $\wedge$ windy=FALSE $\rightarrow$ play=no
r_5: outlook=rainy $\wedge$ $temp. < 69 \wedge temp. \geq 67$ $\wedge$ humidity=medium $\wedge$ windy=FALSE $\rightarrow$ play=no
r_6: outlook=sunny $\wedge$ $temp. < 66 \wedge temp. \geq 64$ $\wedge$ humidity=low $\wedge$ windy=TRUE $\rightarrow$ play=yes
r_d: TRUE $\rightarrow$ class $= yes$

Figure 2.7: A strongly overfitted rule set

2.4 Overfitting Avoidance

In rule learning, *pruning* is a means to prevent overfitting, i.e., a situation where the model just reflects the examples and is unable to generalize to new examples. Note that overfitting may also be reduced by using appropriate heuristics. A maximally overfitted rule set, for instance, encodes each example as a single rule that covers exactly this example. Such a rule set for the dataset presented in Table 2.1 is given in Figure 2.7. Clearly, it will be suboptimal because each new unseen example is classified by the default rule (because it is never covered by any rule). On the other hand, a rule set that is too general, i.e., one that consists only of the default rule (in this case rule r_d) is also not a good choice. Essentially, each new example will also be classified by the default rule then. A good rule set has to trade-off between these two objectives. On the one hand, overfitting rules are usually more accurate (at least on the training set) because they cover only a few examples. On the other hand, it is important to cover many examples to guarantee a certain degree of generality. Thus, in an optimal rule set each rule covers many examples but still is accurate. How to reach a reasonable trade-off is the main research question posed in Chapter 4.

Many pruning algorithms exist for rule learning. They can be divided into *pre-pruning* and *post-pruning*. The first prunes a rule during learning whereas the latter is employed after the theory is learned. Pre-pruning is used in many popular algorithms, e.g., CN2 [20], FOIL [132], or FOSSIL [52]. Strong algorithms for post-pruning include *Reduced Error Pruning (*REP*)* [11] and GROW [21]. Moreover, both strategies can be either combined yielding *Top-Down Pruning* [53] or integrated resulting in *Incremental Reduced Error Pruning* (I-REP) [61]. In the following pre- and post-pruning are described. The integration of the two strategies is also outlined as it is used in the algorithm RIPPER. In the literature a more detailed description of all these pruning techniques can be found [54].

A simple method to do pre-pruning is to stop learning a rule before it is regularly stopped. Usually a rule is refined as long as it covers negative examples or as long as attribute-value tests are left. A test of significance for each refinement is a means to implement pre-pruning. If the distribution of the examples covered by the refined candidate rule is not significantly different from the distribution in the whole dataset the refinement process of this rule is stopped. This method is implemented, e.g., in CN2 and works by comparing the *likelihood ratio statistic* to a χ^2 distribution with one degree of freedom. Other mechanisms use *encoding length restrictions* of the candidate rules or require a minimum heuristic value of each candidate rule (called *cutoff stopping criterion*). Note that pre-pruning is not limited to stop the refinement of a single rule, but can also be used to stop the addition of rules to the rule set.

As mentioned above, a rule can also be pruned by relying solely on the heuristic value. Actually, the search procedure sketched in Algorithm 2.2 relies exactly on this. Here, not necessarily the last refinement is returned but the refinement that has reached the highest heuristic value. In this sense, it can happen that the selected candidate rule covers some negative examples.

The most famous method for post-pruning is *Reduced Error Pruning (*REP*)*. Essentially, the training set is divided into a *growing set* (usually $2/3$ of the whole training data) and a *pruning set* ($1/3$). A rule set is learned on the growing set without paying any attention to overfitting. Often, even overfitting rules are learned to give a good starting point for REP after the theory is learned. Then, for each rule, conditions are deleted one by one as long as the quality of the rule does not decrease on the pruning set. A major problem of this strategy is that it is inefficient. By inducing an overfitted rule set, many conditions have to be deleted during the process of pruning. Another effect is that by removing conditions from a rule, it will become more general, i.e., covering more examples including some that were not covered by the rule beforehand. Then, rules that were learned after the current rule can become obsolete because the examples covered by them are already covered by the pruned rule.

In summary, the original REP strategy has some disadvantages. To overcome them, *Incremental Reduced Error Pruning* (I-REP) [61] was suggested. Here, each candidate rule will be pruned right after it was added to the theory. The main advantage is that training examples that are covered by the pruned rule[6] can be removed so that they do not influence the learning of subsequent rules. The I-REP strategy proved to be successful [53] and hence is used in the most powerful rule learning algorithms as, e.g., in RIPPER [22]. It is also the only post-pruning technique that is relevant in this work.

[6] Due to its lower number of conditions a pruned rule covers more examples than a consistent rule.

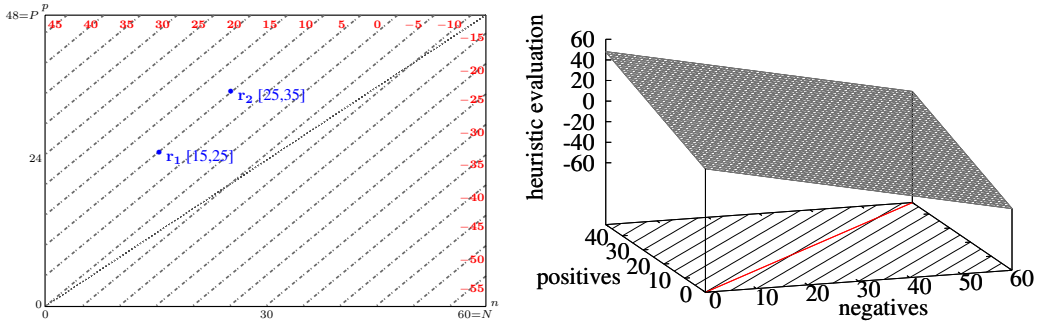


Figure 2.8: Isometrics in 2-d and 3-d coverage space

2.5 Visualization with Coverage Space Isometrics

In this section, we will briefly recapitulate coverage spaces, which will be our primary means of visualizing the behavior of the investigated heuristics.

Previously, it was suggested to visualize the behavior of rule learning heuristics by plotting their isometrics in *coverage space*, an un-normalized version of ROC-space [60]. Unlike ROC-spaces, the coverage space plots p (the absolute number of covered positive examples) on the y-axis and n (the absolute number of covered negatives) on the x-axis. For example, the point $(0,0)$ represents the empty theory where no example is covered at all. A good algorithm should navigate the learning process in the direction of the point $(0,P)$, which represents the optimal theory that covers all positive examples and no negatives. The point $(N,0)$ represents the opposite theory, and the universal theory, covering all P positive and N negative examples, is located at (N,P).

We can also represent individual rules r_i by a point (n_i,p_i) where $n_i \in N$ are the covered negative examples and $p_i \in P$ are the covered positives. *Isometrics* connect rules $r_1,...,r_q$ which have an identical heuristic value but cover different numbers of examples. The preference bias of different heuristics may then be visualized by plotting the respective heuristic values of the rules on top of their locations in coverage space, resulting in a 3-dimensional (3-d) plot $(n,p,h(p,n))$ [45] (right picture of Figure 2.8). A good way to view this graph in two dimensions is to plot the *isometrics* of the learning heuristics, i.e., to show contour lines that connect rules with identical heuristic evaluation values. Figure 2.8 shows examples of a 2-d and 3-d coverage space that both contain isometrics of *accuracy* ($h_{acc} = p-n$). The left one shows the respective values assigned by the heuristic as numbers attached to the contour lines whereas the right one shows them as a 3-d surface. The rules r_1 (covering 15 negatives and 25 positives) and r_2 ($n = 25, p = 35$) both have an accuracy of ten and therefore lie on the same isometric. For visualization, one is

primarily interested in the shape of the isometrics. Thus, we will typically omit the evaluation value from the graph and prefer the 2-d plots.

In the following all rule learning heuristics used in this thesis are defined and described. In doing so, the preference bias of each of the heuristics is also summarized. For some of them the isometrics in the coverage space are shown to illustrate these preferences.

2.6 Rule Learning Heuristics

Functions that are employed in situations where full knowledge of the field is inaccessible are called *heuristics*. They are used to compute an approximate solution. This section takes up the short discussion of heuristics given in Section 2.2.3. Depending on the field, heuristics are also called *evaluation metrics* or *objective functions*. In association rule mining, e.g., metrics to evaluate the quality of an association rule are called *interestingness measures* (cf. [159] for an overview). These terms are often used without distinction. We do not want to distinguish them here but want to emphasize that this section is focused on heuristics for classical propositional rule learning.

There are many different heuristics given in the literature (for an overview see [60]). Three examples were already presented above, namely *MaxPosCoverage*, *MinNegCoverage*, and their combination *accuracy*. One of the main differences between the heuristics is whether they evaluate the quality on an absolute scale or on a relative one. The first type of heuristics evaluates the performance independently of any previous rule whereas the second type of heuristics measures the quality of a rule related to its predecessor. To distinguish these two kinds, we define the first group as *value*-heuristics and the second one as *gain*-heuristics. While most of the algorithms use *value*-heuristics (e.g., CN2 [20] or AQ [113]), a few also employ *gain*-heuristics (as RIPPER [22] does).

The work reported in this thesis mainly concerns *value*-heuristics. Those are divided into three different categories depending on how they try to optimize the two criteria discussed below. Heuristics can also be separated into linear and non-linear ones [60]. In our case, however, it is more convenient to categorize them among their complexity in implementing consistency and coverage. Recall that, in principle, the goal of a rule learning algorithm is to find a simple set of rules that explains the training data and generalizes well to unseen data. This means that individual rules have to optimize two criteria simultaneously:

Coverage: The number of positive examples that are covered by the rule should be maximized.

Consistency: The number of negative examples that are covered by the rule should be minimized.

Thus, each rule can be characterized by the statistics given in the confusion matrix (cf. Table 2.3). In this sense, most rule learning heuristics depend on p, n, P, and N, but combine these values in different ways. Sometimes heuristics also rely on the length l of the rule or on the statistics of the previous rule as *gain*-heuristics do. For those, the positive and negative coverage of a rule's predecessor is defined by p' and n'. Most often, only the four values of the current rule are used as input for the heuristic. In general, all values a heuristic is computed from are called *coverage statistics* in the remainder of this thesis.

Later on, we will evaluate the utility of taking the rule's length into account (cf. Section 4.3.2). However, as our goal is to evaluate a rule irrespective of how it has been learned, we will not consider the parameters p' and n'. Heuristics like FOIL's information gain [130] (see Section 2.6.4), which include p' and n', may yield different evaluations for the same rule, depending on the order in which its conditions have been added to the rule's body. Moreover, as discussed above, rules with different predecessors are incomparable, and thus it is not possible to return the best rule encountered in a search which is a crucial step in Algorithm 2.2. We will not further consider heuristics of this type in this thesis.

As P and N are constant for a given dataset, heuristics that do not rely on the rule's predecessor differ effectively only in the way they trade off completeness (maximizing p) and consistency (minimizing n). Thus they may be viewed as functions $h(p,n)$. Thus, they only depend on the number of covered positive and negative examples and are unable to discriminate between rules that cover the same number of positive and negative examples. So it follows from the first observation that the equation given below holds for all rules r_i.

$$h\left(r_i\right) \equiv h\left(n_i, p_i\right)$$

Resulting from the second observation it is obvious that

$$r_1 \neq r_2 \nrightarrow h\left(r_1\right) \neq h\left(r_2\right)$$

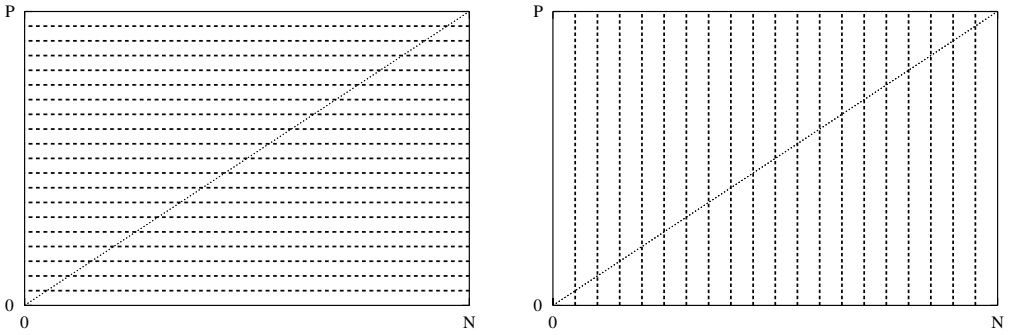
holds.

In the following, we will survey the heuristics that will be investigated. Most of these heuristics have already been discussed [60], so we will keep the discussion short.

Value-heuristics can be divided into *basic heuristics*, which primarily focus on one aspect (i.e., either consistency or coverage), *composite heuristics*, which provide a fixed trade-off between the two objectives, and *parametrized heuristics*, which provide a parameter that allows to tune this trade-off.

Table 2.4: Basic heuristics

name	formula
MaxPosCoverage	$h_p = p$
MinNegCoverage	$h_n = -n$
recall or true positive rate (tpr)	$h_{rec} = \frac{p}{P}$
false positive rate (fpr)	$h_{fpr} = \frac{n}{N}$
full coverage	$h_{cov} = \frac{p+n}{P+N}$



(a) *recall* (b) *MinNegCoverage*

Figure 2.9: Isometrics for *recall* and *MinNegCoverage*

2.6.1 Basic Heuristics

These heuristics are rather simple and do either optimize consistency or coverage on its own. Table 2.4 gives an overview of these heuristics. *MaxPosCoverage* and *MinNegCoverage* were already discussed in Section 2.2.3. The *true positive rate* which is also called *recall* computes the coverage on the positive examples only. It is – on its own – equivalent to simply using p (because P is constant). In other words it is equivalent to the heuristic *MaxPosCoverage*. Due to its independence of covered negative examples, its isometrics are parallel horizontal lines. Thus, *MaxPosCoverage* and *recall* have the same isometric structure which is displayed in Figure 2.9 (a). The *false positive rate* computes the coverage on the negative examples only. Its isometrics are parallel vertical lines (shown in Figure 2.9 (b)). It is equivalent to *MinNegCoverage*.

Full coverage is the fraction of all covered examples. The maximum heuristic value is reached by the universal theory, which covers all examples (the point

Table 2.5: Composite heuristics

name	formula
accuracy	$h_{acc} = p - n$
weighted relative accuracy (wra)	$h_{wra} = \frac{p}{P} - \frac{n}{N}$
precision	$h_{prec} = \frac{p}{p+n}$
Laplace	$h_{lap} = \frac{p+1}{p+n+2}$
correlation	$h_{corr} = \frac{p \cdot N - n \cdot P}{\sqrt{P \cdot N \cdot (p+n) \cdot (P-p+N-n)}}$
odds ratio	$h_{odds} = \frac{p \cdot (N-n)}{n \cdot (P-p)}$

(N, P) of the coverage space). The isometrics are parallel lines with a slope of -1 (similar to those of the lower right graph in Figure 2.13).

2.6.2 Composite Heuristics

The heuristics shown in the previous section only optimize one of the two criteria, consistency or coverage. In this section, we will discuss a few standard heuristics that provide a fixed trade-off. Table 2.5 shows an overview of them. Note that *accuracy* which is a direct combination of the two objectives that are implemented by the heuristics *MinNegCoverage* and *MaxPosCoverage* is also included. *Accuracy* was introduced in Section 2.2.3, but below the corresponding isometrics are discussed. The isometrics of *accuracy* are also used to illustrate how the coverage space works (cf. Figure 2.8 in the section before).

Precision

Another way to combine the two objectives is to look at the fraction of correctly classified examples among all the covered examples. It is implemented in the heuristic *precision*. Its isometrics are rotating around the origin as can be seen in Figure 2.10. *Precision* is known to learn overly complex rules, as will also become obvious from the results shown in the Tables 4.3 and 4.4. More precisely, for rules with high consistency, coverage becomes less and less important. All rules with maximum consistency ($h_{prec} = 1.0$) are considered to be equal, irrespective of their coverage. This can be seen well from the isometric structure (cf. Figure 2.10), where the slopes of the isometrics become steeper and steeper when they approach the P-axis (approximately the region left from the red vertical line),

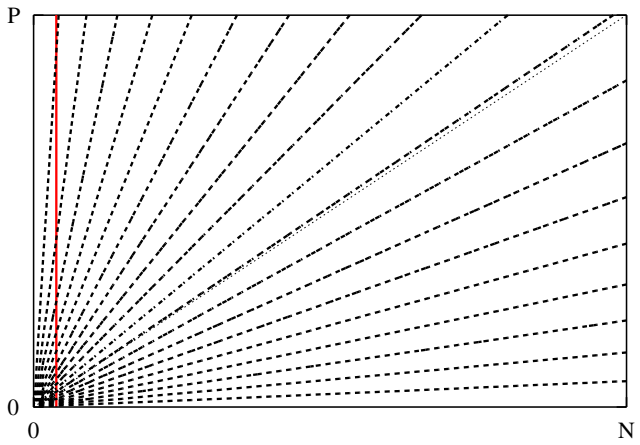


Figure 2.10: Isometrics for *precision*

which by itself forms the isometric for the maximum consistency case. The inverse behavior (preferring coverage over consistency for regions with high coverage) can also be observed near the N-axis, but this region is not interesting for practical rule learning systems.

Laplace

Laplace, in contrast, is an attempt to alleviate the overfitting behavior of *precision* by initializing the counts for p and n with one, thereby effectively moving the rotation point of *precision* to $(-1,-1)$ in the coverage space. It is used in the CN2-algorithm [20]. However, it is known that the *Laplace* heuristic will still lead to serious overfitting if used without appropriate pruning heuristics. Thus, it also places too strong emphasis on consistency over coverage.

Accuracy

Accuracy, essentially, computes the percentage $(p+(N-n))/(P+N)$ of correctly classified examples among all training examples. As P and N are typically constant for the evaluation of a set of candidate rules, this is equivalent to the simpler $p-n$. Its isometrics in coverage space are parallel lines with a slope of one (45 degrees) as depicted in Figure 2.8. Interestingly, the pruning criterion of I-REP [61] and also the heuristic of PROGOL [122] are implemented by using *accuracy*. We will see later in this thesis that this measure over-generalizes, i.e., it places too strong emphasis on coverage.

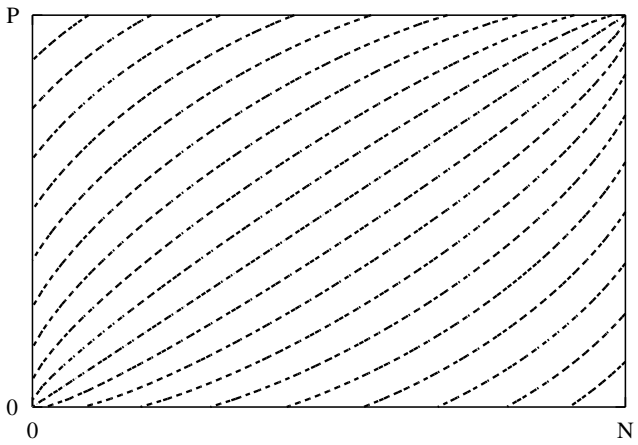


Figure 2.11: Isometrics for *correlation*

Weighted Relative Accuracy

Weighted relative accuracy (*wra*) [104, 129] is an attempt to adapt to the distribution of positive and negative examples in the dataset. As a result, the isometrics are now parallel to the diagonal of the coverage space instead of those of *accuracy* which have a slope of one (cf. the upper right graph of Figure 2.13). The measure has been successfully used in subgroup discovery [105]. However, for inductive rule learning, the experimental evidence given in [162], which is consistent with our own experience presented later in this thesis, suggests that this measure has a tendency to over-generalize.

Correlation

Correlation computes the correlation coefficient between the predicted and the target labels. Like *weighted relative accuracy*, its isometrics are symmetrical around the diagonal, but their ends are bended towards the $(0,0)$ and (N,P) points (cf. Figure 2.11). The measure has exhibited a good performance in the inductive rule learning algorithm FOSSIL [52] (where it was formulated as a FOIL-type *gain*-heuristic, i.e., p' and n' were used instead of P and N), and has been frequently used in association rule and subgroup discovery [9, 182].

Odds Ratio

The *odds ratio* essentially computes the strength of association between two variables. In the case of using it with a confusion matrix and an equal distribution of

Table 2.6: Parametrized heuristics

name	formula
cost measure	$h_c = c \cdot p - (1-c) \cdot n$
relative cost measure (rcm)	$h_{c_r} = c_r \cdot h_{rec} - (1-c_r) \cdot \frac{n}{N}$
F-Measure	$h_F = \frac{(\beta^2+1) \cdot h_{prec} \cdot h_{rec}}{\beta^2 \cdot h_{prec} + h_{rec}}$
m-estimate	$h_m = \frac{p + m \cdot \frac{P}{P+N}}{p+n+m}$
Klösgen measure	$h_\omega = (h_{cov})^\omega \cdot \left(h_{prec} - \frac{P}{P+N}\right)$

P and N, a value of one means that the positive and negative examples covered are distributed equally, a value below one means that the correlation is negative, i.e., more negatives than positives are covered, and values above one meaning that $p > n$. A general overview and some equivalences of many different measures (including the *odds ratio*) is given in the literature [160].

2.6.3 Parametrized Heuristics

Although the measures discussed in the previous section aim at trading off consistency and coverage, they implement a fixed trade-off. As experience shows, this is suboptimal, e.g., it often unduly prefers consistency or coverage. In this section, we will discuss five heuristics that allow to tune this trade-off with a parameter. All of them are illustrated in Table 2.6. We will start with two cost measures, which directly trade off absolute or relative positive and negative coverage. Thereafter, we will see three measures that use *precision* for optimizing consistency, but use different measures (*recall*, *weighted relative accuracy*, and *full coverage*) for optimizing coverage [88].

The parametrized heuristics can also be visualized in coverage space. But to do so, a parameter setting has to be given. Depending on this parameter, the isometric structure of the heuristic changes. To give an impression how the isometrics evolve over different parameter settings, isometrics for the *Klösgen measure* are given in Figure 2.13. For a visualization of the other heuristics, we refer to Chapter 4 where the parameter settings were optimized. For those settings, isometrics are given in Figure 4.3.

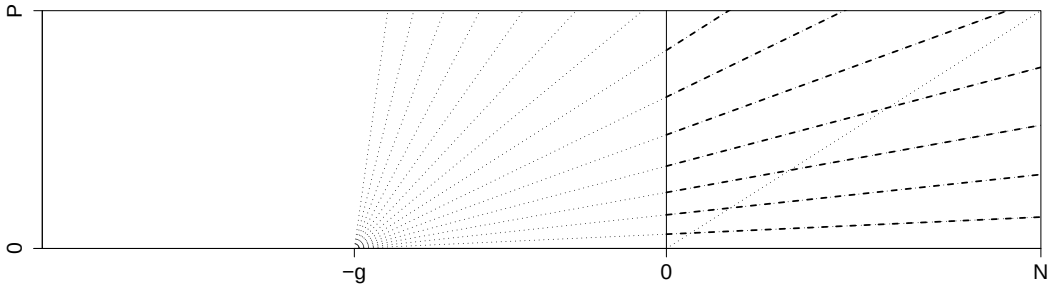


Figure 2.12: General behavior of the F-Measure

Cost Measures

The *cost measure* allows to directly trade off consistency and coverage with a parameter c. Setting $c = 0$ only considers consistency, $c = 1$ only coverage. If $c = 1/2$, the resulting heuristic is equivalent to *accuracy*. The isometrics of this heuristics are parallel lines, with a slope of $(1-c)/c$. This heuristic was derived by Fürnkranz and Flach [60].

An extension of the *cost measure* quite similar to the modification of *accuracy* that yielded *weighted relative accuracy* is the *relative cost measure (rcm)*. It trades off the *true positive rate* and the *false positive rate*. This heuristic is quite similar to the *cost measure*. In fact, for any particular dataset, the *cost measure* and the *relative cost measure* are equivalent if $c_r = P/(P+N) \cdot c$. However, the performance of fixed values of c and c_r over a wide variety of datasets with different class distributions will differ. Clearly, setting $c_r = 1/2$ implements *weighted relative accuracy*.

F-Measure

The F-measure [146] has its origin in Information Retrieval and trades off the basic heuristics *precision* and *recall*. Its isometrics are illustrated in Figure 2.12. The figure is taken from [60]. Basically, the isometrics are identical to those of precision, with the exception that the rotation point does not originate in $(0,0)$ but in a point $(-g,0)$, where g depends on the choice of β. If $\beta \to 0$, the origin moves towards $(0,0)$, and the isometrics correspond to those of *precision*. The more the parameter is increased the more the origin of the isometrics is shifted in the direction of the negative N-axis. The observable effect is that the lines in the isometrics become flatter and flatter. Conversely, if $\beta \to \infty$, the resulting isometrics approach those of *recall* which are horizontal parallel lines.

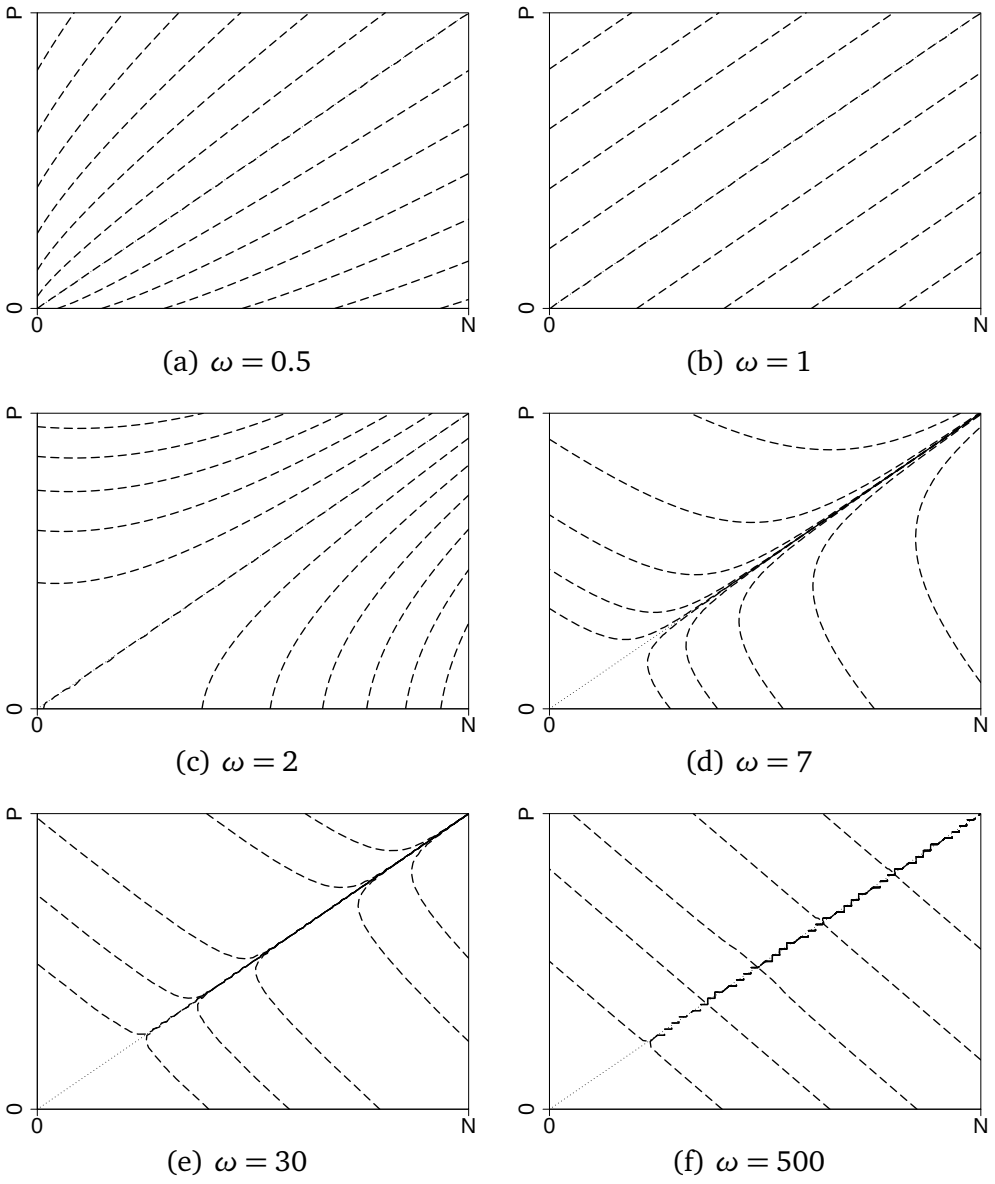


Figure 2.13: Klösgen-Measure for different settings of ω

m-estimate

The idea of the m-estimate [15] is to presume that a rule covers m training examples *a priori*, maintaining the distribution of the examples in the training set ($m \cdot {}^{P}/_{(P+N)}$ examples are positive). For $m = 2$ and assuming an equal example distribution ($P = N$), we get *Laplace* as a special case.

If we inspect the isometrics in relation to the different parameter settings, we observe a similar behavior as discussed above for the F-measure, except that now the origin of the turning point is fixed on the N-axis, but it is shifted in the direction of the negative diagonal of the coverage space (cf. [60], for an illustration). $m = 0$ corresponds to *precision*, and for $m \to \infty$ the isometrics become increasingly parallel to the diagonal of the coverage space, i.e., they approach the isometrics of *weighted relative accuracy*. Thus, the m-estimate trades off *precision* and *weighted relative accuracy*.

Klösgen Measure

The *Klösgen measure* trades off *precision gain* (the increase in *precision* compared to the default distribution $P/(P+N)$) and *full coverage*. The isometrics of *precision gain* on their own behave as the isometrics of *precision*, except that their labels differ (the diagonal now always corresponds to a value of zero).

Setting $\omega = 1$ results in *weighted relative accuracy*, and $\omega = 0$ yields *precision gain*. Thus, the Klösgen measure starts with the isometrics of *precision* and first evolves into those of *weighted relative accuracy*, just as the m-estimate. However, the transformation takes a different route, with non-linear isometrics. The first two graphs of Figure 2.13 shows the result for the parameter settings $\omega = 0.5$ and $\omega = 1$ (*weighted relative accuracy*), which were suggested by Klösgen. All graphs of Figure 2.13 are taken from [88] and have previously appeared [84].

With a further increase of the parameter, the isometrics converge to *full coverage*. The middle left graph shows the parameter setting $\omega = 2$, which was suggested in [180]. Contrarily to the previous settings, the isometrics now avoid regions of low coverage, because the influence of the (negative) coverage is increased. A further increase of the parameter results in sharper bends of the isometrics. The influence of *weighted relative accuracy* (the part parallel to the diagonal) vanishes except for very narrow regions around the diagonal, and the isometrics gradually transform into those of coverage.

Another interesting variation of the Klösgen measure is to divide h_{cov} by $1 - h_{cov}$ instead of raising it to the ω-th power. It has been shown before that this is equivalent to *correlation*. This family of measures was first proposed in [96], and has been frequently used for subgroup discovery. For a more detailed discussion of the *Klösgen measure* see [58].

2.6.4 Gain-Heuristics

Recall that the main difference between *gain*- and *value*-heuristics is that the latter evaluates a candidate rule on an absolute scale, relying only on the coverage statistics. A selection of those heuristics is described above. However, *gain*-heuristics

proceed by comparing the quality of the current candidate rule with its predecessor. One of the first algorithms that uses a *gain*-heuristic was FOIL [132]. The same heuristic is still used in the RIPPER algorithm.

$$\textit{foil gain: } h_{fgain} = p' \cdot \left(log_2(\frac{p'}{p'+n'}) - log_2(\frac{p}{p+n}) \right)$$

Foil gain computes the weighted information gain. Here, it is counted how much the refinement with the current condition yields in comparison to the original candidate rule. The measure is motivated by an information theoretic background. The information of the current candidate rule ($-log_2\ p/(p+n)$) is compared to the information of the refined candidate rule where the new condition is added ($-log_2\ p'/(p'+n')$) and weighted by the coverage on positive examples of the refined candidate rule (p') [132].

2.7 Evaluation Methods for Rule Learning Algorithms

A crucial issue in machine learning is the evaluation of algorithms [91]. Rule learning is no exception. However, the evaluation methods described below are not only valid for rule learning, but are universally usable to evaluate machine learning algorithms. A naive approach is to simply compute the correctly classified instances among all instances on the training set. In that case, an intuition is given of how good the algorithm has adapted the training data. Problematically, overfitting is not taken into account then. Usually, an algorithm that has a good performance on the training dataset sustains losses in accuracy on unseen data because the algorithm has not generalized from the training data.

For this reason, a solid performance measure has to estimate the accuracy that the algorithm will have on unseen data. This is also called *generalization* performance. One method to do so is the so-called *cross-validation*. In the following this method is explained. Then, measures that are independent from the learning task are illustrated followed by a discussion of measures that are used to evaluate regression and classification algorithms. In the end, it is shown how rankings are evaluated and statistical tests are described.

2.7.1 Cross-Validation

The goal of a cross-validation is to give an estimate how good an algorithm will perform on new data that were not present during the learning phase. This "new data" should be seen as data that is completely independent but is drawn from the same population as the training data. The idea is to aggregate several measurements into a single performance estimate. Therefore, the dataset is split into f folds where

each instance resides exactly in one fold. Usually, the class distribution in each fold is the same as in the complete dataset. In this case, the cross-validation is *stratified*. There are many different suggestions how many folds should be used, but often ten folds are chosen. While there is no clear evidence that this is the best choice, many sources have empirically evaluated that a so-called 10–fold cross-validation yields estimates that reflect the generalization performance sufficiently [177, 99]. Nevertheless, some authors proposed different cross-validations (see, e.g., [147, 35]), mostly for correcting a paired t-test, but still 10–fold cross-validation is most widely used. One may argue that using a so-called *leave-one-out cross validation* where the number of folds equals the number of examples (in the following called t) is a better choice than using ten folds. The main problem here is that this method is very inefficient as t models have to be learned instead of only ten. Also, using bootstrap methods where the training set is partitioned into a certain percentage of training and testing data is a bad choice as this method uses a single random split instead of ten. However, a detailed overview on cross-validation and related work can be found in the literature [137].

In such a 10–fold cross-validation the dataset is split into ten folds of approximately similar size $(f_1, f_2, ..., f_{10})$. Then the first rule set is learned by using the first $f-1$ folds $f_1, f_2, \ldots, f_9$ for training and the last fold f_{10} for testing. The second rule set is learned by using the folds $f_1, f_2, \ldots, f_8, f_{10}$ for training and fold f_9 for testing and so on until all of the ten folds have served as test fold once. In the end, we have ten measurements, each associated with one test fold. Then, usually the mean of these measurements is used as final performance estimate.

Note that one approach to reduce the variance is to perform a 10–fold cross-validation q times (called $q \times 10$–fold cross-validation). This extends the runtime of the 1×10–fold cross-validation by a factor of q. Also, the procedure ends up with $q \times 10$ single measurements. These are also aggregated by using the mean to yield a single performance estimate. For performance reasons, we used simple 1×10–fold cross-validation for all experiments. Note that whenever $q = 1$ and $f = 10$, we abbreviate with 10–fold cross-validation.

2.7.2 Theory size

In the following, we describe metrics that are independent from the learning task. There are some uniform measures that are valid both in classification and regression as they do not estimate accuracy or error directly. The most important one is the *size* of the theories. To measure the size, one can use the **average number of rules** of the rule sets R_i. As before d denotes the number of datasets.

$$average\ \#\ rules = \frac{1}{d}\sum_{i=1}^{d}|R_i|$$

The **average number of conditions** is the average number of conditions of the rule sets R_i. Note that the average number of conditions or the term conditions itself is often abbreviated with *conds*.

$$average\ \#\ conditions = \frac{1}{d}\sum_{i=1}^{d}|conds(R_i)|$$

where the function $conds(R_i)$ yields the number of conditions of all rules of the i-th ruleset R.

Note that the theory size implicitly also measures comprehensibility albeit a clear correlation is hard to infer. Often, there is a trade-off between comprehensibility and the accuracy of a rule set. When the rule learner is focused too strongly on one of these concurrent goals, the rule set will be suboptimal. Either it is accurate but has too many rules with too many conditions so that a human is unable to interpret the rule set any more or the theory contains only a few rules but its classification performance is insufficient. How many rules can be interpreted or comprehended by a human still is an open question which yet cannot be answered in all of its aspects. However, for practical applications, the theory size can be equally important as accuracy. Size is also related to the coverage of single rules as a high coverage often means that the rule is rather simple whereas a low coverage indicates that we have an overly complex rule (cf. Section 2.2.2). This interpretation of the size has, among others, led to the experiments presented in Chapter 4.

There are other ways to measure comprehensibility. Especially when different rule sets should be compared based on their comprehensibility a structure called *exception directed acyclic graph (EDAG)* has been proposed [65, 64]. A rule set can be converted into such an *EDAG* which then can be used for comparing the size and the comprehensibility of the rule set. As the authors also note the main means for measuring comprehensibility is to count objects such as arcs, premises, and conclusions in an *EDAG* or simply conditions in a rule [62]. To simplify the comparison, we used the number of conditions to measure the size and the comprehensibility, respectively. On the one hand this is much more efficient than the three conversion steps of the INDUCT algorithm [63] necessary to obtain an *EDAG* and on the other hand we believe that a comparison based on the presented measure is sufficient for our purposes that mainly are focused on an intuitive idea of how big the rule sets are. As mentioned above a solid comparison or estimate of comprehensibility is impossible anyhow which confirms our choice.

2.7.3 Evaluation of Regression Rule Learning Algorithms

In the meta-learning experiments in Section 4.3 and for the evaluation of the regression rule learners (Chapter 6 and 7), evaluation measures for numerical target values are needed. In the following some basic metrics are described. Note that

these measures are computed on a single dataset. To aggregate several measurements computed on a set of datasets, the individual values are averaged. In the following, y'_j denotes the predicted value and y_j the true value for the j-th instance out of t instances in total. $\hat{y}$ stands for the mean over all t instances in a dataset with numerical target values. To obtain true values for instances a cross-validation is used as described before.

Mean absolute error (*mae*) is the deviation of the predicted value y' from the true target value y, averaged over all t instances (the union of all instances in the test folds of the cross-validation).

$$mae = \frac{1}{t}\sum_{j=0}^{t}|y - y'_j|$$

Root mean squared error (*rmse*) essentially is the standard deviation (σ) if we set $y'_j = \hat{y}$, for j=1,...,n, where $\hat{y}$ is the mean over all examples.

$$rmse = \sqrt{\frac{1}{t}\sum_{j=1}^{t}(y_j - y'_j)^2}$$

A problem of the two abovementioned evaluation metrics is that they are domain-dependent. As the magnitude of the target variable changes for different datasets, the magnitude of the error measurements changes as well. For example, a *rmse* of 1.0 may be near optimal for a dataset where the target values are in $[-10000, 10000]$. Contrarily, if the value range of the dataset is $[0, 1]$ an error of 1.0 is clearly suboptimal. Hence, the evaluations of different datasets are incomparable. One way to deal with this problem is to normalize the values. Therefore, usually the deviation from the mean is used:

$$dev_{mean} = \sum_{j=1}^{t}(y_j - \hat{y})^2$$

The deviation from the mean measures the variance when the mean over all instances is the predicted value. It is used to normalize the *rmse* to get the **relative root mean squared error** (*rrmse*). Note that the *rrmse* usually resides in [0,1]. Nevertheless, dependent on the dev_{mean} the *rrmse* can have values bigger than one. In these cases using the mean over all examples as prediction is a better choice than the model induced by the learning algorithm. For an analysis and situations where this actually occurs see, e.g., Section 6.5.2. The *rrmse* is defined as follows:

$$rrmse = \frac{rmse}{\frac{1}{t} \cdot dev_{mean}}$$

To measure how many examples are covered by a rule, simply the **relative coverage** is computed:

$$relcov = \frac{\text{examples covered by the rule}}{\text{all examples}}$$

2.7.4 Evaluation of Classification Rule Learning Algorithms

Now, we review some common metrics that are used to evaluate classification algorithms. Note that these measurements are computed on a set of datasets on which the performance was previously estimated by a cross-validation. One of the most popular ones is the **macro-averaged accuracy**. This metric denotes the standard average of the accuracies on the d individual datasets. Note that it may be problematic to simply aggregate measurements by averaging them. A brief consideration of this problem is given below and in Section 2.7.5.

In the following, p_i (n_i) denotes the covered and P_i (N_i) are the total number of positive (negative) examples for the i-th dataset out of a total of d datasets.

Note that the following measures are two different methods to compute a single performance estimate out of a series of individual measurements, i.e., the average accuracies given by the cross-validations on the d datasets.

$$\textit{macro-averaged accuracy} = \frac{1}{d} \sum_{i=1}^{d} \frac{p_i + (N_i - n_i)}{P_i + N_i}$$

A key disadvantage of this method is that the variance of the performances of the algorithms may differ considerably, and the differences in average performance may be dominated by the performance on a few high-variance datasets. Thus, we also consider **micro-averaged accuracy**, which assigns the same weight to each misclassified example. In effect, this method assigns a higher weight to datasets with many examples and those with few examples get a smaller weight. In any case, whenever we refer to accuracy we mean macro-averaged accuracy as this type of measure is much more common than the micro averaging.

Micro-averaged accuracy is the fraction of correctly classified examples in the union of all examples of the different datasets.

$$\textit{micro-averaged accuracy} = \frac{\sum_{i=1}^{d}(p_i + N_i - n_i)}{\sum_{i=1}^{d}(P_i + N_i)}$$

As there are large differences in the variances of the accuracies of the individual datasets, one could also focus only on the *ranking* of the algorithms and neglect the magnitude of the accuracy differences. Small random variations in ranking performance will cancel out over multiple datasets, but if there is a consistent small advantage of one algorithm over the other this will be reflected in a substantial difference in the average rank.

2.7.5 Evaluation of Performance Rankings

As we evaluate a large number of different settings, a key issue is how to combine the individual results into an overall performance. The problem of averaging the results of individual datasets is always eminent and one of the remedies is to use a significance test as described below. Significance tests of this type rely on average ranks rather than on average accuracy values which decreases the probability that a good result of an algorithm may be derived by chance. Another method is to give the accuracy per dataset. In this thesis, usually many datasets are used to evaluate the algorithms which makes it hard to present results on each of the individual datasets. Whenever it is possible, results on single datasets are given. But for the majority of experiments only the averaged results combined with a significance test are presented.

Usually, a simple comparison of the macro-averaged accuracy of several algorithms is not enough to make a solid statement about the performance of the algorithms. Commonly, significance tests are used to be able to decide whether or not one algorithm should be preferred over another. In significance tests the *null hypothesis* states that all algorithms are equal, i.e., have the same performance (e.g., the same macro-averaged accuracy in a cross-validation). Then, based on such a significance test, the null hypothesis can be either rejected (the algorithms differ significantly) or is accepted (the algorithms do not differ significantly). The test for significance is based on a *confidence level* which is the probability of an error, i.e., the test predicts that the algorithms differ but actually they do not. These so-called confidence levels are often 0.01, 0.05, or 0.1. There are many different significance tests, but most commonly, to compare different algorithms on different datasets, a Friedman-Test with a post-hoc Nemenyi-Test is used as suggested by Demšar [32]. To compute the critical F-value for the Friedman-Test average ranks are needed.

The **average rank** is the average of the individual ranks r_i on each dataset. All algorithms can be ranked after their macro- or micro-averaged accuracy on each dataset[7]. For algorithms that got an equal accuracy the rank was computed by averaging their individual ranks. For example, if four algorithms share rank two, three, four and five the rank for each of them would be $(2+3+4+5)/4 = 3.5$. The

[7] In this thesis only macro-averaged accuracies were used to compute the average ranks.

rank $rank_j$ is computed by averaging the ranks r_i of the j-th algorithm that were yielded on all datasets d.

$$rank_j = \frac{1}{d}\sum_{i=1}^{d} r_i \tag{2.5}$$

Then, based on the average ranks Friedman's χ_F^2 is computed.

$$\chi_F^2 = \frac{12 \cdot d}{z \cdot (z+1)} \left[\sum_j rank_j^2 - \frac{z \cdot (z+1)^2}{4} \right]$$

where z denotes the number of algorithms and d is the number of datasets. Note that the test makes only sense when the number of algorithms and datasets is large enough. As a rule of thumb, there should be at least ten datasets and five algorithms [32]. Otherwise, exact values have been computed and can be used.

Friedman's χ_F^2 statistic is rather conservative (see [32] for references). As a remedy, the F_F statistic was proposed [82]. It is computed as given below.

$$F_F = \frac{(d-1) \cdot \chi_F^2}{d \cdot (z-1) - \chi_F^2}$$

which is distributed with $z-1$ and $(z-1) \cdot (d-1)$ degrees of freedom. The critical values for the F-distribution can be found in the literature.

If the null hypothesis of the Friedman test is rejected, i.e., there is significance, a post-hoc test is performed. In this thesis, the Nemenyi test was used that outputs a so-called critical distance CD.

$$CD = q_\alpha \sqrt{\frac{z \cdot (z+1)}{6 \cdot d}}$$

where q_α is the critical value (α is the confidence level) which derives from a studentized range statistic. Note that either the values have to be divided by $\sqrt{2}$ or appropriate tables have to be used[8].

All algorithms that have average ranks that are within the critical distance are not significantly different. If the sum of the average rank of an algorithm and the critical distance is larger than the ranks of other algorithms, they are significantly different. Usually, the confidence level is the same as in the previous Friedman test. In Figure 4.2 one can see an example for such a graphical interpretation of the confidence levels of different algorithms. The main advantage of this method

[8] See, e.g., `http://nikolaos.kourentzes.com/files/Nemenyi_critval.pdf` (visited 2012-05-07)

is that the quality of different algorithms can be interpreted in a direct manner. Often, other statistical tests are much harder to interpret and there is no direct way to sketch the differences among several algorithms.

In Chapter 4 we will use two independent collections of datasets to tune and evaluate different heuristics. On each of them we will rank different heuristics in terms of their macro-averaged accuracy. Then we will need a metric to compare two rankings to make sure that the results are valid. In order to assess this validity, we will compute the *Spearman Rank Correlation* between the rankings of the various heuristics on these two sets.

Spearman Rank Correlation: Given two (averaged and rounded) rankings $rank_j$ and $rank'_j$ for the heuristics $h_j, j = 1,\ldots,z$, the *Spearman Rank Correlation* ρ is defined as

$$\rho = 1 - \frac{6}{z \cdot (z^2 - 1)} \sum_{j=1}^{z} (rank_j - rank'_j)^2$$

note that $rank_j$ is defined in Equation 2.5 and refers to a single number that is the average of the individual ranks of different datasets.

2.7.6 Pairwise Comparisons using the Sign Test

In Chapter 4 we also use a simple *sign test* provided by *Weka* [177]. The sign test is computed on the win/loss/tie statistics, i.e., based on a certain performance measure it can be counted how often one algorithm wins (loses or ties) against another algorithm. On the basis of a binomial distribution (that comes from the definition of the null hypothesis, i.e., that each algorithm will win in approximately half of all cases), a critical p-value can be computed. The p-value stands for the error probability for rejecting the null hypothesis. Note that ties are usually uncounted. More importantly, the substantiality of a win is also uncounted. Thus, if an algorithm only wins by a very low margin, the win will contribute in the same way as a clear improvement counts. Albeit there are several drawbacks [32], the sign test was used in this thesis to have a direct notion whether or not the results of Table 4.6 are significant (by including the p-value of the sign test). Of course, the findings presented there are additionally verified by more sophisticated significance tests.

3 The SeCo-Framework for Rule Learning

This chapter describes the SeCo-Framework our main tool to implement and analyze the different algorithms derived in this thesis. Its key feature is that it allows to configure algorithms with building blocks. The different parts of a general rule learning algorithm are separated and each one of them can be implemented using well-defined interfaces. The modular design of the framework makes it easy to use different configurations of various separate-and-conquer algorithms and the built-in evaluation tool serves as a collector for the output of the different algorithms. Therefore, implementing and evaluating algorithms is a simple task within this framework. The SeCo-Framework is based on an older version of the framework [161]. It still has interfaces to *Weka* [177] (`classifyInstance`, `buildClassifier`), so that it can be included as a rule learning algorithm in *Weka* (in `weka.classifiers.rules`). Also, some of the models (e.g., the class `Instances`) are adapted from *Weka*.

The chapter starts with the presentation of two algorithms. Based on them it is described how separate-and-conquer rule learning is realized in the SeCo-Framework. Section 3.2 deals with specificities of the framework. Here, a categorization of rule learners along three different dimensions is given. In Section 3.3 details about the architecture from a programmer's perspective are given. Section 3.4 shows a summary of all objects that can be configured in the framework. The different implementations that already exist are described. Additionally it is shown how different binarizations can be realized in the framework. The next section then illustrates how concrete algorithms can be implemented using the interfaces and methods described before. In Section 3.6 the evaluation package is shown. It is a component of the SeCo-Framework that is used to compare different algorithms. In Section 3.7 related work is listed and Section 3.8 summarizes the chapter.

3.1 Separate-and-conquer Rule Learning in the SeCo-Framework

Following [55], the Algorithms 3.1 and 3.2 show a generic version of a separate-and-conquer rule learner which can be instantiated into various specific algorithms. Note that in particular, it can simulate the three search strategies that we will describe and use in Chapter 5. The pseudo-code sketched in Algorithm 3.1 and 3.2

Algorithm 3.1 AbstractSeCo*(Examples)*

```
Theory ← ∅
Growing = SplitData (Examples, Splitsize)
Pruning = SplitData (Examples, 1-Splitsize)
while ExamplesLeftToCover(Growing) do
    Rule = FindBestRule(Growing,Pruning)
    Covered = Cover (Rule,Growing)
    if RuleStoppingCriterion (Theory,Rule,Growing) then
        exit while
    Growing = Growing \ Covered
    Theory = Theory ∪ Rule
Theory = PostProcess (Theory, Growing, Pruning)
return (Theory)
```

is an extension of the simple algorithms presented in Chapter 2 (Algorithm 2.1 and 2.2) to overcome the limitations of the rule learner introduced there.

In the beginning, the set of examples is split into a growing and pruning set to implement algorithms like Ripper (for a description see 3.5.3). Note that the split is disjunct, i.e., each example is either in the set *Growing* or in the *Pruning* set. The split is optional and most of the algorithms do not apply such a partitioning of the data that is usually used for a built-in pruning. After the split is completed, the outer loop is entered that lasts as long as positive examples are left in the dataset. The procedure FindBestRule searches the hypothesis space for a rule that optimizes a given quality criterion h (the conquer step). The method EvaluateRule is used to provide the evaluation of a candidate rule on a set of instances using this criterion. The name of the quality criterion (or heuristic) is denoted by a subscript of the letter h (all choices mentioned in Section 2.6 are implemented). A sorted list of candidate rules (*Rules*), which is initialized with an empty set of conditions is maintained. New rules will be inserted in appropriate places (InsertSort), so that *Rules* will always be sorted in decreasing order of the heuristic evaluations h. At each cycle, SelectCandidates selects a subset of these candidate rules, which are then refined using RefineRule. A refinement can be the addition or the deletion of an attribute-value test (a candidate condition). For nominal attributes, candidate conditions are formed by comparing the attribute to all possible values. The framework supports a check for equality, for inequality, and for both relations (cf. the property *nominal.cmpmode* of the *Refiner* in the algorithm descriptions in Section 3.5). Numerical attributes are tested against a splitpoint using the relations $<$ and $\geq$. These are calculated as the mean between two adjacent (previously sorted) values. Note that splitpoints are only tested when the class has changed. This is a reasonable choice because evaluating a point that separates examples with the same class is not beneficial. The calculation of splitpoints can become difficult

r_{c_1} :	TRUE $\rightarrow y = y_1$	[n=N=35,p=P=15]	$h_{acc} = -20$	h_{fgain} =?
	$\downarrow$			
r_{c_2} :	A=a $\rightarrow y = y_1$	[n=8,p=10]	$h_{acc} = 2$	$h_{fgain} = 8.89$
	$\downarrow$			
r_{c_3} :	A=a $\wedge$ B=b $\rightarrow y = y_1$	[n=2,p=8]	$\mathbf{h_{acc} = 6}$	$h_{fgain} = 4.21$
	$\downarrow$			
r_{c_4} :	A=a $\wedge$ B=b $\wedge$ C=c $\rightarrow y = y_1$	[n=0,p=2]	$h_{acc} = 2$	$\mathbf{h_{fgain} = 0.64}$

Figure 3.1: Example of the refinement process with heuristics *accuracy* and *foil gain*

when the class is numerical. Solutions to deal with numeric attributes in that case are given in Section 6.3.1.

However, each refinement (obtained by refining a candidate rule with a candidate condition) is evaluated and inserted into the sorted *Rules* list. If the evaluation of the *NewRule* is better than the best rule found previously, *BestRule* is set to *NewRule*. Figure 3.1 shows the refinement process of a single rule for *value*-heuristics (cf. Section 2.6) and for *gain*-heuristics (cf. Section 2.6.4). The key difference is that for the first the best rule during the whole refinement process is returned and for the latter simply the last rule is returned. Thus, when the heuristic *accuracy* (h_{acc}) is used the FindBestRule procedure will return rule r_{c_3} (the highest evaluation is marked in bold) and when *foil gain* (h_{fgain}) is used the last rule r_{c_4} will be returned.

As depicted in Figure 3.1 the learning of a single rule starts with the empty rule covering all examples (thus $p = P$ and $n = N$) where the class that is learned is fixed (here the class is $y = y_1$). Then all candidate conditions are evaluated by h and the best one is added to the rule. In the example the first condition is `A=a` yielding the candidate rule A=a $\rightarrow$ y=y_1. The next condition is `B=b` and so on as long as negative examples are covered.

Note that *gain*-heuristics inevitably rely on the two stopping criteria (namely the StoppingCriterion and the RuleStoppingCriterion, cf. Section 3.4) or on pruning (cf. Section 2.4) because otherwise a consistent theory is learned which often leads to overfitting.

After each refinement is evaluated once, FilterRules then selects the subset of the sorted rule list *Rules* that will be used in subsequent iterations, and, when all candidate rules in the set *Candidates* have been processed, returns the best encountered rule.

The main objective of FilterRules is to realize three unidirectional search strategies inside the procedure FindBestRule, namely *hill-climbing*, *beam search*, and *exhaustive search*. The method can be adapted by letting only the best b refinements pass to the next iteration. In a hill-climbing search only one candidate rule is refined, i.e., one refinement process is performed as depicted in Figure 3.1. In

Algorithm 3.2 FINDBESTRULE*(Growing, Pruning)*

```
InitRule = INITIALIZERULE (Growing)
InitVal = EVALUATERULE (InitRule)
BestRule = <InitVal, InitRule>
Rules = {BestRule}
while Rules ≠ ∅ do
    Candidates = SELECTCANDIDATES (Rules, Growing)
    Rules = Rules \ Candidates
    for Candidate ∈ Candidates do
        Refinements = REFINERULE (Candidate, Growing)
        for Refinement ∈ Refinements do
            Evaluation = EVALUATERULE (Refinement, Growing, h)
            unless STOPPINGCRITERION (Refinement, Evaluation, Pruning)
            NewRule = <Evaluation, Refinement>
            Rules = INSERTSORT (NewRule, Rules)
            if NewRule > BestRule then
                BestRule = NewRule
    Rules = FILTERRULES (Rules, Growing)
return (BestRule)
```

a beam search a predefined number of candidate rules are refined simultaneously, i.e., more than one refinement process is done. An exhaustive search evaluates every candidate rule that can be build from the data.

Hill-Climbing: Set $b = 1$ to let only the best refinement pass.

Beam Search: Set $b > 1$ so that only the b best refinements pass.

Exhaustive Search: Set $b = \infty$ so that all refinements pass. Note that this is an inefficient version of an exhaustive search, efficient implementations are described in Chapter 5.

Note that for hill-climbing and beam search, SELECTCANDIDATES will always return all rules, whereas an exhaustive search will only look at the first element in the sorted list *Rules*. The three search algorithms are discussed in detail in Chapter 5.

After the best rule is found, the main covering loop removes all examples that are covered by the learned rule from the training set (the *separate* step), and the next rule is learned on the remaining examples. As noted before, the two steps of finding the best rule, adding it to the theory, and removing the covered examples are repeated as long as positive examples are left in the training set. In the simplest version, this ensures completeness and consistency when the proper heuristic is used (cf. Section 2.2.2). One may relax these two constraints so that certain

degrees of incompleteness and inconsistencies are allowed. That is, if some pruning criteria hold. These are implemented in the functions RULESTOPPINGCRITERION, which prevents the addition of further rules, and STOPPINGCRITERION, which stops the refinement of the current candidate rule.

These two criteria and all other basic components are described in detail in Section 3.4. A description of the different ways to implement these abstract procedures is also given there.

3.2 Unifying Rule Learners in the SECO-Framework

Previously, a general algorithm was given [55] which consists of certain building blocks that enable to specify each of the three dimensions given in Section 2.2.5. In the following the basic building blocks or procedures of Algorithm 3.1 and 3.2 are introduced and assigned to the three dimensions.

Language Bias: Is implemented by the method REFINERULE. Possible comparators for the rules are $\neq$, $=$, and using both of them for nominal attributes, and $<$ and $\geq$ for numerical ones. Currently, only propositional rules can be learned. More complex types as, e.g., first-order rules are unavailable yet.

Search Bias: Can be defined by using a search strategy via the INITIALIZERULE and REFINERULE methods, a search algorithm (SELECTCANDIDATES and FILTERRULES), and a search heuristic h (a parameter of EVALUATERULE).

Overfitting Avoidance Bias: Either is implemented by the two stopping criteria (RULESTOPPINGCRITERION in the ABSTRACTSECO-algorithm or STOPPINGCRITERION in the inner loop) or is achieved by the POSTPROCESS-method of the outer loop. Can also be realized by using a heuristic h (which then is used to compute the evaluation by the method EVALUATERULE) that returns a rule that also covers negative examples (cf. Figure 3.1, where *accuracy* returns rule r_{c_3} that covers two negative examples).

The two algorithms 3.1 and 3.2 implement a general framework where most of the separate-and-conquer rule learners can be instantiated. In Section 3.5 some of the most popular algorithms are shown in their respective implementations using the SECO-Framework. Note that in its default setup the framework implements a simple rule learner that was used for many different experiments in this thesis. It is conveniently called SIMPLESECO (Simple Separate-and-Conquer) and is described in Section 3.5.4 and depicted in the configuration shown in Figure 3.10. This simple algorithm was used to optimize different parametrized heuristics (cf. Chapter 4), as the basis for a regression rule learner (cf. Chapter 6 and 7), and for different experiments with the search strategy (cf. Chapter 5). The SIMPLESECO algorithm is quite similar to the rule learner implemented by Algorithm 2.1 and 2.2 with the

only distinction that the refinement of a candidate rule is stopped as soon as no negative examples are covered any more.

The SeCo-Framework implements some general properties, i.e., independently from the current algorithm. These are mostly optimizations that do not affect any actual implementations.

3.2.1 Fixed Properties of the SeCo-Framework

Some pruning and optimization, respectively, is already done in both loops: the outer covering loop does not add the rule *BestRule* to the theory if the rule returned by FindBestRule is empty or if it covers fewer positive than negative examples. When the negative coverage is higher than the positive a rule will never increase the accuracy of the theory. The procedure FindBestRule stops refining the current rule *NewRule* if the StoppingCriterion holds or if an intuitive forward pruning criterion fires.

Forward pruning

Forward Pruning (also called *pruning with optimistic value* [170]) is used to cut off subtrees of the search space without losing performance. In the framework it works as follows: Assume the current rule *NewRule* covers p positive and n negative examples. The best hypothetical refinement of this rule would be a rule $FPrunedRule$ that covers p positives (still covers all positives) and zero negatives (excludes all negatives). Thus, if $h(FPrunedRule) \leq h(BestRule)$, *NewRule* is not inserted into *Rules*, and not further refined. In other words, if the current rule can be refined into a perfect rule given the current coverage statistics and still has a lower evaluation than the best rule so far, the refining is skipped and the next candidate rule is handled.

Missing values

Instances with missing class values are removed from the data in a pre-processing step. Examples where the value of an attribute is missing are counted as never covered. These choices are somewhat arbitrary as there are many other possibilities to deal with missing values. In the literature a comparison of eight different methods can be found [178]. As the results there show, the simple ignorance of instances containing missing instances (called *Ignore* [178]) does not perform significantly worse compared to the more sophisticated methods. The key advantage of this method is that nothing has to be changed in the learning algorithm. Interestingly, the *Ignore* strategy ranked on second place when used with the heuristic m-estimate [178]. Thus, it seems to be appropriate to rely on such a simple strategy (as the authors also conclude [178]). However, it is planned to include some of the more complex strategies in the framework in a later release.

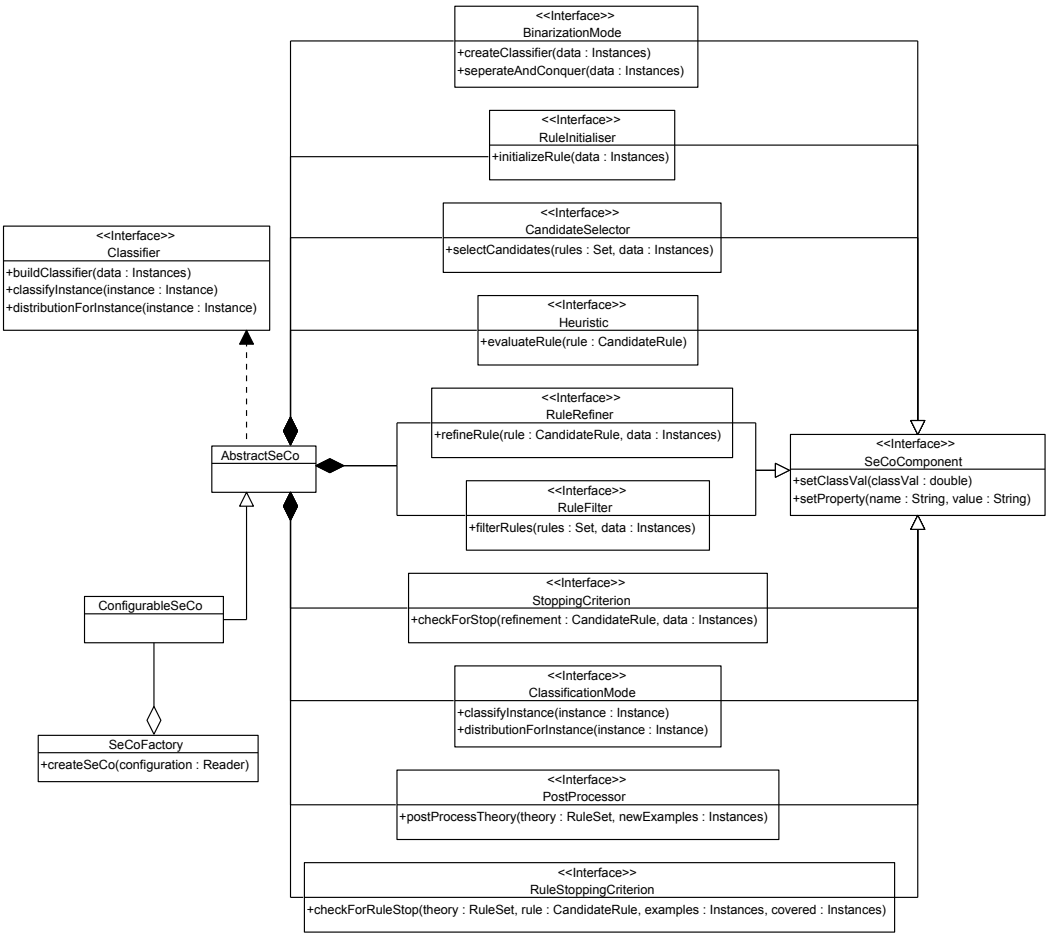


Figure 3.2: *UML* diagram of the SeCo structure

Tie breaking for rules with equal value

For rules with equal evaluation a tie breaking on the covered positive examples is done. Indeed, the rule with the higher positive coverage is chosen. If this tie breaking step still leads to equal values, the next means is to tie break on the length of the rules, where the shorter rule is selected. If the length is also equal the algorithm selects a rule randomly (but controllable by a seed parameter). The tie breaking procedure is especially important for heuristics that tend to overfit as *precision* does. For search algorithms that are more exhaustive this can become an important issue (cf. Chapter 9).

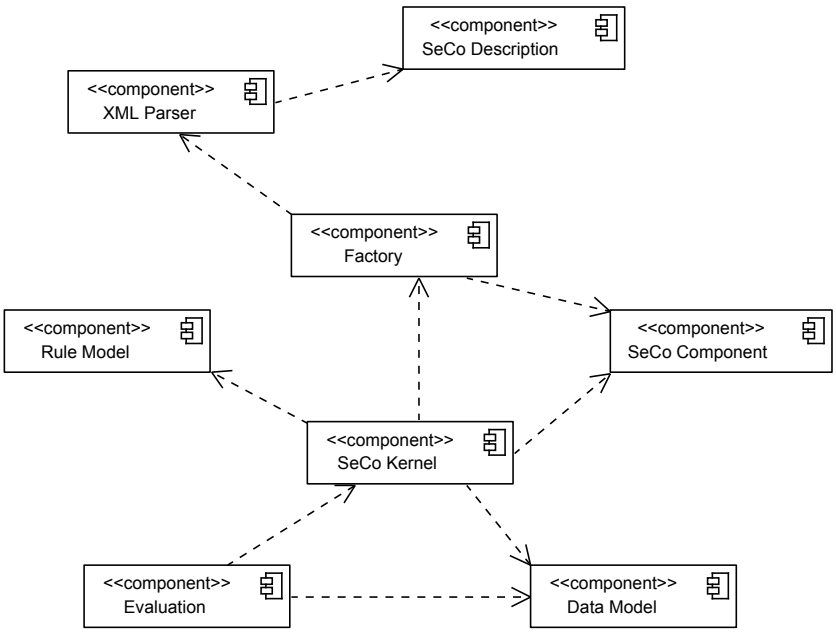


Figure 3.3: *UML* diagram of the SECO components

3.3 Architecture of the SECO-Framework

As mentioned before, the key idea of the framework is to provide different general building blocks. These building blocks were derived from the pseudo-code given in Algorithms 3.1 and 3.2. In the first step, the framework is built up from a specification of each of these blocks given in a *XML*-file. If a building block is unspecified, it will be initialized with the default entry for this entity. These defaults implement a solid choice for each of the building blocks. The default configuration is called SIMPLESECO (cf. Section 3.5.4). Each block is implemented via a Java interface and is described in detail in Section 3.4.

Usually the only information that is needed to start the framework is the path to a folder that contains dataset files. These files have to be in the .arff-format of *Weka* [177]. Other database formats are unsupported right now but there are many freely available converters that are able to convert different database formats into the .arff-format. The information which datasets should be evaluated is the only mandatory field in the framework. If it is indeed the only one given, the framework will initialize the default configuration and evaluate it using a 10–fold cross-validation (cf. Section 2.7.1) for each of the datasets.

In fact, in most cases a second information is given by the user: A path that points to a directory that contains *XML*-configurations of different algorithms. Both

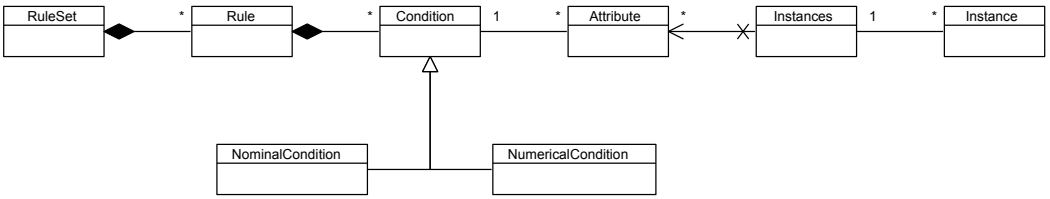


Figure 3.4: *UML* diagram of the SeCo data model

of these two setup informations are given in a property-file that is the input to the *Evaluation Framework* which is a stand-alone component of the SeCo-Framework. This evaluation framework currently can be configured in several ways to evaluate algorithms that are built in the SeCo-Framework. A detailed description of the framework is given in Section 3.6.

In Figure 3.2 a diagram in the *Unified Modeling Language (UML)* of the structure of the SeCo-Framework is shown. It uses the standard components of a *UML* diagram such as arrows depicting generalization of classes, hollow diamond shapes standing for aggregation, and filled diamond shapes are encoding composition.

As mentioned above the framework originally is based on *Weka* [177]. For this reason, there is still an interface to a Classifier class. Note that the framework is compatible to *Weka* versions below `3.6.5`. The SeCoFactory aggregates a ConfigurableSeCo which extends the actual classifier (or rule learner) called AbstractSeCo. This class holds all interfaces that are implemented either by a default setting or by implementations necessary for the current rule learner that should be configured in the SeCo-Framework. Each of those components itself is an ISecoComponent and is implemented by the class that is used for this certain interface.

The ISecoComponent has some basic methods. It is able to set the class value, i.e., which class in the dataset should be learned and it may set a property necessary for a certain interface. Thus, e.g., the IPostProcessor has to know how many optimizations it should employ which is given by a property (in the *XML*-file). Each of the classes implementing one of the interfaces has to implement a certain method. This could be a check for stop for adding rules to a rule set (*checkForRuleStop*) or a method that filters out rules (*filterRule*).

In Figure 3.3 a *UML*-diagram of the components is given and Figure 3.4 shows the data model of the SeCo-Framework. The SeCo-Framework consists of eight components as depicted in Figure 3.3. First, a description in *XML* is given to the factory which initializes the SeCo components by using the *XML Parser*. The kernel is the main entity of the framework and has access to the rule model as well as to the data model.

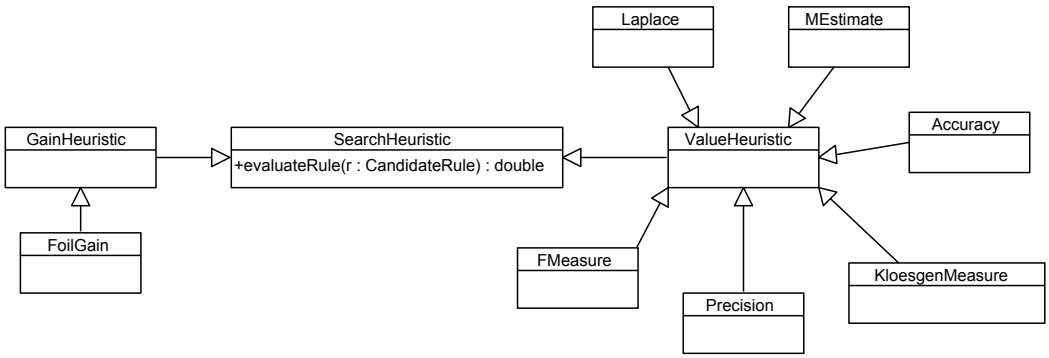


Figure 3.5: *UML* diagram of the SeCo heuristics

The data model depicted in Figure 3.4 consists of everything that is necessary to build a set of rules from a given set of instances. Both the Instance and the Instances classes are similar to those implemented in *Weka* [177]. The class Attribute handles the attributes of the .arff-file. The class Condition is able to build a numeric or nominal condition given an attribute. A rule consists of a finite number of those conditions. The class RuleSet is used to store the rules.

A *UML* description of the heuristics is given in Figure 3.5. These heuristics are used to evaluate candidate rules. A detailed description of the heuristics that are currently implemented in the SeCo-Framework is given in Section 2.6. The abstract class SearchHeuristic defines a heuristic by offering a method for evaluating rules (*evaluateRule*). Each heuristic extends this abstract class. Figure 3.5 only shows a subset of all heuristics (for a complete list see Section 2.6). Note that the parameter of parametrized heuristics can be configured via the *setProperty* method. If no parameter is given, all parametrized heuristics will be initialized with an optimal setting. What this optimal setting is and how it was derived is described in detail in Section 4.2.

3.4 Configurable Objects in the Framework

Table 3.1 lists the structure of an *XML*-configuration for the SeCo-Framework. The Table is an adapted version from a previous one [161]. As can be seen in Table 3.1, the configuration is given as a *seco* (the root element of the *XML*-file) that contains all the different assignments. A *secomp* is a SeCo component, in other words one of the interfaces of Algorithm 3.1 and 3.2. A *jobject* is used to specify additional classes for some of the SeCo components. *Properties* are specifications of an object, e.g., the number of optimizations for the post-processor or the parameter of the heuristic.

Table 3.1: Elements and attributes of a SECO description in *XML*

element	attribute	description
seco		the root element where the configuration is given
secomp		a SECO component, identified by a certain name
	classname	name of the class that implements the component
jobject		an arbitrary Java-Object
	classname	the class name
	setter	part of a string of the setter-method that aggregates the object with the one on the next level (if the method name is e.g., "setHeuristic", the setter has to be "heuristic")
property		sets the property of an object
	name	name of the property
	value	value of the property

In the following the interfaces of the ABSTRACTSECO procedure are explained. The focus here is to provide an overview of the different methods that are implemented. Their functionality is not explained in detail here but is postponed to the actual implementations given in Section 3.5.

EXAMPLESLEFTTOCOVER: Here, a decision is made whether there are examples left to cover, i.e., whether more rules are necessary, or whether all examples are already covered. In an ordered and unordered binarization all positive examples have to be covered by at least one rule (given that the rules all pass the stopping criteria). In the multiclass covering, a user-given percentage of all examples has to be covered.

SPLITDATA: This method is used to split the data into a growing and a pruning set. Some algorithms use a so-called REP or I-REP pruning strategy that requires a split of the dataset: The first part (usually $2/3$ of the dataset) is used to build a rule and on the other part (the remaining $1/3$) the pruning of the rule takes place. The decision whether or not pruning is used is determined in the configuration file by setting the option *growingSetSize*. A value of one means that no pruning is used (the default where *Growing=Pruning*) and values greater than one determine the number of examples for both sets. A number of four, e.g., means that $3/4$ of the examples are used for the growing set and $1/4$ is used for the pruning set.

FINDBESTRULE: The FINDBESTRULE-procedure is used to return the best rule that can be found given the configuration of the algorithm and the current dataset.

RULESTOPPINGCRITERION: It is provided by the *rulestoppingcriterion* secomp (a secomp is an element of the *XML*-description, cf. Table 3.1).

The different implementors are:
CoverageRuleStop: A rule has to cover more positive than negative examples.
DefaultRuleAndCoverageRuleStop: Is the same as before but a rule additionally has to be better than the default rule.
MDLStoppingCriterion: The minimum description length (MDL) rule stop for RIPPER (cf. Section 3.5.3). It is also able to check how many examples are covered to implement the multi class mode of the algorithm.

POSTPROCESS: Implemented by the *postprocessor* secomp. Here, all additional phases that are employed after learning a theory take place. One example is the optimization phase of RIPPER [22] where each rule is pruned one by one. Then, a MDL-based mechanism is used to decide which of three alternative rules is used for the current rule (cf. Section 3.5.3). This implementor is:
PostProcessorRipper: The post-processing method implemented in RIPPER with a default of two optimizations.

General options: These are global options, i.e., **integer** *growingSetSize*: sets the size for the growing and pruning set, **integer** *minNo*: determines how many examples have to be covered by each rule, **boolean** *weighted*: determines whether weighted covering is used or not, and **integer** *seed*: seed for the random number generator to ensure that experiments are repeatable.

In the following the methods of the FINDBESTRULE procedure are inspected. As before, all functions are described rather briefly.

INITIALIZERULE: Implemented by the secomp *ruleinitializer*. Depending on the search strategy that is used, a rule has to be initialized properly. Hence, for a top-down strategy the rule is initialized as a rule with empty conditions (i.e., the body TRUE), thus covering all examples. If a bottom-up strategy is used the rule would be initialized from a random positive example. This rule simply matches all attribute values that are given by the example. The implementors are:
TopDownRuleInitializer: Initialize a rule as empty rule.
RandomRuleInitializer: Initialize a rule based on an example (where the rule covers exactly this example).

SELECTCANDIDATES: The component *candidateselector* is responsible for deciding which Candidates are inspected during the search. Usually this is the part of the algorithm where duplicates are removed or where it is decided in which order the rules are refined. This is important when more extensive types of search algorithms as, e.g., an exhaustive search are used. The implementors are:
SelectAllCandidates: Simply select all candidates.

SelectBestCandidate: Only the best candidate is selected.
SelectStrictlyGreaterCandidatesSelector: Only select a candidate when its heuristic evaluation is bigger than the one of its direct predecessor. Note that this is important when a bidirectional search is used to avoid getting stuck in infinite loops (cf. Section 5.5.1).

REFINERULE: This method is implemented by the *rulerefiner* secomp. Here, the rule is refined by a refinement operator. Depending on the search strategy this operator will vary. For top-down it will implement a specialization of a rule by adding a condition and for bottom-up it will generalize a rule by removing a condition. The implementors are:
TopDownRefiner: Refine a rule is by adding conditions.
BottomUpRefiner: Refine a rule by removing conditions.
BidiretionalRefiner: Refine a rule either by adding or by removing conditions. It is described in Section 5.5.

HEURISTIC: The heuristic that is used to evaluate all candidate rules (cf. Section 2.6). The designated component is *heuristic*.

STOPPINGCRITERION: This method is used to check whether a candidate rule has to be refined any longer. The component is called *stoppingcriterion*. The implementors are:
NoNegativesCovered: Refine the rule as long as negatives are covered.
LikelihoodRatio: Implements the likelihood ratio statistics for CN2 (cf. Section 3.5.1).

FILTERRULES: The *rulefilter* component decides how many rules are refined at each cycle. Thus, in a hill-climbing scenario it will only return the best rule. If a beam search is employed it will return the b best rules (where b determines the size of the beam). When an exhaustive search is used it will return all rules. The implementors are:
BeamWidthFilter: Used to do a beam search.
MultiRuleFilter: Used to combine multiple filters.
ChiSquareFilter: Perform a χ^2-test for significance.

3.4.1 Binarization in the SECO-Framework

As discussed before, separate-and-conquer rule learning algorithms essentially solve the concept learning problem. To this end, their output is a rule set that is able to decide whether a given new example is part of the concept or not. Real-world problems often contain more than two classes, i.e., an example can have more than two states. To overcome this problem different schemes were introduced in Section 2.3, summarized there under the term binarization methods. The

Table 3.2: Combination of the binarization method and classification method

BinarizationMode	ClassificationMode
LearningByOrderedBinarization	ClassificationByDecisionList
LearningByMultiClassCovering	ClassificationByDecisionList
LearningByUnorderedBinarization	ClassificationByMajorityVoting
LearningByUnorderedBinarization	ClassificationByWeightedMajorityVoting

SeCo-Framework is also able to handle multi-class problems by using a binarization procedure as a pre-processing step. Thus, each call of the *separateAndConquer* method of the class AbstractSeco is conducted with a set of examples that are already converted into a binary problem.

In general, the SeCo-Framework is able to learn an ordered decision list, an unordered list of rules, and a decision list of rules where the class is not fixed in advance. The first one is the default method. Rule sets that are learned in this fashion contain blocks of rules that predict the same class because each class was learned in turn as described in Section 2.3.4. In an unordered list, rules also appear in blocks with the same class, but the ordering of them is neglected because all rules in the list are checked whether they cover the test example or not. Here, rules are learned for all classes including the most frequent class (which is ignored when a decision list is learned). To solve situations where more than one rule covers the example but both rules predict different classes, a voting mechanism is used. This can be simple majority voting where each rule has the same weight in the voting or weighted voting where the votes are currently computed by using the *Laplace* heuristic. This simple heuristic has good results among some other heuristics [157].

In the last case rules are learned by the so-called multiclass covering were subsequent rules may predict different classes. In the classification phase, the rule set is also used as a decision list, i.e., the first rule that covers the example is used to predict the class. Note that in this case, there is no natural stopping criterion because the binarization algorithm does not iterate over the classes (cf. the figures for the different class binarizations given in Section 2.3). Hence, a simple stopping criterion would be to stop inducing more rules when a certain number of examples are covered. We have experimented with this parameter and not surprisingly it seems to be the best choice to cover all examples. Nevertheless, a default rule is still necessary because examples in the test set may be uncovered anyhow.

Previously, a comparison of the three approaches was conducted [124]. The experiments were divided into a comparison on binary datasets and one on multiclass datasets. There are two dimensions in which the methods show different behavior:

```
<seco growingSetSize="1" minNo="1" weighted="false" seed="0">
    <heuristic classname="Laplace"/>
    <rulerefiner classname="TopDownRefiner" nominal.cmpmode="equal"/>
    <stoppingcriterion classname="LikelihoodRatio" threshold="0.9"/>
    <rulestoppingcriterion classname="CoverageRuleStop"/>
    <candidateselector classname="SelectAllCandidatesSelector"/>
    <rulefilter classname="BeamWidthFilter" beamwidth="5"/>
</seco>
```

Figure 3.6: *XML* configuration of CN2

- The number of rules and conditions, and
- the accuracy of the final model.

The expectation was that the multiclass covering and the ordered binarization should induce smaller rule lists whereas the unordered binarization should yield bigger rule sets. This actually is confirmed [124]. Due to the flexibility of the (weighted) voting approaches, the performance of these two approaches was higher compared to the other two approaches. However, in the remainder of the thesis, we concentrate on decision lists. In spite of the somewhat lower performance, a decision list has several advantages:

- The size of the rule sets,
- the simple and efficient classification phase,
- no need to define an artificial stopping criterion, and
- no need to re-compute the heuristic evaluations.

The abovementioned mechanisms are implemented in two interfaces called IBinarizationMode and IClassificationMode (cf. Figure 3.2). Both of them are not directly implemented in the AbstractSeCo procedure, but are used as wrapper methods, providing the algorithm with the correct examples. Table 3.2 displays the connections of the two interfaces and their implementors. Note that the default mode is to learn a decision list by using an ordered binarization.

3.5 Example Configurations

In the following some well-known algorithms are shown as their respective implementations inside the SeCo-Framework.

```
<seco growingSetSize="1" minNo="1" weighted="false" seed="0">
   <heuristic classname="Accuracy"/>
   <rulerefiner classname="AqrRefinerTopDown" negativeSelection="0.0"
      seedChoice="first"/>
   <stoppingcriterion classname="NoNegativesCoveredStop"/>
   <rulestoppingcriterion classname="AqrRuleStop"/>
   <candidateselector classname="SelectAllCandidatesSelector"/>
   <rulefilter classname="BeamWidthFilter" beamwidth="3"/>
</seco>
```

Figure 3.7: *XML* configuration of AQ

3.5.1 CN2

The famous CN2-algorithm[1] [20] uses a top-down approach with a beam size of five. The conditions of a rule test for equality of a certain attribute value. The heuristic used to select the best refinement is *Laplace*. CN2 employs a test on significance for each rule. The main idea here is to guarantee that a rule has a significantly different distribution of the examples it covers compared to the distribution of all examples. To check whether a rule is significant or not the likelihood ratio statistic is used [93]. Note that the combined use of *Laplace* and significance testing is questionable as previously explained [19]. Nevertheless, significance testing is also used in the implementation available online[2]. For this reason, the test for significance is also included in the SeCo-version of CN2. Figure 3.6 shows the corresponding configuration file.

As can be seen there the heuristic is changed from the original *entropy* to *Laplace*. As RuleRefiner the *TopDownRefiner* is used. All rules are selected by the SelectCandidates method and as RuleStoppingCriterion the *CoverageRuleStop* is used. This means that a rule is only added if it covers more positive than negative examples. As StoppingCriterion the *NoNegativesCoveredStop* is used. For filtering rules the *BeamWidthFilter* with a size of five is employed.

3.5.2 AQ

In Figure 3.7 a sample configuration of the AQ algorithm [113] is given. Most importantly the RuleRefiner is different from the implementation of CN2. AQ refines rules by taking a positive seed example and converting it into a rule. Now, in contrast to other algorithms, the set of possible refinements contains only those of

[1] Note that we implemented the improved version of Clark and Boswell [19].

[2] CN2 can be downloaded from `http://www.cs.utexas.edu/users/pclark/software/` (visited 2012-04-24)

```
<seco growingSetSize="3" minNo="2" weighted="false" seed="0">
    <heuristic classname="FoilGain"/>
    <rulerefiner classname="TopDownRefiner" nominal.cmpmode="equal"/>
    <stoppingcriterion classname="NoNegativesCoveredStop"/>
    <rulestoppingcriterion classname="MDLStoppingCriterion"/>
    <candidateselector classname="SelectAllCandidatesSelector"/>
    <rulefilter classname="BeamWidthFilter" beamwidth="1"/>
    <postprocessor classname="PostProcessorRipper" optimizations="2"/>
</seco>
```

Figure 3.8: *XML*-configuration of RIPPER

that rule. Then, a new rule is initialized as empty rule. The algorithm now chooses a negative example and tries to exclude it from the current rule by specializing the rule with the conditions generated from the positive seed example. This process runs as long as negative examples are covered by the rule. This kind of refinement procedure is implemented in the *AQRefinerTopDown*. In the default configuration of AQ no negative example has to be covered and the first positive example in the dataset is chosen as seed example. The beam size is set to three, the heuristic is *accuracy*, and a rule is only added if it is better than the default rule.

3.5.3 RIPPER

In Figure 3.8 the configuration of the RIPPER algorithm [22] is shown[3]. Here, the heuristic *foil gain* is used to select the best refinement. Note that this heuristic is the only *gain*-heuristic implemented in the SECO-Framework. It evaluates a rule in comparison to its predecessor (see Section 2.6.4). A rule is refined until it does not cover any more negative examples. Another important difference to the other algorithms is that a rule has to cover at least two examples. This requirement comes from the original implementation (cf. [22]) and is realized by the general property *minNo*. The main difference to the other two algorithms is that in RIPPER a split of the examples into a growing and a pruning set is done and that a post-processing is employed. The size of the split is given by the *numFolds* property. In RIPPER, $^2/_3$ of the examples are used to build the growing set and $^1/_3$ of the examples are taken for the pruning set. This is done to learn a rule on the growing set and prune it directly afterwards on the pruning set. This strategy is known as *Incremental Reduced Error Pruning* (I-REP) [61] and is described in Section 2.4. The stopping criterion for adding a rule to the rule set is different here. Hence, a MDL-based stop is performed. However, RIPPER also works in a top-down fashion. Therefore, the new rule is initialized as one without any condition. Despite the

[3] Note that we implemented the *Weka* version of RIPPER named JRIP.

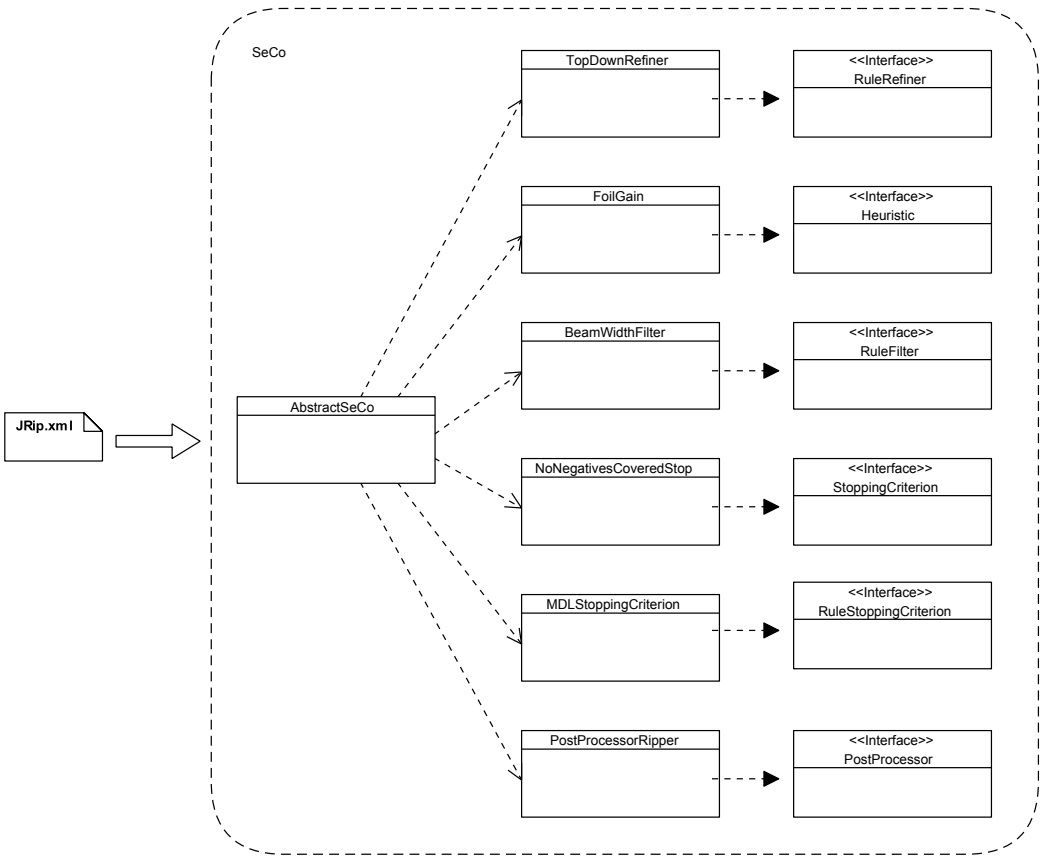


Figure 3.9: *UML* diagram of the implementation of RIPPER in the SECO-Framework

two other algorithms, this one uses a simple hill-climbing search (referred to as a beam search with size one). Whenever the learning of a rule set is finished, RIPPER applies an optimization phase as post-processing step. The MDL and the optimization phase are described below.

For clarification purposes, in Figure 3.9 an *UML* diagram of the RIPPER algorithm as implementation in the SECO-Framework is shown. Each of the interfaces is implemented by the classes specified in the *XML*-file given in Figure 3.8. Note that the *growingSetSize* is set to three, meaning that the dataset is split into a growing set ($^2/_3$) and a pruning set ($^1/_3$) and therefore the I-REP strategy is used to prune rules right after they are learned.

Table 3.3: The 17 datasets used for experiments with the optimization phase, nominal attributes are abbreviated with A_{nom}, numerical ones with A_{num}, and t is the number of instances

group	dataset
only A_{nom}	breast-cancer titanic vote-1
$\|A_{nom}\| > \|A_{num}\|$	tic-tac-toe cleveland-heart-disease horse-colic hypothyroid hepatitis vote
huge t	audiology krkp sick-euthyroid
only A_{num} or $\|A_{num}\| > \|A_{nom}\|$	anneal glass2 iris wine soybean

Minimum Description Length

The minimum description length has its origin in information theory [141]. The idea is to use a hypothesis to compress the given input data. Thus, the better the hypothesis reflects any regularities in the data, the less bytes are needed to compress the data. Additional background on the mathematical formulation of the MDL principle can be found in the literature [71]. Note that the MDL can also be used for model selection purposes. The selection then can be done by choosing the model with the lowest MDL.

The MDL-based stopping criterion in RIPPER basically calculates the description length of the examples and the rule set containing the new rule. Then, the rule is included when the computed description length is 64 bits shorter than the smallest description length that could be reached until this stage of building [22]. The idea to use MDL as stopping criterion for rule learning was derived by Quinlan [133].

Optimization Phase

The optimization of a rule set works incrementally by learning each rule completely new (called *Replacement*), by adding more conditions to it (called *Revision*), and by leaving it untouched. From the resulting three rules the best one is used to replace the original rule in the rule set. This best rule is determined by the MDL metric where the MDL for the whole theory is calculated for each variant. Then, for each of the three variants, the MDL of the rule set where the current rule is replaced with one of those variants is compared to the MDL where the original rule is used. To this end, the variant that results in the lowest MDL for the rule set is used.

By default RIPPER applies two optimizations of the rule set. To ensure that this is a good choice, we have experimented with the number of optimizations [38]. In this work, 17 datasets were used to figure out in what conditions which number of optimizations works best. Table 3.3 lists the datasets. They are described in Table 4.1. To gain a better understanding of the conditions in which the optimization phase works best, the datasets can be divided into four different groups. The first group contains three datasets that only have nominal attributes. The trend here is that the theory learned by RIPPER could not be improved by the optimization phase at all. The second group contains six datasets that contain more nominal than numeric attributes. In this group the optimization phase can improve the rule sets up to a certain number of optimizations and then the accuracy drops again. Usually in this group of datasets two optimizations seem to be enough when the gain in accuracy and the effort necessary for additional optimizations are related to each other. The third group contains three datasets that have a huge number of instances (t). Here, using more than two optimization seems to be beneficial. The last group contains five datasets that only have numerical or more numerical than nominal attributes. In this group more optimizations also stand for higher accuracy.

Note that the gain in accuracy is negligible. A clear identification in what conditions the optimization phase is profitable was impossible. Due to the recommended value of two optimizations and the diversity of the datasets that are used in this thesis, the decision was to use the default value.

3.5.4 SIMPLESECO

The SIMPLESECO algorithm represents the default rule learner of the SECO-Framework. It will be mainly used in Chapter 4 to instantiate an algorithm with which the parametrized heuristics introduced in Section 2.6.3 will be optimized. The experiments with the search algorithm conducted in Chapter 5 are based on the SIMPLESECO algorithm. It also forms the basis of the algorithm sketched in Section 6.3 that is used to learn rules for regression data. SIMPLESECO was used as

```
<seco growingSetSize="1" minNo="1" weighted="false" seed="0">
    <heuristic classname="MEstimate"/>
    <rulerefiner classname="TopDownRefiner" nominal.cmpmode="equal"/>
    <stoppingcriterion classname="NoNegativesCoveredStop"/>
    <rulestoppingcriterion classname="CoverageRuleStop"/>
    <candidateselector classname="SelectAllCandidatesSelector"/>
    <rulefilter classname="BeamWidthFilter" beamwidth="1"/>
</seco>
```

Figure 3.10: *XML* configuration of SIMPLESECO

base learner for the regression by classification scheme (cf. Section 7.3). When the framework is invoked without a *XML*-configuration file, the SIMPLESECO learner will be instantiated using the heuristic m-estimate.

As the configuration given in Figure 3.10 unfolds, the algorithm is similar to CN2 with three exceptions. First of all, the heuristic is different by using the m-estimate instead of *Laplace*. Secondly, there is no significance test here, which means that a rule is simply added if it covers more positive than negative examples, independently whether its coverage statistics differ significantly from the total distribution of positive and negative examples or not. Lastly, the beam size is reduced to one yielding standard hill-climbing search. The reason for this setting mostly stems from efficiency considerations, but, however, it is also confirmed to be a reasonable choice in Chapter 5.

Discussion of the SIMPLESECO-learner

As mentioned above, for the majority of experiments in this thesis the rule learner SIMPLESECO was used. For this reason the algorithm is discussed in more detail. We want to stress that this algorithm is quite typical for commonly used covering algorithms. In particular, it is more or less identical to the second version of the popular CN2 [19] algorithm as highlighted before. The main difference lies in the class binarization. CN2 can be used in two different modes: an *unordered* mode, which learns rules for each class, always using all other classes as the negative examples, and a *decision list* mode, which is able to learn rule lists with arbitrary class assignments (cf. Section 2.3.4). Other differences include that CN2 is unable to handle missing class values and those described above.

In this thesis, we wanted to gain a principled understanding of what constitutes a good evaluation metric for inductive rule learning for both classification and regression tasks. Therefore, we did not employ explicit stopping criteria or pruning techniques for overfitting avoidance, but solely relied on the evaluation of the rules by the used rule learning heuristic. Note, however, that this does not necessarily

mean that we learn an overfitted theory that is complete and consistent on the training data (i.e., a theory that covers all positive and no negative examples), because many heuristics will prefer impure rules with a high coverage over pure rules with a lower coverage (cf. Figure 3.1).

The SIMPLESECO-algorithm is also quite similar to the FOIL [132] algorithm, which forms the basis of many rule learning algorithms, most notably RIPPER [22]. The key difference here is that FOIL-based algorithms do not evaluate refinements on an absolute scale, but relative to their respective predecessors, i.e., they focus on the *gain* that a rule obtains in comparison to its predecessor. While this is a reasonable approach, gain-based algorithms cannot directly compare the evaluation of two rules with different predecessors, and are therefore unable to identify the best rule encountered during the search. Instead, they always return the last rule searched (cf. Section 2.6 and Figure 3.1). Thus, their performance crucially depends on the availability of a pruning heuristic or a stopping criterion, which determines when the refinement process should stop (cf. Section 2.6 and Section 3.4). FOIL uses a heuristic based on minimal description length for this purpose [132, 59], whereas RIPPER employs the I-REP technique, which prunes each rule after it has been learned [61, 54]. On the other hand, algorithms that can be implemented in the SECO-Framework return the last rule searched when a *gain*-heuristic is used and the rule with the highest evaluation during the search when *value*-heuristics are used. In the latter case, a stopping heuristic assumes the role of a filtering criterion, which filters out unpromising candidates, but does not directly influence the choice of the best rule [19]. Because of this dependency on stopping criteria, we do not further consider *gain*-heuristics in this thesis. However, we note that an empirical study comparing *gain*- to *value*-heuristics is an open research question.

3.6 Evaluation Package

In this section the *Evaluation Framework* is described in detail. It is used to evaluate different configurations of the SECO-Framework. There are five different modes to evaluate a rule learning algorithm:

1. Evaluate a previously learned rule set (*existingModelEvaluation*),
2. learn and store a rule set (*createAndSaveModel*),
3. learn a rule set on a part of the training data and evaluate it on the remaining examples of the training data (*trainTestSplit*)
4. learn a rule set on training data and evaluate it on existing test data (*trainTestValidation*), and
5. using cross-validation when no test data is available (*crossValidation*).

```
<validationProperty
    interact="false"
    reportState="true"
    log="true"
    logFile="d:\logs\evaluation_log.txt"
    statisticalTests="true">

    <createAndSaveModel
        trainDataFolder = "d:\data\Arff\trainingDatasets"
        configurationFolder = "d:\configurations"
        saveModelToFolder = "d:\savedModels">
    </createAndSaveModel>

    <existingModelEvaluation
        testDataFolder = "d:\data\Arff\testDatasets"
        savedModelFolder = "d:\savedModels">
    </existingModelEvaluation>

    <trainTestSplit
        dataFolder = "d:\data\Arff\trainingDatasets"
        splitPercentage = "80"
        configurationFolder = "d:\configurations">
    </trainTestSplit>

    <trainTestValidation
        trainDataFolder = "d:\data\Arff\trainingDatasets"
        testDataFolder = "d:\data\Arff\testDatasets"
        configurationFolder = "d:\configurations">
    </trainTestValidation>

    <crossValidation
        seed = "1"
        passes = "1"
        folds = "10"
        stratified = "true"
        dataFolder = "d:\data\Arff\trainingDatasets"
        configurationFolder = "d:\configurations">
    </crossValidation>

    <output
        format="csv"
        path="d:\results\results.csv">
    </output>
</validationProperty>
```

Figure 3.11: Property-file of an evaluation configuration

All of the modes are depicted in the configuration file given in Figure 3.11. Usually only the desired method should be included but, in any case, the last configuration present in the *property*-file will be used. In the case of an evaluation of an existing model, a path to a folder containing these models is mandatory. Additionally, a path to a folder where the test datasets are stored has to be given. To store a rule set that was learned on a given dataset the *createAndSaveModel* procedure is used. Here, the training datasets, the folder that contains the configurations of the SECO-Framework, and the folder where the rule sets should be stored has to be given. Note that the mandatory fields are summarized in Table 3.4.

A simple method for evaluating a model is to use the *trainTestSplit*. Here, the folder containing the datasets and the percentage for the training data has to be given. Then, for each configuration provided in the configuration folder a rule set

Table 3.4: Fields of the evaluation

field	createAnd-SaveModel	existing-Model-Evaluation	trainTestSplit	trainTest-Validation	cross-Validation
trainDataFolder	X			X	
testDataFolder		X		X	
dataFolder			X		X
configurationFolder	X		X	X	X
saveModelToFolder	X				
savedModelFolder		X			
seed					X
passes					X
folds					X
stratified					X

is learned on the given percentage of training instances and evaluated on the rest of the data. As emphasized before, the configuration folder is not mandatory. If no configuration is given, the framework will solely use the default algorithm. This setting is valid whenever configurations of the SeCo-Framework are given. Note that the evaluation framework is able to parse a whole folder and skips all files that do not contain a proper *XML*-configuration.

In the fourth case, namely the *trainTestValidation*, two folders containing datasets have to be present. One containing the training datasets and one specifies where the test datasets reside. The filenames should end with `_train` in the first folder and with `_test` in the second one. In the fifth case a regular cross-validation as described in Section 2.7.1 is applied. To do so, a path to a dataset folder has to be present. In the same way as before, a folder containing the configurations can be included. The cross-validation has some additional fields to specify its parameters. The *seed* is used to control the random partitioning of the data into the folds. The motivation behind this is to ensure that experiments are reproducible by using the same seed and consequently also the same pseudo-random partitioning of the folds. The parameter *passes* denotes how often the cross-validation should be repeated with a different segmentation of the data. *Folds* is used to specify the number of folds. The field *stratified* determines whether the partitioning should be stratified, i.e., ensuring that the same number of examples for each class are present in each fold, or whether the folds are totally random.

Additionally to the specification of the five different modes, several other properties can be described in the properties-file which is given to an instance of the *Evaluation Framework*. In the beginning of the configuration file in Figure 3.11, some general parameters are set. These include the *reportState*, which, if turned

on, informs the user in a separate window about the status of the current evaluation. It is also possible to use logging by turning *log* on. If so, *logFile* denotes the path to the file that should be used for logging. The most important parameter is *statisticalTests*. If statistical tests are used, the framework automatically runs a Friedman-Test. When it is successful, i.e., if the average ranks of the used algorithms are significantly different, a post-hoc Nemenyi-Test is employed. The output of the statistical test is the confidence level with which the Friedman-Test succeeded and the *critical distance* of the Nemenyi-Test. This type of significance test was suggested by Demšar [32] and is described in Section 2.7.5. If the parameter "interact" is true information about possible errors that may have happened during the evaluation is displayed to the user. These could be that a configuration is not capable to learn the current dataset due to some restrictions, e.g., that the algorithm is unable to handle numeric attributes, that the data folder contains datasets that are malformed, or similar problems.

The last parameter that has to be specified in the *property*-file of the evaluation is the output mode. Here, the user can define if the output of the classifiers (i.e., the rule set, the number of rules/conditions, the macro/micro-averaged accuracy and so on) should be displayed in the Java console or if it is desired to write these statistics into a file. This could either be a standard (unformatted) text file (similar to the console output) or a comma-separated file (*csv*) which can be imported into programs that are able to interpret those file formats (e.g., Open Office). For convenience, all statistics computed during the evaluation are listed in the *csv*-file so that one is able to select a subset of the information that is interesting for the desired task.

The evaluation also offers a summary at the end of a complete run. Here, the macro- and micro-averaged accuracy (cf. Section 2.7) of all configurations on all datasets are shown. Additionally a ranking is computed by averaging the ranks that the configurations achieved on each dataset (ranked by their macro-average accuracy). If two algorithm share the same rank, the same scheme as discussed in Section 2.7.5 is utilized. Information about the training performance and the test performance (whatever method it was created from) is shown. The configurations are identified by numbers to save space and are ranked by their macro-average accuracy in the validation. The abbreviation with numbers stems from the situation that for each configuration the implementors as described in Section 3.4 are listed, which can consume much space.

3.7 Related Work

There has not been much work on general frameworks for rule learning. The popular frameworks for data mining are focused to provide many different algorithms including rule learning algorithms, but without concentrating on the implementation of them. Usually, the objective there is to provide solid means for data mining

and machine learning in all their aspects (cf. for instance *Weka* [177] or *Rapid Miner* [114]). The SeCo-Framework is intended to unify various rule learning algorithms and to focus solely on rule learning.

However, some people have worked on frameworks particularly designed for rule learning. For instance, a general framework for learning an ensemble of decision rules exists [29]. The framework allows to change the loss function and the optimization procedure. Nevertheless, it is an ensemble technique and based on boosting [48, 148]. Chevaleyre and Zucker proposed a framework for learning rules from multiple instance data [17]. It is also shown how Ripper can be adapted to the task of learning multiple instances. In the literature a framework for learning rules from data can be found [4]. Despite of the general title, the task of learning rules is tackled by a neural network that pre-processes the data. After this pre-processing step the rules are learned via a *PAC-like* algorithm [166]. *PAC* stands for *probably approximately correct learning* where the goal is to select a hypothesis that will have a low generalization error with high probability.

The most promising approach, however, is given by Giacometti et al. [69]. The framework is very general and it is shown that nearly all rule learning algorithms can be implemented. However, it seems that there is no actual implementation yet. A comparison of the implementations and the original algorithms is also missing. Despite its closeness to our approach, the paper focuses more on a procedure to incrementally construct a global model (called *IGMA* there).

The Orange [33] data mining framework, on the other hand, provides different basic classes for rule learning among a collection of various data mining algorithms. Analogously to the SeCo-Framework the process of learning a rule set is separated into different components. These components are more or less identical to those defined in the SeCo-Framework. The key difference is that Orange does not implement many algorithms yet. Nevertheless, an implementation of CN2 is available. However, many major components the SeCo-Framework consists of are missing in Orange. Thus, a separation into a *growing* and *pruning* set is impossible, a post-processing of rules is unavailable, the binarization is not configurable, and some other very basic functionalities of the SeCo-Framework are also missing.

Another approach is to decouple the search for a single rule (a local pattern) and the combination of multiple rules (global model). Essentially, it seems obvious that those two parts need to be considered separately. The resulting framework is called LeGo [98, 58]. In essence, three different phases that form the basis of the LeGo approach are identified:

Local Pattern Discovery: A set of candidate patterns that do not have to meet any quality criterion is produced.

Pattern Set Discovery: A quality criterion is applied to the result of the first phase to identify meaningful patterns.

Global Modeling: Turn the pattern sets into well-balanced global models.

The SeCo-Framework does not decouple these three phases. Instead, they are tightly integrated. Essentially, the procedure FindBestRule serves as collector for patterns by building candidate rules from the data. These candidate rules are stored in the sorted list *Rules* (cf. Algorithm 3.2). The second phase of the LeGo approach is done by selecting a certain number of best candidate rules for further refinement. Consequently, these are the candidate rules with the highest evaluation values. The global modeling is on the one hand done by removing the covered examples after a new rule is learned and on the other hand by using a decision list. Alternatives include unordered rule lists or pairwise models as described in Section 2.3.4. Another approach would be to use weighted covering where the instances covered by a rule are not entirely removed but their weight is only reduced [23, 175, 66]. This strategy is also implemented in the SeCo-Framework.

Note that the experiments of Chapter 5, where more complex search algorithms are used, can be seen as other choices for the phase of the discovery of pattern sets. In this sense, the outcome of the *local pattern discovery* phase could be altered by employing different search algorithms. For example, the chance of getting stuck in a local optimum diminishes when more refinement paths are evaluated. For a detailed discussion of this topic see Chapter 5.

3.8 Summary

In this chapter, the SeCo-Framework for rule learning was introduced. The architecture of the framework was presented and illustrated by *UML* diagrams. Here, the versatility of the framework was shown and interfaces to extend the framework were presented. Furthermore, it was shown how the different components can be implemented. For some well-known rule learning algorithms, sample configurations within the SeCo-Framework were given. These algorithms were also discussed with a special focus on a comparison to the SimpleSeCo rule learner as it forms the basis for most of the experiments performed later in this thesis. The configurations illustrated that it is easy to extend the framework by implementing new components instead of a whole new algorithm. The framework is consummated by an evaluation component that eases the comparison of several algorithms.

4 Heuristics for Classification

In this chapter we concentrate on heuristics for classification. First, for some parametrized heuristics the parameter is optimized via a parameter tuning framework. Then, a completely new heuristic is learned with a meta-learning approach. This chapter is based on the papers [84, 85, 86, 88].

The main goal addressed in this thesis is to understand the properties of rule learning heuristics, that will allow them to perform well in a wide variety of datasets. We consider classification heuristics in this chapter and heuristics for regression in Chapter 6 and 7. Although different classification rule learning algorithms use different heuristics, there has not been much work on trying to characterize their behavior. Notable exceptions include [104], in which weighted relative accuracy was proposed as a novel heuristic, and [60], in which a wide variety of rule evaluation metrics were analyzed and compared by visualizing their behavior in ROC space. There is also some work on comparing properties of association rule evaluation measures (e.g., [159]) but these have different requirements than classification rules (e.g., completeness is not an issue there).

In this chapter, we approach this problem empirically. We will first empirically compare and analyze a number of known rule learning heuristics (Section 4.2). Rule learning heuristics, in one way or another, trade off consistency and coverage. On the one hand, rules should be as consistent as possible by only covering a small percentage of negative examples. On the other hand, rules with a high coverage tend to be more reliable, even though they might be less precise on the training examples than alternative rules with lower coverage. An increase in coverage of a rule typically goes hand-in-hand with a decrease in consistency, and vice versa. In fact, the conventional top-down hill-climbing search for single rules follows exactly this principle: starting with the empty rule, conditions are greedily added, thereby decreasing coverage but increasing consistency (cf. Algorithm 3.2).

The trade-off between consistency and coverage is also related to the *Bias/Variance Dilemma* [67]. Essentially, the worse a function fits data, the lower the variance will be. In contrast, the better the fit of a function to the data is, the higher the variance will be. In this sense, the more consistent a rule is, i.e., the better it fits the examples, the higher is its error on unseen data, i.e., the higher the variance will be. Contrarily, the more examples a rule covers the worse it predicts the target class but the lower the variance on new data will be. Similarly to rule learning, it is desired to have a low error and a low variance, i.e., a high consistency and a high coverage. Where usually the bias and the variance can be measured for different loss functions [83], this is not directly possible for rules. One may relate

the loss function, i.e., measuring how good an algorithm is, to the rule evaluation metric. Then, the problem of determining the best trade-off can be approached empirically. Consequently, we will show that five well-known rule evaluation metrics (a cost trade-off, a relative cost trade-off, the m-estimate, the F-measure, and the *Klösgen measure*, cf. Section 2.6.3) provide parameters that allow to control this trade-off. In an extensive experimental study – to our knowledge the largest empirical comparison of rule learning heuristics to date – we aimed at determining optimal values for each of their respective parameters. We will compare these settings to standard heuristics and show that the new settings outperform the fixed consistency/coverage trade-offs that are commonly used as rule learning heuristics. By testing the performance of the optimized heuristics on an additional selection of datasets not used for optimization, we will ensure that this performance gain is not due to overfitting the training datasets.

However, optimizing parameters constrains the candidate heuristics to known functional shapes. Consequently, we will try to leave these constraints behind (Section 4.3) and try to discover entirely new heuristics. The key idea is to meta-learn such a heuristic from experience, without a bias towards existing measures. Consequently, we created a large meta dataset (containing information from which we assume that the "true" performance of a rule can be learned) and use various regression methods to learn to predict this performance. On this dataset, we learn an evaluation function and use it as a search heuristic inside our implementation of a simple rule learner. We report on the results of our experiments with various options for generating the meta datasets, with different feature sets and different meta-learning algorithms. In particular, we try to evaluate the importance of rule length as an additional feature and consider a delayed-reward scenario where the learner tries to predict the performance of the *completed* rule from its incomplete current state in the search space.

In essence, the research presented in this chapter is a direct follow-up to the paper [58]. In this paper, several research issues were posed that arise in the context of separate-and-conquer algorithms. One of them is the trade-off between precision and coverage. The authors have given some settings for such a trade-off. Additionally, it was also mentioned that the optimal trade-off yet needs to be determined. The vision of a meta-learning heuristic was also posed in the paper. Basically, it summarizes the research conducted in the paper [57]. Later in this chapter we will extend the framework provided in this paper and use it mainly for generating the meta data. We also try to learn a function that models the out-of-sample precision, but our main goal is to provide stable heuristics that are learned without a bias imposed from settings as the precision/coverage trade-off mentioned above.

Some first answers to these questions are given in this chapter. Albeit we do not claim that the research presented in this chapter is a complete solution of these issues, it can be seen as a first step towards optimal rule learning heuristics.

Table 4.1: The 27 tuning datasets

dataset	# instances	# nominal attributes	# numerical attributes	# classes
anneal	798	32	6	6
audiology	226	69	0	24
breast-cancer	286	9	0	2
cleveland-heart-disease	303	7	6	5
contact-lenses	24	4	0	3
credit	490	4	0	3
glass2	163	0	9	2
glass	214	0	9	2
hepatitis	155	13	6	2
horse-colic	368	15	7	2
hypothyroid	3,163	18	7	2
iris	150	0	4	3
krkp	3,196	36	0	2
labor	57	8	8	2
lymphography	148	15	3	4
monk1	124	6	0	2
monk2	169	6	0	2
monk3	122	6	0	2
mushroom	8,124	22	0	2
sick-euthyroid	3,163	23	7	2
soybean	683	35	0	19
tic-tac-toe	958	9	0	2
titanic	2,201	3	0	2
vote-1	435	15	0	2
vote	435	16	0	2
vowel	990	0	9	11
wine	178	0	13	3

We will start the chapter with the experimental setup (Section 4.1). Section 4.2 summarizes the research on the optimization of rule learning heuristics, while Section 4.3 provides insights in the meta-learning approach. Then, after the empirical part is finished, related work is given in Section 4.4 and the chapter is concluded in Section 4.5.

4.1 Experimental Setup

The primary goal of our experimental work is to derive search heuristics that are optimal in the sense that they will result in the best overall performance on a wide

Table 4.2: The 30 validation datasets

dataset	# instances	# nominal attributes	# numerical attributes	# classes
auto-mpg	398	2	5	4
autos	205	10	15	7
balance-scale	625	0	4	3
balloons	76	4	0	2
breast-w	699	0	9	2
breast-w-d	699	9	0	2
bridges2	105	10	1	2
colic	368	15	7	2
colic.ORIG	368	20	7	2
credit-a	690	9	6	2
credit-g	1,000	14	6	2
diabetes	768	0	8	2
echocardiogram	74	1	7	2
flag	194	17	10	6
hayes-roth	132	4	0	3
heart-c	303	7	6	5
heart-h	294	7	6	5
heart-statlog	270	0	13	2
house-votes-84	435	16	0	2
ionosphere	351	0	34	2
labor-d	57	16	0	2
lymph	148	15	3	4
machine	209	0	7	8
primary-tumor	339	17	0	22
promoters	106	57	0	2
segment	2,310	0	19	7
solar-flare	333	10	0	8
sonar	208	0	60	2
vehicle	846	0	18	4
zoo	101	17	0	7

variety of datasets. Thus, we have to keep several things in mind. First, our results should be valid for a large collection of datasets with different characteristics. Second, we have to be careful not to overfit the selected datasets. Finally, we have to select ways for assessing the performance of a heuristic. In this section, we will describe our choices for addressing these concerns.

We used the SimpleSeCo algorithm described in Section 3.5.4 for all experiments presented in this chapter. Here, only the heuristics were exchanged while the other components of the algorithm stay the same (except when Ripper is configured).

Thus, no dedicated pruning is used, candidate rules are built in a top-down fashion and are refined as long as negative examples are covered. A candidate rule is only added to the rule set if it covers more positive than negative examples.

4.1.1 The Datasets

We arbitrarily selected the following 27 *tuning datasets* from the UCI-Repository [46]. Table 4.1 gives a summary of these datasets. The goal was to ensure that many different domains are included and that the characteristics of the datasets differ in terms of number of instances, number of numerical/nominal attributes, and the number of classes.

Only these datasets were used for making comparative choices between different heuristics (e.g., for optimizing a parameter of a heuristic, or for meta-learning a heuristic).

To check the validity of the optimization results, we selected 30 additional *validation datasets* applying the same selection process as used for the 27 datasets.

These datasets were used for validation only, no choices were based on the results of these datasets. Table 4.2 summarizes the validation datasets.

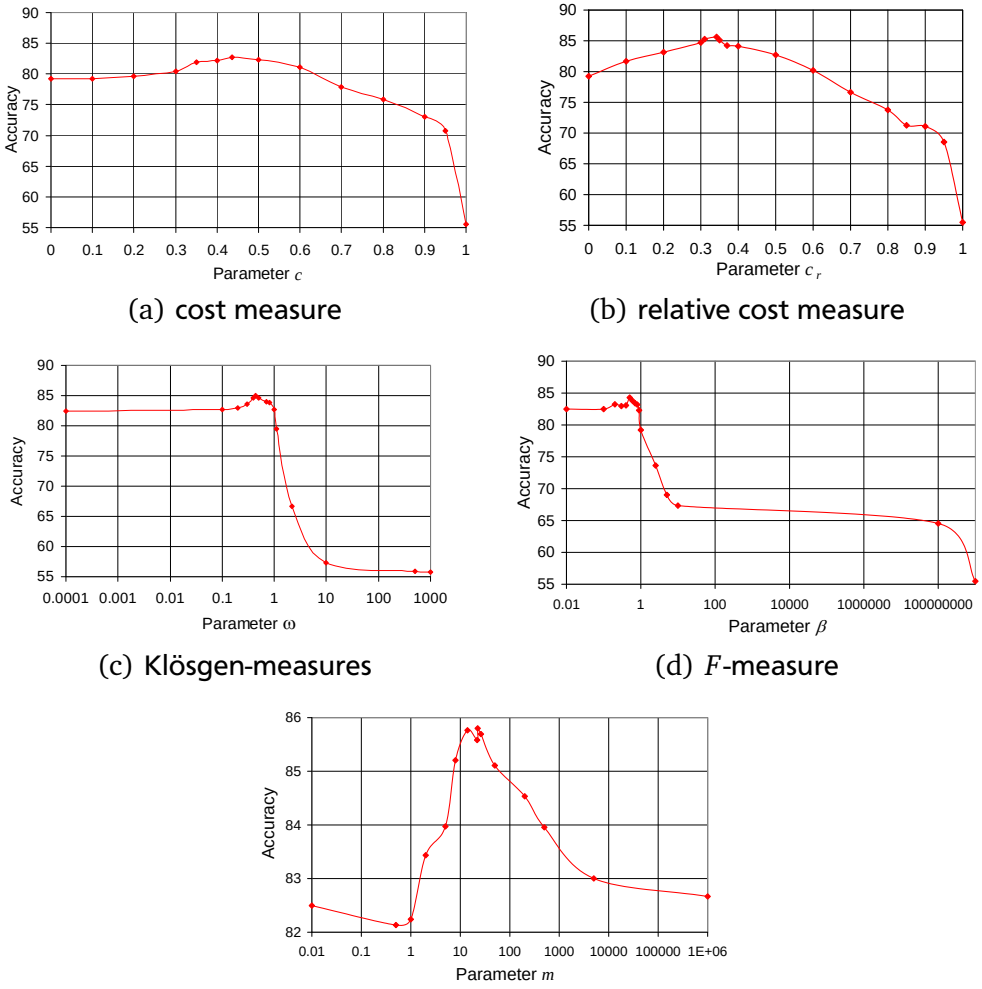
4.2 Optimization of Parametrized Heuristics

In this section, we will determine optimal parameters for the five parametrized rule evaluation metrics that we introduced in Section 2.6.3. For three of them, namely the F-measure, the *Klösgen measure*, and the m-estimate optimal parameters have already be determined [84]. Hence, we only give a brief review of this work here. For the remaining two, i.e., the two cost measures, we will analyze the average accuracy of the different heuristics under various parameter settings, identify optimal parameter settings, compare their coverage space isometrics, and evaluate their general validity.

4.2.1 Search Strategy

This section briefly describes the method for searching for the optimal parameter setting. The expectation was that for all heuristics, a plot of accuracy over the parameter value will roughly result in an inverse U-shape, i.e., there will be overfitting for small parameter values and over-generalization for large parameter values, with a region of optimality in between.

The parameter tuning framework uses a greedy search algorithm that continuously narrows down the region of interest. First, it tests a wide range of intuitively appealing parameter settings to get an idea of the general behavior of each of the



(a) cost measure (b) relative cost measure

(c) Klösgen-measures (d) F-measure

(e) m-estimate

Figure 4.1: Macro-averaged accuracy over parameter values for the five parametrized heuristics

five parametrized heuristics. The promising parameters were further narrowed down until we had a single point that represents a region of optimal performance.

The used procedure is greedy and not guaranteed to find a global optimum. In particular, there is a risk to miss the best parameter. If the curve is smooth, these assumptions are justified, but on real-world data we should not count on this.

The framework is based on a hill-climbing search where only one parameter is refined and investigated further on. Clearly, if a beam search is used where many parameters are tested in each time step, the chance would be greater to retrieve the best parameter instead of a local optimum. It was suggested to refine three

parameters simultaneously [84] and we also used this setting. The parallel search of more than one parameter can be seen as a beam search. The advantage is that local optima can be detected more easily.

However, also note that it is not a crucial issue to find an actual global optimum. If we can identify a region that is likely to contain the best parameter for a wide variety of datasets, this would already be sufficient for our purposes. We interpret the found values as good representatives for optimal regions.

4.2.2 Optimal Parameters for the Five Heuristics

Our first goal was to obtain optimal parameter settings for the five heuristics. As discussed above, the found values are not meant to be interpreted as global optima, but as representatives for regions of optimal performance. Figure 4.1 shows the obtained performance curves. Note that the parameters for the *Klösgen measure*, the F-measure, and the m-estimate are plotted on a logarithmic scale.

Cost Measures

Figures 4.1 (a) and (b) show the results for the two cost measures. Compared to the other measures, these curves are comparably smooth, and optimal values could be identified quite easily. Optimizing only the consistency (i.e., minimizing the number of negative examples without paying attention to the number of covered positives) has a performance of close to 80%. Not surprisingly, this can be improved considerably for increasing values of the parameters c and c_r. The best performing values were found at $c = 0.437$ (for the cost metric) and $c_r = 0.342$ (for the relative cost metric). Further increasing these values will decrease performance because of over-generalization. If the parameter approaches one, there is a steep descent because optimizing only the number of covered examples without regarding the covered negatives is, on its own, a bad strategy.

It is interesting to interpret the found values. For the cost metric, the optimal value $c = 0.437$ corresponds to a slope of $(1-c)/c \approx 1.3$, i.e., one false positive corresponds to approximately 1.3 true positives. Thus, consistency is favored over coverage. More interestingly, this bias towards consistency not only holds for absolute numbers but also for the true positive and false positives *rates*. Note that *wra*, which has been previously advocated as rule learning heuristic [162], corresponds to a value of $c_r = 0.5$, equally weighting false positive rate and true positives rate. Comparing this to the optimal region for this parameter, which is approximately

between 0.3 and 0.35, it can be clearly seen that it pays off to give a higher weight to the false positive rate, thereby favoring consistency over coverage.[1]

It is also interesting to compare the results of the absolute and relative cost measures: Although, as we have stated above, the two are equivalent in the sense that for each individual dataset, one can be transformed into each other by picking an appropriate cost factor, the *relative cost measure* has a clearly better peak performance exceeding 85%. Thus, it seems to be quite important to incorporate the class distribution $^{P}/_{(P+N)}$ into the evaluation metric. This is also confirmed by the results of the m-estimate and the Klösgen measure which are described below.

The *Klösgen measure*, the F-measure, and the m-estimate

Figure 4.1 (c), (d), and (e) display the results of the *Klösgen measure*, the F-measure, and the m-estimate. The curves of the *Klösgen measure* and the F-measure are quite similar. For large values of its parameter the accuracy of the F-measure decreases once again. This phenomenon is not noticeable for the *Klösgen measure*. For both measures the region $[0.1, 1]$ is the important one. For parameter values smaller than 0.1 the performance stays constant and for values bigger than one the found rules are too general and over-generalize. As all three heuristics implement the heuristic *precision* for parameter settings of zero the performance at the starting point of each of the three figures is the same. The average accuracy of *precision* is 82.36 which is better than minimizing the number of negative examples on its own.

The m-estimate has a lot more fluctuations and in fact two maxima. These two best parameter settings are close in terms of accuracy. The tuning process for the m-estimate was the hardest among all heuristics because the first iteration of the search procedure did not exhibit any clear tendencies. The m-estimate trades off between *precision* and *wra*. The latter, on its own, has a much better performance than simply optimizing the number of covered positive examples[2]. Clearly, this results in more effort to find the best setting. However, the optimal setting of $m = 22.466$ reached the best average accuracy of 85.87%. For the *Klösgen measure* a setting of $\omega = 0.4323$ with an accuracy of almost 85% was optimal and for the F-measure $\beta = 0.5$ had the best accuracy of 84.14%.

4.2.3 Experimental Results of the Tuned Heuristics

In this section, we compare the parameters which have been found for the five heuristics (cf. also Table 4.3). Then we show experiments to make sure that our

[1] Interestingly, the optimal value of $c = 0.342$ corresponds almost exactly to the micro-averaged default accuracy of the largest class (for both tuning and validation datasets). We are still investigating whether this is coincidental or not.

[2] The heuristic *recall*, that maximizes the true positive rate, has a performance of about 55%

Table 4.3: Results of the optimal parameter settings on the 27 tuning datasets

heuristic	average accuracy		average rank	average # conditions
	macro	micro		
m-estimate ($m = 22.466$)	85.87 (1)	93.87 (1)	4.54 (1)	36.85 (4)
relative cost ($c_r = 0.342$)	85.61 (2)	92.50 (6)	5.54 (4)	26.11 (3)
Klösgen ($\omega = 0.4323$)	84.82 (3)	93.62 (3)	5.28 (3)	48.26 (8)
JRip	84.45 (4)	93.80 (2)	5.12 (2)	16.93 (2)
F-measure ($\beta = 0.5$)	84.14 (5)	92.94 (5)	5.72 (5)	41.78 (6)
JRip-P	83.88 (6)	93.55 (4)	6.28 (6)	45.52 (7)
correlation	83.68 (7)	92.39 (7)	7.17 (7)	37.48 (5)
wra	82.87 (8)	90.43 (12)	7.80 (10)	14.22 (1)
cost measure ($c = 0.437$)	82.60 (9)	91.09 (11)	7.30 (8)	106.30 (12)
precision	82.36 (10)	92.21 (9)	7.80 (10)	101.63 (11)
Laplace	82.28 (11)	92.26 (8)	7.31 (9)	91.81 (10)
accuracy	82.24 (12)	91.31 (10)	8.11 (12)	85.93 (9)

results are not only due to overfitting of the 27 tuning datasets. We will then also describe experiments in which the tuned heuristics are used in two other rule learning algorithms, which have not been implemented by us, namely different versions of CN2 and RIPPER. Note that we used the original versions of the algorithms because they were not implemented in the SECO-Framework at this point in time.

Results on the 27 Tuning Datasets

Table 4.3 shows the results of the different heuristics on the 27 datasets, on which the parameters were tuned. We show micro- and macro-averaged accuracy, the average rank of the method on the datasets, and the average number of conditions of the learned rule sets. The numbers in brackets indicate the ranking of the methods according to each method. A comparison to JRIP (the *Weka*-implementation of RIPPER [22]) is also given in which the algorithm is used with and without pruning (-P). The table is sorted according to macro-averaged accuracy.

Consistent with this metric, the m-estimate and the *relative cost measure* clearly outperformed the other parametrized heuristics, as well as the standard heuristics, which we have also briefly described in Section 2.6.3. Interestingly, the *relative cost measure* performs much worse with respect to micro-averaged accuracy, indicating that it performs rather well on small datasets, but worse on larger datasets. These two heuristics also outperform JRIP on these datasets.

Table 4.4: Results of the optimal parameter settings on the 30 validation datasets

heuristic	average accuracy		average rank	average # conditions
	macro	micro		
JRip	78.98 (1)	82.42 (1)	4.72 (1)	12.20 (2)
relative cost ($c_r = 0.342$)	78.87 (2)	81.80 (3)	5.28 (3)	25.30 (3)
m-estimate ($m = 22.466$)	78.67 (3)	81.72 (4)	4.88 (2)	46.33 (4)
JRip-P	78.54 (4)	82.04 (2)	5.38 (4)	49.80 (6)
Klösgen ($\omega = 0.4323$)	78.46 (5)	81.33 (6)	5.67 (6)	61.83 (8)
F-measure ($\beta = 0.5$)	78.12 (6)	81.52 (5)	5.43 (5)	51.57 (7)
correlation	77.55 (7)	80.91 (7)	7.23 (8)	47.33 (5)
Laplace	76.87 (8)	79.76 (8)	7.08 (7)	117.00 (10)
precision	76.22 (9)	79.53 (9)	7.83 (10)	128.37 (12)
cost measure ($c = 0.437$)	76.11 (10)	78.93 (11)	8.15 (11)	122.87 (11)
wra	75.82 (11)	79.35 (10)	7.82 (9)	12.00 (1)
accuracy	75.65 (12)	78.47 (12)	8.52 (12)	99.13 (9)

Table 4.5: Spearman rank correlation between rankings of Table 4.3 and of Table 4.4

average accuracy		average	
macro	micro	rank	# conditions
0.85315	0.92308	0.88112	0.98601

Interestingly the cost metric performed rather bad. We think that this is due to the fact that this is the only parametrized heuristic that does not include information about the class distribution into its evaluation function. The m-estimate, the *Klösgen measure*, and the *relative cost metric* directly include the a priori probability of the positive class ($P/(P+N)$), whereas the F-measure only normalizes the positive examples. The results from our meta-learning experiments (Section 4.3) will support this hypothesis.

In terms of theory size, *wra* is the clear winner, with a predictive performance that exceeds the one of most other standard heuristics. This confirms the results of [162]. However, there is a large gap to the performance of JRIP and the parametrized heuristics. This indicates that, while *precision* and *Laplace* obviously overfit the data, *wra* has a tendency to over-generalize.

Obviously, the good results of the parametrized heuristics must be put into perspective because the parameters of the heuristics were optimized to perform

Table 4.6: Win/Loss/Tie Statistics and the p-values on the 30 validation datasets

Win/Loss/Tie p-Value	*precision*	*Laplace*	*accuracy*	*wra*	*correlation*	Sum
cost measure	12/17/1 *0.458*	11/17/2 *0.345*	13/16/1 *0.711*	15/14/1 *1.000*	13/14/3 *1.000*	64/78/8
relative cost measure	18/9/3 *0.122*	18/8/4 *0.0755*	23/7/0 *0.00522*	20/6/4 *0.00936*	19/9/2 *0.0872*	98/39/13
m-estimate	24/6/0 *0.00143*	20/9/1 *0.0614*	19/10/1 *0.136*	19/10/1 *0.136*	20/6/4 *0.00936*	102/41/7
Klösgen measure	22/8/0 *0.161*	18/10/2 *0.185*	23/7/0 *0.00522*	19/10/1 *0.136*	18/8/4 *0.0755*	100/43/7
F-measure	21/6/3 *0.00592*	18/11/1 *0.265*	24/4/2 *0.00018*	21/9/0 *0.0428*	17/9/4 *0.169*	101/39/10
Sum	97/46/7	85/55/10	102/44/4	94/49/7	87/46/17	

well on this subset of datasets (they were, however, not optimized on individual datasets). Thus, in order to get a fair comparison, it seems necessary to evaluate the methods on independent datasets, which were not used for tuning the parameters.

Validity of the Results on 30 Validation Datasets

In order to make sure that our results are not only due to overfitting of the 27 tuning datasets, we also evaluated the found parameter values on 30 new validation datasets. The results are summarized in Table 4.4. The numbers in brackets describe the rank of each heuristic according to the measure of the respective column. Again, JRIP with and without pruning is also included for comparison. Similarly as Table 4.3, it is also sorted by the macro-averaged accuracy.

Qualitatively, we can see that the relative performance of the heuristics in comparison to each other, and in comparison to the standard heuristics does not change much. The only obvious difference is the considerably better performance of JRIP, which indicates that some amount of overfitting has happened in the optimization phase. However, the performance of the best metrics is still comparable to the performance of JRIP, although the latter achieves this performance with much smaller rule sizes.

Table 4.5 shows the Spearman rank correlation coefficients (cf. Section 2.7.5) between the ranking of the heuristics on the tuning datasets and on the validation datasets. For all four measurements, we observe a correlation > 0.85, which makes us confident that the found optimal parameters are not overfitting the tuning datasets, but will also work well on new datasets.

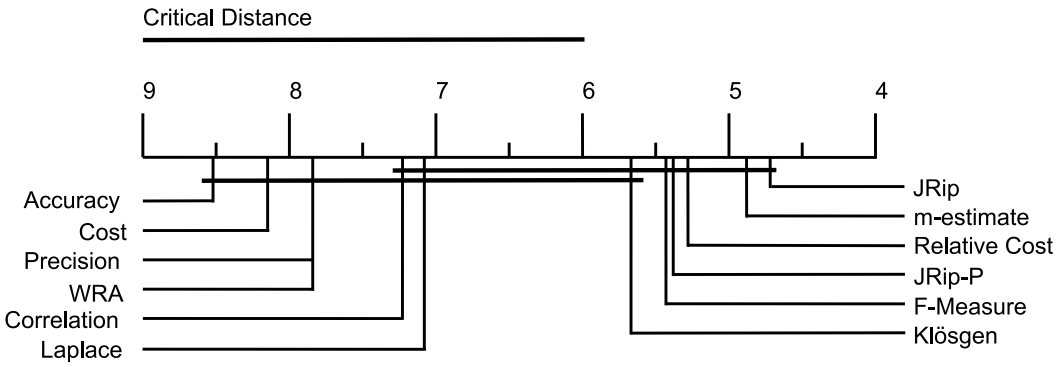


Figure 4.2: Comparison of all heuristics against each other with the Nemenyi test. Groups of heuristics that are not significantly different (at $p = 0.05$) are connected.

Table 4.6 gives a more fine-grained view on the performances of the optimized heuristics versus the standard heuristics on the 30 validation datasets. It shows for each pair of optimized and standard heuristic the number of wins, losses, and ties for the optimized heuristic (for macro-averaged accuracy). Below these three values, we show the p-value for a sign test with these values (cf. Section 2.7.5). The last column shows the sum of the values of the previous columns, i.e., they show how often the heuristic in this row has outperformed any of the heuristics in the columns. The row sums in the last row can be interpreted accordingly.

Again, we can see that, with the exception of the *cost measure*, all optimized heuristics outperform the standard heuristics on the majority of the datasets. There is not a single case where a standard heuristic has more wins than an optimized heuristic. In fact, each optimized heuristics has at least 17 wins and not more than ten losses. In many cases, the margin is much larger, and many of the differences are highly significant, even with the crude sign test.

Finally, Figure 4.2 displays a comparison of the ten heuristics and the two versions of JRip done with the Nemenyi test as suggested by [32] (cf. Section 2.7.5). The results from above are verified, which means that only the *Klösgen measures* and the *cost measure* are not significantly better than *accuracy*, *precision* and *wra*. All other heuristics including JRip outperform these heuristics significantly. Furthermore, even though *correlation* and *Laplace* are not significantly worse, we notice a large gap between all standard heuristics and the tuned ones (except the *cost measure*).

Table 4.7: Macro-averaged accuracy of the five heuristics when used in JRip and in CN2

Heuristic	JRip	JRip-P	CN2-u	CN2-dl	SimpleSeCo
Default Heuristic	79.19	79.01	78.06	78.44	—
m-estimate	78.64	78.36	77.86	78.17	79.22
Klösgen	78.04	78.10	78.42	78.70	78.71
F-measure	78.54	77.13	77.62	77.98	78.70
Relative Cost	77.71	78.03	77.53	77.80	79.09
Cost	74.57	70.54	72.97	75.75	76.89

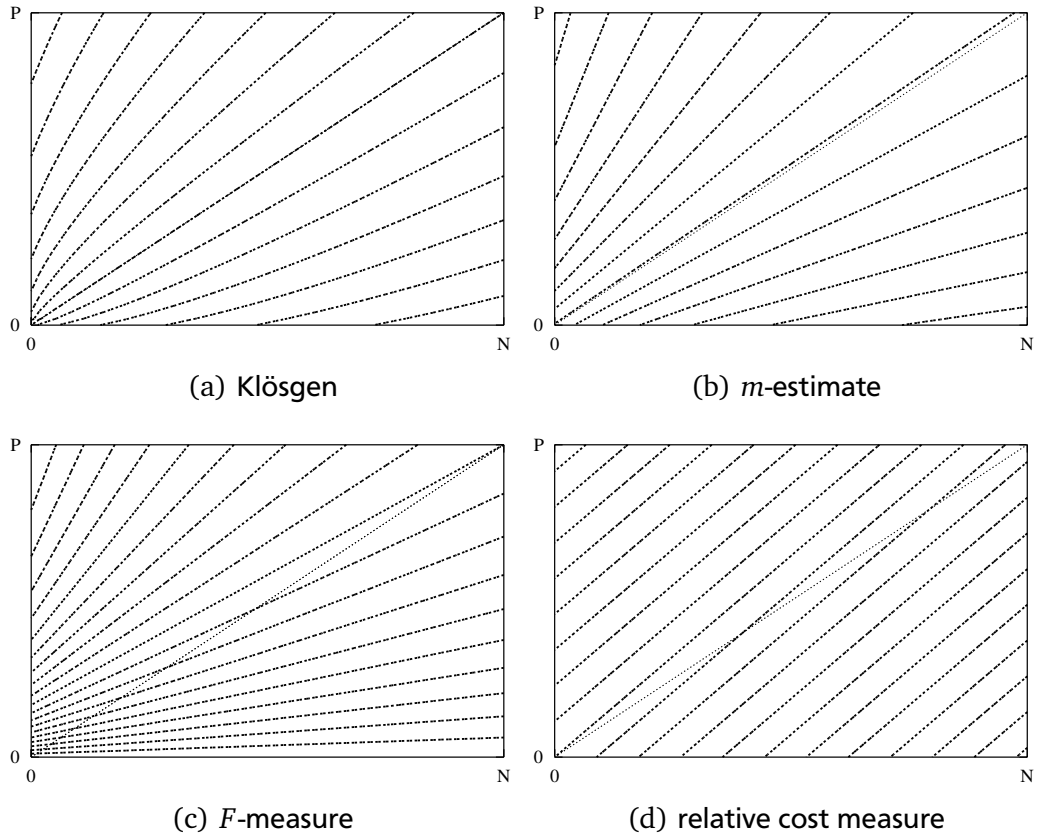
Validity of the Results in Comparison to other Algorithms

We also implemented the heuristics in the original implementation of CN2[3] and JRip, the *Weka*-implementation of Ripper. For a detailed discussion of Ripper see Section 3.5.3. Table 4.7 displays the results for evaluating these algorithms on the 30 validation datasets. Two datasets were left out because they contain missing class values that cannot be handled by CN2. Therefore, all results displayed in Table 4.7 are calculated on the remaining 28 datasets. JRip was used with and without pruning (-P). The other parameters were left at default values. In particular, CN2 was run in ordered (-dl) and unordered mode (-u) and with the default beam width of five.

Table 4.7 summarizes the results of these experiments. With respect to the relative order of the parametrized metrics, they essentially confirm our previous results: the m-estimate, the *relative cost measure*, the *Klösgen measure* and the F-measure are essentially indistinguishable (with a slight over-all advantage for the m-estimate), with the *cost measure* clearly lagging behind.

Compared to the original implementations, however, the results are not entirely as expected. First, we can note that JRip's *gain*-heuristic seems to outperform our heuristics. This is not implausible, because we have already observed above that heuristics that take the prior class distribution into account outperform heuristics that do not. *Gain*-heuristics can be interpreted as normalizing the example distribution so that the distribution of the covered examples of the previous rule is used as a prior distribution for finding the next literal [59]. While, for reasons outlined in Section 3.5.4, it is beyond the scope of this thesis, the question whether *gain*-heuristics are generally preferable to *value*-heuristics certainly deserves further investigation. We are currently working on this.

[3] Available from `http://www.cs.utexas.edu/users/pclark/software/` (visited 2012-04-24).



(a) Klösgen (b) *m*-estimate

(c) *F*-measure (d) relative cost measure

Figure 4.3: Isometrics of the best parameter settings

More surprising to us, however, was that our metrics did not improve CN2's default heuristic (*Laplace*) in terms of predictive accuracy although they did learn simpler rules in many cases. As we have discussed in Section 3.5.4, we consider the differences between our SIMPLESECO implementation and CN2's original implementation to be only minor, the only major difference being the strategy for handling multiple classes. Nevertheless, the improvements over *Laplace*, which we have observed in our implementation, do not seem to carry over to this implementation. Apparently, the differences between the algorithms are larger than we had expected, which is also witnessed by the large difference in predictive accuracy between CN2 and SIMPLESECO.

In general, while the learned values are certainly reasonable for other algorithms, our measures performed the best inside our own algorithm. Our SIMPLESECO implementation outperforms both CN2 and JRIP when used with our learned metrics. So, while the good performance seems to carry over to new datasets, the metrics seem to capture some aspects that are specific to the algo-

rithm used. We will discuss this issue in a bit more detail in Chapter 9. In some sense, these results correspond to the results of Chapter 5, where we will show that different heuristics may exhibit very different behaviors when used with different beam widths.[4]

4.2.4 Interpretation of the Learned Heuristics

Figure 4.3 shows the isometrics of the best parameter settings of the m-estimate, the F-measure, the Klösgen-measure, and the *relative cost measure*. It is interesting to compare the implemented preference structures. The *Klösgen measure* and the m-estimate appear to implement quite similar behavior. Their isometrics have almost the same shape, except that those of the *Klösgen measure* are slightly non-linear. The F-measure is also quite similar in the upper left region (high coverage and high consistency), but differs slightly in the low coverage regions, where it is necessarily parallel to the N-axis. The isometrics for the *relative cost measure* are confined to parallel lines. The slope of these isometrics seem to form an average: In high coverage and high consistency regions the slope is less steep than in the other heuristics, while in low coverage and low consistency regions it is considerably steeper. In any case, the slope is steeper than the diagonal, i.e., it is obvious that this heuristic gives a higher weight to consistency than to coverage.

4.3 Meta-Learning of Rule Learning Heuristics

While the previous section has focused on determining optimal parameters for a given functional form, we will now try to learn a function $h(p,n)$ from scratch. Thus, in this part of the chapter we do not optimize parameters any more but we try to take a different route. First it has to be defined how a function without a predefined form can be learned. In the following, we will therefore frame this problem as a meta-learning task, in which we try to predict the "true" performance of a rule on the test set.

Note that the work reported here was previously published ([85] and [88]). The framework to generate the meta data was suggested earlier ([57] and [58]). Albeit we re-used most of their design to generate such meta data, the focus in these works was different as the main goal there was to obtain predictions for out-of-sample precision, which indeed is also a by-product of our work, but not the main goal.

[4] Note that the results described here are also from different beam widths (CN2 has a default beam width of five, while SimpleSeCo uses hill-climbing). We had also tried to run CN2 with hill-climbing (beam size one), and the results were qualitatively similar. However, we are not sure whether hill-climbing works correctly in CN2. In particular, we noticed that the results were considerably worse because numerical attributes are practically ignored in that mode. Thus, we decided to omit these results.

4.3.1 Meta-Learning Scenario

The key issue for our work is how to define the meta-learning problem. It is helpful to view the rule learning process as a reinforcement learning problem [158]: Each (incomplete) rule is a state, and all possible refinements (e.g., all possible conditions that can be added to the rule) are the actions. The rule learning agent repeatedly has to pick one of the possible refinements according to their expected utility until it has completely learned the rule. Then, the learner receives a reinforcement signal (e.g., the estimated accuracy of the learned rule), which can then be used to adjust the utility function. After a (presumably large) number of learning episodes, the utility function should converge to a heuristic that evaluates a candidate rule with the quality of the *best* rule that can be obtained by refining the candidate rule.

However, for practical purposes this scenario appears to be too complex. A reinforcement learning approach on this problem was tried before [13], but with disappointing results. For this reason, we tried another, conceptually simpler approach, which tries to learn the same function in a supervised fashion: Each rule is evaluated on a separate test set, in order to get an estimate of its true performance. As a target value, we can either directly use the candidate rule's performance (*immediate reward*), or we can use the performance of its best refinement (*delayed reward*). We evaluated both approaches.

Meta Data Generation

The setup described in this section is related to the setup presented in the paper [57]. Albeit we generate meta data in a similar manner, the goal of the research presented here is different.

We have noted above, that heuristics typically depend on the number of true and false positives, and on the total number of positive and negative examples. However, most heuristics model non-linear dependencies between these values. In order to make the task for the learner easier, we will not only characterize a rule by the values p, n, P, and N, but in addition also use the following parameters as input for the meta-learning phase:

- $tpr = \frac{p}{P}$, the true positive rate of the rule
- $fpr = \frac{n}{N}$, the false positive rate of the rule
- $Prior = \frac{P}{P+N}$, the a priori distribution of positive and negative examples
- $prec = \frac{p}{p+n}$, the fraction of positive examples covered by the rule

Algorithm 4.1 GENERATEMETADATA*(TrainSet,TestSet)*, adapted from [57]

```
# loop until all positive examples are covered
while POSITIVE(TrainSet) ≠ ∅ do
    # find the best rule
    Rule ← FINDBESTRULE(TrainSet)
    # stop if it doesn't cover more positives than negatives
    if |COVERED(Rule,POSITIVE(Examples))|
        ≤ |COVERED(Rule,NEGATIVE(Examples))| then
        break
    # loop through all predecessors
    Pred ← Rule
    repeat
        # record the training and test coverage
        p ← |COVERED(Pred,POSITIVE(TrainSet))|
        n ← |COVERED(Pred,NEGATIVE(TrainSet))|
        P ← |COVERED(Pred,TOTALPOSTIVE(TrainSet))|
        N ← |COVERED(Pred,TOTALNEGATIVE(TrainSet))|
        l ← LENGTH(Rule)
        p̂ ← |COVERED(Pred,POSITIVE(TestSet))|
        n̂ ← |COVERED(Pred,NEGATIVE(TestSet))|
        # print out meta training instance
        print P,N,P/(P+N),p,n,p/P,n/N,p/(p+n),l
        # print out meta target information
        print p̂,n̂,p̂/(p̂+n̂)
        Pred ← REMOVELASTCONDITION(Pred)
    until Pred = null
    # remove covered training and test examples
    TrainSet ← TrainSet \ COVERED(Rule,TrainSet)
    TestSet ← TestSet \ COVERED(Rule,TestSet)
```

Thus, we characterize a rule r by an 8-tuple

$$h(r) \leftarrow h(P,N,Prior,p,n,tpr,fpr,prec)$$

In Section 4.3.2, we will also consider the *rule length* l as an additional input.

As explained above, we try to model the relation of the rule's statistics measured on the training set and its "true" performance, which is estimated on an independent test set. Thus, a meta training instance consists of the abovementioned characteristics for the corresponding rule. The training signal is the performance of the rule on the test set. For assessing the performance of the rule, we typically use its out-of-sample precision, but, again, we have also experimented with other choices.

As we want to guide the entire rule learning process, we need to record this information not only for final rules – those that would be used in the final theory – but also for all their predecessors. Therefore all candidate rules which are created during the refinement process are included in the meta data as well. Algorithm 4.1 shows this process in detail.

It should be noted, that we ignored all rules that do not cover any instance on the test data. Our reasons for this were that on the one hand we did not have any training information for this rule (the test precision that we try to model is undefined for these rules), and that on the other hand such rules do not do any harm (they won't have an impact on test set accuracy as they do not classify any example).

To ensure that we obtain a set of rules with varying characteristics, the following parameters were modified:

Datasets: To create preferably different rules, we trained the models on the 27 tuning datasets described in Section 4.1.1.

5×2 cross-validation: Our primary interest was to obtain a lot of rules which characterize the connection between training set statistics and the test set precision. For this reason, we performed five iterations of a 2–fold stratified cross-validation (with different random seed). Hence, the training and test sets have equal size so that we do not have to account for statistical variance. Note that statistics of all folds were collected as we want to have a huge number of different rules.

Classes: A 1-vs-all class binarization (cf. Section 2.3.1) was performed for each dataset and each fold. Thus, each class once was treated as the positive one whereas the union of all other classes were treated as the set of negative examples, respectively. Then, rules were learned on all the resulting two-class datasets.

Heuristics: The heuristics were also varied. The composite heuristics described in Section 2.6.2 were used except the *odds ratio*. As mentioned there, these heuristics represent a great variety of different learning biases, some overfitting, some underfitting. Consequently, the rule learner was run once for each heuristic.

In total, our meta dataset contains $87,380$ examples.

Meta-Learning Algorithms

We used two different methods for learning functions on the meta data. First, we applied a simple linear regression using the Akaike criterion [2] for model

selection. A key advantage of this method is that we obtain a simple, easily comprehensible form of the learned heuristic function. Note that the learned function is nevertheless non-linear in the basic dimensions p and n because of the abovementioned non-linear terms that are used as basic features.

Nevertheless, the type of functions that can be learned with linear regression is quite restricted. In order to be able to address a wider class of functions, we also tried a multilayer perceptron (MLP) with back propagation algorithm and sigmoid nodes. We used various sizes of the hidden layer (one, five, and ten), and trained for one epoch (i.e., we went through the training data once). We have also tried to train the networks with a larger number of epochs, but the results no longer improved. We selected the abovementioned numbers of nodes in the hidden layer because we wanted to have a very simple model that can be trained very fast. Nevertheless, the functions that can be learned with one node in the hidden layer are restricted to linear ones. For this reason we also tried five and ten nodes to make sure that we have not selected a model that is too simple to perform reasonable on the meta-learning task.

Both algorithms are provided by *Weka* [177] and were initialized with standard parameters. We had also tried a support vector machine for Regression. As the Regression SVMs of *Weka* (SVMREG and SMOREG) could not be trained on the meta data because the datasets were too big, we resorted to the use of LIBSVM [16]. However, the results were comparable to those of the neural network, which in turn was worse than the linear regression. Hence we do not include experimental results of the SVM.

4.3.2 Experimental Results

In this section, we discuss our experimental results with the meta-learning approach. We will start with a straight-forward baseline experiment that uses the meta data as described in Section 4.3.1, and then try to experimentally answer the questions whether inclusion of the rule length improves the result, whether learning in the delayed reward scenario is better than learning from immediate rewards, and whether other heuristic functions perform better than (predicted) precision.

Baseline Experiment

In a first experiment, we wanted to see how accurately we can predict the out-of-sample precision of a rule using the meta data as described in Section 4.3.1. We trained a linear regression model and a neural network on the eight measurements that we use for characterizing a rule using the precision values measured on the test sets as a target function. Table 4.8 displays results for the linear regression and three neural networks with different numbers of nodes in the hidden layer

Table 4.8: Accuracies of SIMPLESECO for several meta-learned heuristics

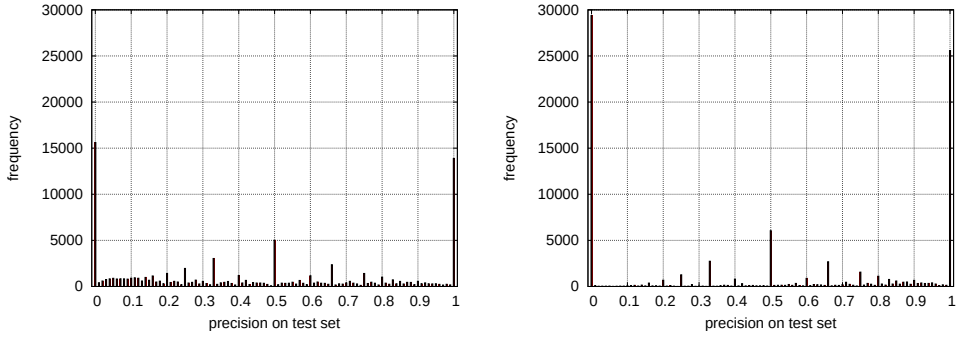
heuristic	*mae*	average accuracy		average
		macro	micro	# conditions
Linear Regression	0.22	77.43%	80.19%	117.6
MLP (1 node)	0.28	77.81%	81.43%	121.3
MLP (5 nodes)	0.27	77.37%	80.45%	1,085.8
MLP (10 nodes)	0.27	77.53%	80.27%	112.7

on the same 30 validation datasets that were used before (cf. Section 4.1.1). The macro- and micro-averaged accuracy is given. Additionally, the *mean absolute error (mae)* of a cross-validation on the meta-learned training set is displayed and the average number of conditions is shown. The performances of the three algorithms are quite comparable, with the possible exception of the neural network with five nodes in the hidden layer. The heuristic learned by this network induced very large theories (over 1,000 conditions on average), and also had a somewhat worse performance in predictive accuracy. In general, the experiments seem to show that a linear combination of the available features is sufficient, and that more nodes in the hidden layer will not yield performance improvements. Note, however, that the mean absolute error displayed in Table 4.8 measures the error made by the regression model on unseen data. A low mean absolute error on a dataset does not implicate that the function works well as a heuristic. For example, a systematic, large over-estimation of the heuristic value may result in a higher absolute error than a small random fluctuation around the correct value, but may produce a much better performance if the correct ordering of values is preserved. This becomes evident as the linear regression has a lower mean absolute error than the neural network with one node but works worse as heuristic.

If we compare these results to those of Table 4.4 (column macro-averaged accuracy), we can see that the learned heuristics outperform all standard heuristics with the exception of *correlation*. However, they do not quite reach the performance of the optimized parametrized heuristics.

Significance of Rule Length

Some rule learning algorithms include the length of the learned rule into their evaluation function. For example, the ILP algorithm PROGOL [122] uses $p - n - l$ as a search heuristic for a best-first search. The first part, $p - n$, directly optimizes *accuracy* (for a fixed dataset, i.e., where the total number of positive (P) and negative (N) examples are fixed), and the length of the rule is used to add an additional bias for simpler rules. However, as longer rules typically cover fewer



(a) immediate reward (b) delayed reward

Figure 4.4: Histogram of the frequency of observed precision values for the two scenarios

examples, penalizing the length of a rule may also be considered as another form of bias for high-coverage rules, which could also be expressed by maximizing p (or $p + n$). In any case, we also experimented with the rule length as an additional parameter. For both, linear regression and neural networks this did not lead to notable changes in the performance of the heuristics (e.g., for linear regression, the performance dropped by 0.03%).

Predicting the Value of the Final Rule

Rule learning heuristics typically evaluate the quality of the current, incomplete rule, and use this measure for greedily selecting the best candidate for further refinement. However, as discussed in Section 4.3.1, if we frame the learning problem as a search problem, a good heuristic should not evaluate a candidate rule with its discriminatory power, but with its potential to be refined into a good final rule. Such a utility function could be learned with a reinforcement learning algorithm, which will learn to predict in each step of the refinement process which refinement is most likely to lead to a good final rule. Unfortunately, it was pointed out that this approach does not work satisfactorily [13].

As an alternative, we applied a method which can be interpreted as an "offline" version of reinforcement learning. We simply assign each candidate rule the precision value of its final rule in one refinement process. As a consequence, in our approach all candidate rules of one refinement process have the same target value, namely the value of the rule that has eventually been selected. Because of the deletion of all final rules that do not cover any example on the test set, we decided to remove all predecessors of such rules as well. This seemed to be the best way to

Table 4.9: Accuracies of SIMPLESECO for two meta-learned heuristics (delayed rewards)

heuristic	*mae*	average accuracy		average # conditions
		macro	micro	
Linear Regression	0.33	77.95%	80.97%	95.63
Neural Network	0.35	78.37%	81.43%	53.97

handle the predecessors because we would not have a reasonable value to predict. Thus, the new meta dataset contains only 77,240 examples in total.

Figure 4.4 shows a histogram of the observed test-set precision values for the candidate rule (immediate reward) and for the final rule that has been learned when refining this candidate (delayed reward). Clearly, in the case of delayed rewards, the frequency of simple precision values as 0, 0.5, and 1 increases, because there are much more rules that only cover a few examples.

Table 4.9 shows the accuracies of two heuristics that were learned in this setting, the first one with a linear regression and the second one with a neural network with a single node in the hidden layer. Both macro- and micro-averaged accuracies are displayed, the average number of conditions is given, and as in Table 4.8 the *mae* is shown. In particular the neural network outperformed the original setting (cf. Table 4.8) and approaches the performance of the heuristics obtained by parameter optimization (Table 4.4).

Predicting Other Heuristic Functions

So far, we focused on directly predicting the out-of-sample precision of a rule, assuming that this would be a good heuristic for learning a rule set. However, this choice was somewhat arbitrary. Ideally, we would like to repeat this experiment with out-of-sample values for all common rule learning heuristics. In order to cut down the number of required experiments, we decided to directly predict the number of covered positive ($\hat{p}$) and negative ($\hat{n}$) examples. Then we can combine the predictions for these values with any standard heuristic h by computing $h(\hat{p}, \hat{n})$ instead of the conventional $h(p, n)$. Note that the heuristic h only gets the predicted coverages ($\hat{p}$ and $\hat{n}$) as new input, all other statistics (e.g., P,N) are still measured on the training set. This is feasible because we designed the experiments so that the training and test set are of equal size, i.e., the values predicted for $\hat{p}$ and $\hat{n}$ are predictions for the number of covered examples on an independent test set of the same size as the training set.

Table 4.10 compares the performance of various heuristics using the p and n values measured on the training set, and the $\hat{p}$ and $\hat{n}$ values predicted for the test

Table 4.10: Comparison of various heuristics with training-set coverages (p, n) and coverages predicted by the neural network $(\hat{p}, \hat{n})$

heuristic	args	average accuracy		average
		macro	micro	# conditions
accuracy	(p, n)	75.65%	78.47%	99.13
	$(\hat{p}, \hat{n})$	75.39%	78.62%	110.80
precision	(p, n)	76.22%	79.53%	128.37
	$(\hat{p}, \hat{n})$	76.53%	80.43%	30.00
wra	(p, n)	75.82%	79.35%	12.00
	$(\hat{p}, \hat{n})$	69.89%	75.23%	29.97
Laplace	(p, n)	76.87%	79.76%	117.00
	$(\hat{p}, \hat{n})$	76.80%	80.77%	246.80
correlation	(p, n)	77.55%	80.91%	47.33
	$(\hat{p}, \hat{n})$	58.09%	65.35%	40.40

set by a trained neural network. In general, the results are disappointing. For three of the five heuristics, no significant change could be observed, but for *wra* and *correlation*, the performance degrades substantially.

A surprising observation is the rather low complexity of the learned theories. For instance, the heuristic *precision* produces very simple theories when it is used with the out-of-sample predictions, and, by doing so, increases the predictive accuracy. Apparently, the use of the predicted values of $\hat{p}$ and $\hat{n}$ allows to prevent overfitting, because the predicted positive/negative coverages are never exactly zero and therefore the overfitting problem observed with *precision* does not occur any more. The *Laplace* heuristic shows a similar trend, but in this case the predictions result in more complex rules than the original ones.

In summary, it seems that the predictions of both the linear regression and the neural network are not good enough to yield true coverage values on the test set. A closer look at the predicted values reveals that on the one hand both regression methods predict negative coverages and that on the other hand for the region of low coverages (which is the important one) too optimistic values are predicted (for both the positive and the negative coverage). The acceptable performance is caused by a balancing of the two imprecise predictions (as observed with the two *precision*-like metrics) or rather by an induced bias which tries to omit the extreme values in the evaluations (which are responsible for overfitting).

Table 4.11: Coefficients of various functions learned by linear regression

	Baseline Experiment	Delayed Reward Scenario	Delayed Reward + Logarithmic Coverage	
accuracy	77.43%	77.59%		78.88%
P	0.0001	0.0	$\log(P+1)$	0.0709
N	0.0001	0.0002	$\log(N+1)$	−0.0255
$\frac{P}{P+N}$	0.7485	0.8772	$\frac{P}{P+N}$	0.0521
p	−0.0001	−0.0002	$\log(p+1)$	0.1139
n	−0.0009	0.0002	$\log(n+1)$	−0.0588
$\frac{p}{P}$	0.165	0.2103	$\frac{p}{P}$	0.1379
$\frac{n}{N}$	0.0	−0.297	$\frac{n}{N}$	−0.3673
$\frac{p}{p+n}$	0.3863	0.1367	$\frac{p}{p+n}$	−0.1032
$const.$	0.0267	0.2282	$const.$	0.427

4.3.3 Interpretation of the Learned Functions

In this section, we will try to interpret the learned functions by looking at the learned weights and by looking at their coverage space isometrics.

Coefficients of the Linear Regression

Table 4.11 shows the coefficients for three learned regression models. In the baseline experiment, three features had a notably high weight:

- the a priori class distribution of the examples in the training data,
- the precision of the rule, and
- the true positive rate.

These feature weights were significant with a p-value smaller than 2×10^{-16} computed with the *summary*-method of R[5]. Only the false positive rate was not statistically significant. At first it may be surprising that the false positive rate is insignificant, but its main role is to ensure consistency, which can – in the regions of interest – also be ensured with *precision*. Thus, if we only consider feature weights above 0.1 as important for the model, the learned heuristic linearly combines class distribution, coverage and consistency. Informally, we can also observe that, in

[5] `http://www.r-project.org/` (visited 2011-11-05)

line with our observations from Section 4.2, consistency receives a higher weight than coverage, although it is not entirely clear whether these values are directly comparable.

This can be more clearly seen from the coefficients learned in the delayed reward scenario, where the function was trained on the test set precision of the best refinement of the rule. The function is quite similar to the previous one, except that the consistency is now enforced through two factors:

- a high negative weight on the false positive rate and
- a positive weight on precision.

In this scenario the weight of the total positives was only significant at $p = 0.01$, all others were also significant with a p-value of at least 3.69×10^{-5}.

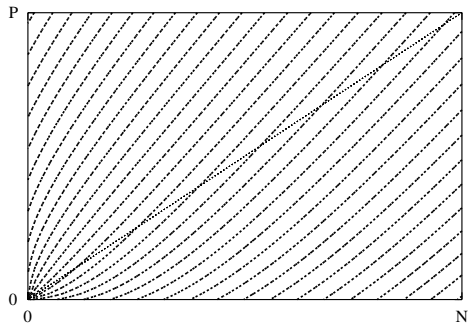
In both cases, the current coverage of a rule (p and n) and the total example counts of the data (P and N) have comparably low weights. This is not that surprising if one keeps in mind that the target value is in the range $[0, 1]$, while the absolute values for p and n are in a much higher range. We nevertheless included them because we believe that in particular for rules with low coverage, the absolute numbers are more important than their relative fractions. A rule that covers only a single example will typically be bad, irrespective of the size of the original dataset.

In the light of these results, we made two more experiments: In the first, we removed the four coverage values from the input, and learned another function from the remaining four features. This did not change the performance very much (77.20% macro-averaged accuracy).

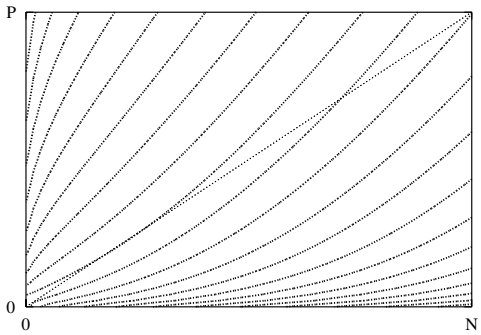
In a second experiment, we used the logarithmic values $\log(P+1)$, $\log(N+1)$, $\log(p+1)$, $\log(n+1)$ instead, with the idea that the importance of differences in coverage is proportional to the coverage. This considerably improved the results for linear regression. The last part of Table 4.11 shows the learned function. There are a few interesting differences to the previous functions:

- the logarithmic coverage values get a much higher weight than their absolute counterparts (all significant at $p < 2 \times 10^{-16}$),
- the prior class probability $P/(P+N)$ receives a much lower weight (still significant with $p = 0.0021$), and
- precision receives now a negative weight (also significant at $p < 2 \times 10^{-16}$), which is presumably counterbalanced by the much higher negative weight on the false positive rate.

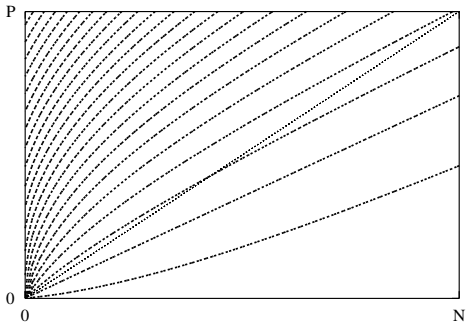
Note that in the remainder of the thesis whenever we refer to a heuristic called *Linear Regression* (h_{LR}) we mean the variant trained with delayed rewards on logarithmic features as shown in Table 4.11 in the rightmost column.



(a) Linear Regression



(b) Linear Regression with logarithmic features



(c) Neural Network

Figure 4.5: Isometrics of heuristics meta-learned with two linear regressions and a neural network in the delayed reward scenario

Isometrics of the Heuristics

To understand the behavior of the learned heuristics, we will again take a look at their isometrics in coverage space. Figure 4.5 shows isometrics of the heuristic learned in the experiment with delayed rewards (with and without the logarithmic features) in a coverage space with 60×48 examples (the sizes were chosen arbitrarily but are the same as for all other previous and subsequent isometrics). The upper left part of the figure displays the isometrics of the heuristic that was learned by linear regression on the dataset that used only the relative features. The upper right part shows the isometrics of the heuristic that was learned by linear regression with the logarithmic features. The bottom part shows the best-performing neural network (the one that uses only one node in the hidden layer).

Apparently, all functions learn somewhat different heuristics. Superficially, the isometrics of the linear regression heuristic (Figure 4.5 (a)) are quite similar to the

parallel lines of the *relative cost measure*, but, just as we observed in the experiments of Section 4.2 (cf. Figure 4.3 (d)), their slope is generally > 1, i.e., false positives are weighted more heavily than true positives. The isometrics for the neural net seems to employ a trade-off similar to those of the F-measure. The shift towards the N-axis is reminiscent of the F-measure (cf. Figure 2.12), which tries to correct the undesirable property of *precision* that all rules that cover no negative examples are evaluated equally, irrespective of the number of positive examples that they cover. Interestingly, in the upper left corner of the coverage space, the isometrics of the linear regression function with logarithmic features (Figure 4.5 (b)) have a quite similar appearance as the isometrics for the *Klösgen measure*. In the region around the diagonal, the isometrics of the linear regression are non-linear with a steeper slope. Beneath the diagonal, the isometrics approach those of the F-Measure with the exception that they do not start in the origin.

However, in all cases the isometrics have a non-linear shape, which bends them towards the N-axis when they approach the P-axis. Thus, in regions with high consistency, the bias that prefers consistency over coverage is even more emphasized. This also has a somewhat surprising effect, namely a small bias towards rules that cover a low number of positive examples (compared to regular *precision*). Intuitively, one would expect the opposite, namely that rules with low coverage are avoided because they are likely to be unreliable and noisy. This confirms our results for the *Klösgen measure*, where we could see that parameter values $\omega > 1$ encode a bias that avoids low coverage regions (cf., e.g., the graph for $\omega = 2$ in Figure 2.13), but that these values did not perform well empirically. In some sense, this may be interpreted as support for the well-known *small disjuncts problem*, first observed in [79], namely that rules with low coverage have a rather high error compared to rules with higher coverage, but that they also cannot be omitted without a loss in accuracy. As shown by Holte et al. [79], despite the high error rate these small disjuncts have, they are indeed necessary as otherwise it is even more likely that the examples previously covered by them are classified incorrectly.

4.4 Related Work

While there are several empirical comparisons of splitting heuristics for decision tree induction [115, 12], there are, somewhat surprisingly, relatively few works that empirically compare different rule learning heuristics. For example, in [106, 107] several heuristics for inductive logic programming were compared. Most works only perform a fairly limited comparison, which typically introduces a new heuristic and compares it to the heuristic used in an existing system. A typical example for work in this area is [162], where the performance of *wra* was compared to the performance of CN2's *Laplace*-heuristic. To our knowledge, our work reported in this chapter is the most exhaustive empirical work in this respect.

On the other hand, considerable progress has been made in the principal understanding of rule learning heuristics. As discussed in Section 2.5, Fürnkranz and Flach [60] have introduced coverage space isometrics as a means for visualizing rule evaluation metrics. Using this tool, they have derived several interesting results, such as that the m-estimate effectively trades off *precision* and *wra*. While their paper contributed to a better understanding of rule learning heuristics, the authors concluded that, in general, rule learning heuristics are not yet well understood.

There has also been considerable progress on analyzing rule evaluation metrics that are commonly used in descriptive induction tasks such as association rule discovery or subgroup discovery. Most notably, 21 rule learning heuristics were surveyed and compared according to a set of desirable properties [159]. In general, it was concluded that the choice of the right interestingness measure is application-dependent, but also situations were identified in which many measures are highly correlated with each other [159]. Bayardo, Jr. and Agrawal [6] analyzed several heuristics in support and confidence space, and showed that the optimal rules according to many criteria lie on the so-called support/confidence border, the set of rules that have maximum or minimum confidence for a given support level. Recently, it was showed that a group of so-called null-invariant measures (measures that are not influenced by the number of records that do not match the pattern) can be generalized into a single parametrized heuristic [181]. We plan to analyze this parametrized heuristic with the apparatus that we have used for our results in Section 4.2.

Naturally, there are some similarities between heuristics used for descriptive and for predictive tasks. For example, Lavrač, Flach, and Zupan [104] derived *wra* in an attempt to unify these two realms, or Fürnkranz and Flach [59] analyzed filtering and stopping heuristics and showed that FOIL's information gain search and MDL-based pruning has a quite similar effect as support and confidence thresholds that are commonly used in association rule discovery. Nevertheless, it is important to note that good heuristics for descriptive induction are not necessarily well-suited for predictive induction (*wra* is a good example). The key difference is that in the latter case one typically needs to learn an entire rule set, where lack of coverage in individual rules can be corrected by the entire ensemble of rules. Inconsistencies, on the other hand, cannot be corrected by the induction of additional rules (at least not in the case of concept learning). In this light, the result of this chapter, that good heuristics for predictive induction will favor consistency over coverage, appears to be reasonable.

Our results may also be viewed in the context of trying to correct overly optimistic training error estimates (resubstitution estimates). In particular, in some of our experiments, we try to directly predict the out-of-sample precision of a rule. This problem has been studied theoretically ([149] and [119]). In other works, it has been addressed empirically. For example, Vapnik et al. used empirical data to

measure the VC-Dimension of learning machines [167]. Previously, meta data also was created in a quite similar way, and it was tried to fit various functions to the data ([57] and [58]). But the focus there is the analysis of the obtained predictions for out-of-sample precision, which is not the key issue in our experiments.

4.5 Summary

In this chapter, on the one hand, optimal parameter settings for five parametrized heuristics were presented and, on the other hand, new heuristics were learned by a meta-learning approach. In particular, we have determined suitable default values for commonly used parametrized evaluation metrics. This is of considerable practical importance, as we showed that these new values outperformed conventional search heuristics and performed comparably to the RIPPER rule learning algorithm. On the other hand, however, we have also seen some indication that these values capture aspects of our algorithm that may not fully transfer to other rule learning algorithms. Nevertheless, for the simple rule learner the values proved to be a reasonable choice. The optimal choices may be influenced by more complex mechanism as, e.g., the optimization phase of RIPPER.

In the evaluation of these settings we found that it is beneficial to include the prior distribution of the dataset and to weight consistency more heavily than coverage. Interestingly, the meta-learned heuristics where no bias towards existing measures was imposed, also implement a quite similar preference structure as observed for the parametrized heuristics (cf. the isometrics given in Figure 4.3 and 4.5). For example, the coefficient of the prior distribution had a high weight in the meta-learning experiments, which confirms the importance noticed for those parametrized heuristics that include the prior distribution.

5 A Comparison of Search Algorithms for Heuristic Rule Learning

In this chapter we examine how different rule learning heuristics behave with increased search effort. Therefore, we implemented four different search strategies, i.e., hill-climbing, beam search, a true exhaustive search, and a bidirectional search. The aim of the experiments was to extend a previous work [134]. Here, for rule learning, a phenomenon called *oversearching* was observed, which essentially says that increased search effort will not only not improve the results but may even lead to a decrease in accuracy. However, this work was limited to the evaluation of only a single heuristic. A true exhaustive search was also not implemented there [134]. The key question is how the search heuristics we have used so far behave when the search algorithm is exchanged and whether they suffer from the problem of oversearching or not. Note that the aim is to utilize different search algorithms for refining a single rule. Conversely, one could also use an exhaustive search that optimizes the whole rule set which usually imposes too much computational effort to be done in practice (cf. [139]).

The work reported here is also related to the optimization phase of the RIPPER algorithm (cf. Section 3.5.3). One major reason for the good performance of this algorithm is its flexibility. On the one hand, the I-REP strategy used in RIPPER enables the algorithm to find theories that are adjusted to the actual dataset, albeit not overfitting it. On the other hand, the post-pruning also facilitates the adaptation to different situation that occur in different datasets. The reason for both advantages is that the algorithm is able to revert previous decisions that may have led to a suboptimal rule set by allowing to prune the rule(s). A key question is whether these advantages can also be achieved by allowing the search algorithm itself to remove conditions of a rule during the search. In this case, a suboptimal choice in an early refinement step of the algorithm may be reversed by deleting the condition that has led to the bad rule. For this reason, we have implemented a bidirectional search inside the SECO-Framework and have evaluated it against RIPPER. Additionally, a comparison of the bidirectional approach with the beam search and the true exhaustive search is given.

This chapter extends [87] and [95]. It starts with a brief sketch about the implemented algorithm (Section 5.1). Then, the three unidirectional search strategies are introduced in Section 5.2. In the following section, the experimental setup for this chapter is recapped. In Section 5.4 the results of the experiments with the unidirectional searches are presented. The next section (Section 5.5) follows up with

a description of bidirectional rule learning including a new experimental setup as well as empirical results and a summary. Related work about search algorithms for rule learners is given in Section 5.6 and Section 5.7 consists of a summary.

5.1 Implementation of the Algorithm

For our experiments, the SIMPLESECO learner was used (cf. Section 3.5.4). The process of searching for single rules closely follows the general procedure described in Algorithm 3.2. The search algorithms hill-climbing and beam search are used as provided by the SECO-Framework but the efficient exhaustive search does not match the interface and therefore is implemented in an extra function. At the moment, when the efficient exhaustive search is used, the algorithm can only handle nominal attributes, we are currently working on an implementation that can also use numerical attributes.

For the experiments, the heuristics are exchanged and the type of search is implemented in different ways. Note that the tie-break mechanism used in the SECO-Framework is especially important for the heuristic *precision* (cf. Section 5.4.1).

The SIMPLESECO does not use a special pruning or optimization phase. Nevertheless the Forward Pruning (cf. Section 3.2.1) greatly improves the runtime of the algorithm when the search effort increases.

5.2 Search Strategies

Most global rule learning algorithms employ a hill-climbing strategy [22, 105], which starts with a rule that covers all examples (the empty rule) and evaluates all possible conditions as candidate extensions of the rule. The best extension according to some heuristic measure is selected and added to the rule. This is repeated until no condition improves the heuristic measure (e.g., when no negative example is covered). This strategy is implemented in the SECO-Framework in the TOPDOWNREFINER.

On the other hand, association rule discovery typically is performed via an exhaustive search that discovers all patterns that satisfy a given set of constraints. In the simplest case, this can be used for rule induction by repeatedly generating all possible rules and selecting the one that is the best extension to the current rule set.

Hill-climbing is fast, but may get stuck in local optima, whereas exhaustive search will find a global optimum, but is slow. To trade off between these two methods, beam search can be used, which has a parameter b to determine how many rules are refined in a single steps. A beam search with $b = 1$ is equivalent to hill-climbing, whereas $b = \infty$ corresponds to an (inefficient) exhaustive search.

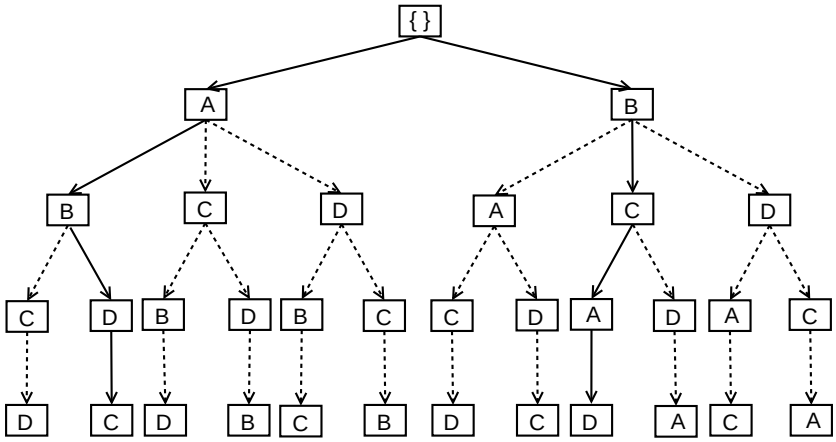


Figure 5.1: Hill-Climbing and Beam Search

5.2.1 Hill-Climbing

The advantage of hill-climbing clearly lies in its efficiency concerning both memory and time issues. The disadvantage is that the search can be stuck in a local optimum without finding the global optimum. Note that for any refinement process there is exactly one path through the search space. In the first step of the FINDBESTRULE procedure all attribute-value pairs are generated and evaluated. Until then the used conditions are stored so that they are not generated twice.

Figure 5.1 (left subtree) shows an example for a refinement path through the search space (depicted with solid arrows). Assume that only the attributes `A` and `B` can be selected in the first test (attribute values are omitted). Hill-climbing now adds the attribute with the highest evaluation value (in this case `A`). In the second step there are only three possible Tests (`B`, `C` and `D`). Attribute `B` is selected, `D` and `C` are remaining, hill-climbing selects `D` and then only `C` remains. Note that now the refinement process is terminated because the set of possible attributes has been exhausted. The rule that is returned is not necessarily the last one generated ($A \wedge D \wedge B \wedge C \rightarrow CLASS$) since other rules along the refinement path could have a higher evaluation value and would be returned instead.

5.2.2 Beam Search

Beam search is useful for avoiding situations where a locally optimal choice is globally suboptimal. The idea is simply to refine b rules simultaneously. The beam size b may be viewed as a parameter that allows to trade off between hill-climbing and exhaustive search: $b = 1$ corresponds to hill-climbing, while for $b \rightarrow \infty$, beam

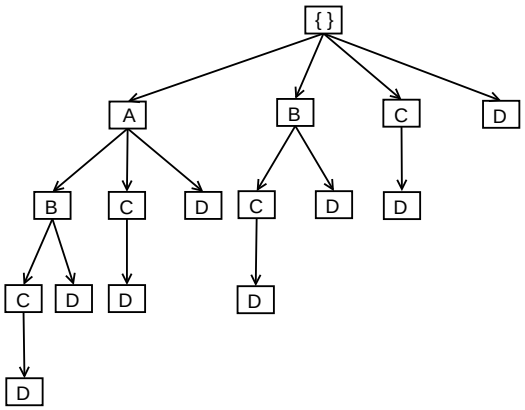


Figure 5.2: Ordered Exhaustive search

search turns into an exhaustive search. There is some work on determining a good beam size for a single dataset [134] but it is still an open question how to choose b. In our experiments we tested the beam sizes $1, 2, 4, \ldots, 2^i, \ldots, 2048$.

Our implementation of the beam search generates all attribute-value pairs in each step but only adds a rule if it is not contained in the current beam. Thus, the search space is basically unordered due to many possible refinement paths starting from different attribute tests. Therefore, the algorithms only store what attributes are already used for one single rule because for nominal attributes a test of an attribute should only occur once in a rule.

Figure 5.1 also shows an example for beam search with a beam size $b = 2$. Assume the same start situation as in the example before. Beam search is able to both refine `A` and `B`. It adds conditions in the same way as done in hill-climbing. At last, both rules contain the same four conditions. In this case, as described above, the second one would not be added to the current beam. Instead, the rule with the second best evaluation is included in the beam.

5.2.3 Exhaustive Search

There are some global rule learning systems that incorporate an exhaustive search, most notably OPUS [170]. The SimpleSeCo implementation sketched in Section 3.5.4 suffers from the problem that the same rule can be reached over multiple refinement paths, as also shown in Figure 5.1. Presumably for this reason, Quinlan and Cameron-Jones did not include a true exhaustive search into their comparison, but used a maximum beam width of 512 (cf. [134]).

We implemented a more efficient version, based on the ordered search algorithm of the OPUS^o rule learner [170]. As shown in Figure 5.2, the algorithm only

generates each rule once, and therefore does not has to check if it is already contained in the beam. Thus, the computational demands of the exhaustive search are much smaller than those of a beam search with $b \to \infty$ because the latter may generate multiple paths that end in the same refinement. Note that the "true" exhaustive search is not included in the SECO-Framework yet, whereas the inefficient exhaustive search (beam search with $b \to \infty$) is an actual component.

5.3 Experimental Setup

Our goal was to evaluate the predictive performance of different heuristics on a large number of datasets with different beam sizes. To illustrate the effects of increasing beam sizes we also experimented with a rule learner that only learns one single rule for each class. Ideally, these effects are illustrated best on datasets where local minima occur or which are hard to learn. As there are some datasets that show strong performance variations among different beam sizes, we also include some plots of individual datasets.

We used nine different heuristics described in Section 2.6. These were *precision*, *Laplace*, the *m*-estimate, *accuracy*, *wra*, the *relative cost measure*, the *odds ratio*, *correlation*, and the *Linear Regression* heuristic described in Section 4.3. Each of these heuristics implements its own strategy for navigating the search process in the right direction (cf. Section 2.6).

Two heuristics, *m*-estimate [15] and the *relative costs measure* [60], have parameters. Based on the results of Section 4.2, we will use the settings $m = 22.466$ for the *m*-estimate, and $c_r = 0.342$ for the *relative cost measure*.

It should be noted that classical rule evaluation metrics, with which we are dealing in this thesis, typically focus on evaluating the discriminatory power of a rule. However, if we consider the learning of a rule as a search problem, as we do in this chapter and have done in the meta-learning experiments in Section 4.3, a candidate rule should rather be evaluated by its potential of being refined into such a rule. In particular for the parametrized and meta-learned heuristics mentioned above, the parameter has been optimized in the context of a hill-climbing algorithm, and will implicitly take this into account.

Some of the commonly used UCI datasets [46] were too big to use (in terms of attribute-value pairs and classes) due to the vast memory demands of larger beam sizes. The datasets we used were:

> *autos-d, balloons, breast-cancer, breast-w-d, bridges2-d, colic.ORIG-d, contact-lenses, hayes-roth, hepatitis-d, monk1, monk2, monk3, mushroom, primary-tumor, promoters, solar-flare, soybean, tic-tac-toe, titanic, vote-1, vote, zoo*

These datasets are described in Section 4.1.

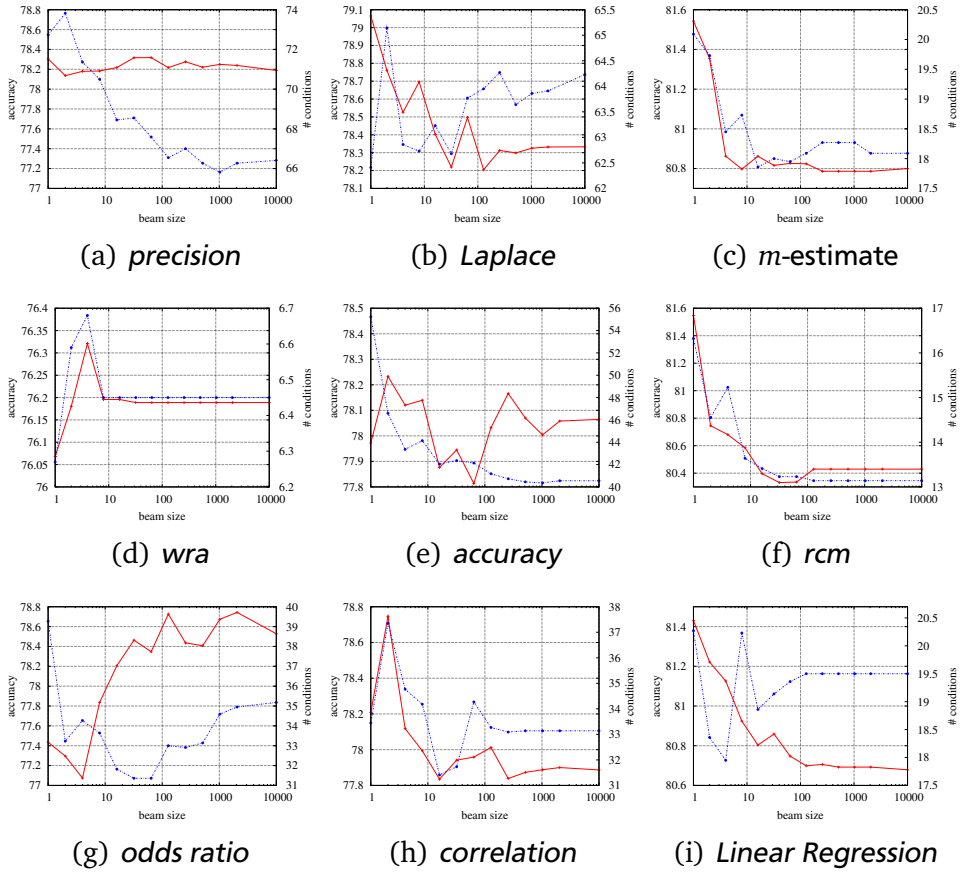


Figure 5.3: Accuracy (solid) and number of conditions (dotted) vs. beam size for all heuristics averaged over all datasets

We focused on datasets with primarily nominal attributes, but also included some that contained numerical attributes (marked with the suffix "-d"). In this case, the numerical attributes were discretized into ten different values, using equal-width discretization.

On each dataset, a 10–fold stratified cross-validation was used to obtain performance statistics for all different combinations. As a crude measure for comparing the performance of heuristics we use the macro-averaged accuracy (cf. Section 2.7.4). We are aware of the problems with averaged results. However, we will also look at the behavior of individual datasets. We do not report training set accuracy, as it will typically increase with increased search effort.

Finally, we also report *runtime* measurements, and the *size* of the learned theory in terms of the total number of conditions.

5.4 Results

This section provides a detailed discussion of our empirical study. We examine each heuristic in terms of the predictive accuracy and size of the learned theories (Section 5.4.1). Thereafter, in Section 5.4.2, we discuss the complexity and accuracy of single rules. To illustrate how the rules change when they are searched more exhaustively we also included a version of the algorithm where the outer covering loop is removed (cf. Algorithm 3.1), so that only a single rule is learned for each class. Subsequently, we discuss some results on individual datasets (Section 5.4.3), and also compare the runtimes of the different methods (Section 5.4.4). Additionally, in Chapter 9 a discussion is given about the interaction of covering algorithms with deep search algorithms.

5.4.1 Varying the Beam Size

Figure 5.3 displays the results in terms of accuracy (left y-axis, solid line) and number of conditions (right y-axis, dotted line) for all heuristics. The x-axis displays the varied beam sizes in logarithmic scaling. A beam size of $10,000$ is used for denoting exhaustive search in the graph (but the search was performed in a truly exhaustive fashion, as described in Section 5.2.3).

The individual results differ considerably. Some heuristics show a clear profit of the simultaneous refinement of more than one search path. *Precision*, e.g., remains at the same performance level (with some fluctuation) and simultaneously decreases the theory size. This is a bit surprising, as one might expect that exhaustive search will be more likely to discover overfitted rules that cover only a few positive and no negative examples, but are chosen because they have a higher precision than any other rule that covers some negative examples. Part of the explanation is that we use positive coverage as a tie breaker when multiple rules have the same evaluation (cf. Section 3.2.1), which means that we will always find the pure rule with the highest coverage. As we will observe later (in Section 5.4.2), this results in longer rules that nevertheless have a higher coverage than the rules found by hill-climbing. This effect is particularly pronounced for *precision*, but can also be observed for other heuristics, where the tie-break rule does not play such a prominent role.

Laplace shows much more fluctuation than *precision* and more exhaustive search ends up with a lower accuracy than the simple hill-climbing. This, essentially, confirms the results of [134]. Interestingly, in contrast to the results of *precision*, the theory size seems to increase steadily (with the exception of the outlier for a beam size of two). However, both *precision* and *Laplace* arrive at very similar theories with about 65 conditions on average, and an average predictive performance of about 78.2% (note that the scales of the graph differ not only on an absolute scale

but also relative to the two curves). Apparently, while the *Laplace* measure is effective in preventing overfitting for hill-climbing algorithms, its performance degrades to the performance of *precision* when used with exhaustive search.

The results of the m-estimate in terms of accuracy are quite similar to those of *Laplace*, except that at all beam sizes, it learns much smaller and more accurate theories. The m-estimate does not show strong fluctuations in theory size like *Laplace*. Contrarily to the latter it ends up with a smaller theory. Remind that the m-value that we are using in these experiments has been tuned in a hill-climbing algorithm (cf. Section 4.2), which may, in this case, be partly responsible for the observed performance degradation for higher beam sizes. One noticeable difference is that the larger theory sizes at lower beam sizes appear to be systematic here.

Wra, on the other hand, is a very stable heuristic. The reason for this behavior is that it effectively over-generalizes [104]. It learns by far the smallest theories (only about 6.5 conditions on average), with a performance that lags considerably behind the performance of the other heuristics. Thus, the optimal rules are typically found at very shallow search depths and comparably small beam sizes. *Accuracy* is much less stable and shows fluctuations throughout the entire range. Its worst behavior is with beam sizes of about ten to 100. The behavior of the *relative cost measure (rcm)* [60], which is a generalization of *wra* to arbitrary linear costs between the *tpr* and the *fpr*, is quite similar to the behavior of the m-estimate. This, however, is not surprising, because its parameter has also been optimized for hill-climbing, and effectively realizes a quite similar heuristic as the optimal choice of the m parameter. This has been previously noted by inspection of the coverage space isometrics of both heuristics (cf. Figure 4.3), and is confirmed here.

For *odds ratio*, we can observe a steady increase in accuracy with increasing beam sizes. For the theory size we can see that it first decreases strong but then it gets bigger and ends up in about four conditions less than with hill-climbing. Here, exhaustive search clearly helps to discover better theories.

Interestingly all three heuristics that were optimized for hill-climbing (m-estimate, *relative cost measure (rcm)*, and the *Linear Regression* heuristic) do not perform well under deeper searches when comparing hill-climbing with exhaustive search. In all cases, exhaustive search finds simpler theories than greedy search, but these are of a lesser quality. This is consistent with several previous results that show that in contrast to the assumptions of *Occam's razor*, simpler theories often exhibit a worse performance (we refer to [37] for a summary of such results). We explain these results with the fact that these heuristics have been optimized for hill-climbing (cf. Section 4.2), and that they thus implicitly take the search process into account. As discussed above, a good heuristic for a hill-climbing search should try to predict the quality of the best rule to which it can be refined to, in order to make sure that the path to the best final rule can be found. This, on the other hand, is not necessary for exhaustive search, where we are guaranteed to find the best rule.

Table 5.1: Average number of rules and conditions and their ratio for hill-climbing and exhaustive search, averaged over all datasets

heuristic	beam	avg. # rules	avg. # conditions	$\frac{\#\text{ conds}}{\#\text{ rules}}$
precision	1	27.68	72.73	2.63
	∞	20.36	66.41	3.26
Laplace	1	24.23	62.41	2.58
	∞	19.73	64.23	3.26
m-estimate	1	8.36	20.09	2.40
	∞	7.05	18.09	2.57
wra	1	3.32	6.27	1.89
	∞	3.18	6.45	2.03
accuracy	1	20.23	55.23	2.73
	∞	13.00	40.55	3.12
rcm	1	7.00	16.32	2.33
	∞	5.32	13.14	2.47
odds ratio	1	11.64	39.27	3.38
	∞	11.5	35.18	3.06
correlation	1	12.55	33.45	2.67
	∞	11.23	33.14	2.95
Linear Regression	1	8.41	20.27	2.41
	∞	7.18	19.50	2.72

5.4.2 Single Rules

The results in the previous section have shown that exhaustive search generally finds simpler theories, i.e., theories with a smaller total number of conditions. However, a closer investigation of this result uncovers an interesting finding. Table 5.1 shows the average complexity of the theories found by hill-climbing (first line) and exhaustive search (second line). As expected, both the total number of rules as well as the total number of conditions are much lower for exhaustive search than for hill-climbing search. Thus, exhaustive search clearly finds more compact theories than hill-climbing search.

However, if we look at the ratio of these two values, i.e., at the average number of conditions per rule, exhaustive search almost always (*odds ratio* being the only exception) finds longer rules than hill-climbing. Together with the result that it also induces fewer rules, this means that on average, the rules found with exhaustive search must have higher coverage than the rules produced with hill-climbing. This

Table 5.2: Macro-averaged accuracy and number of conditions averaged over all datasets for a learner that only learns a single rule per class

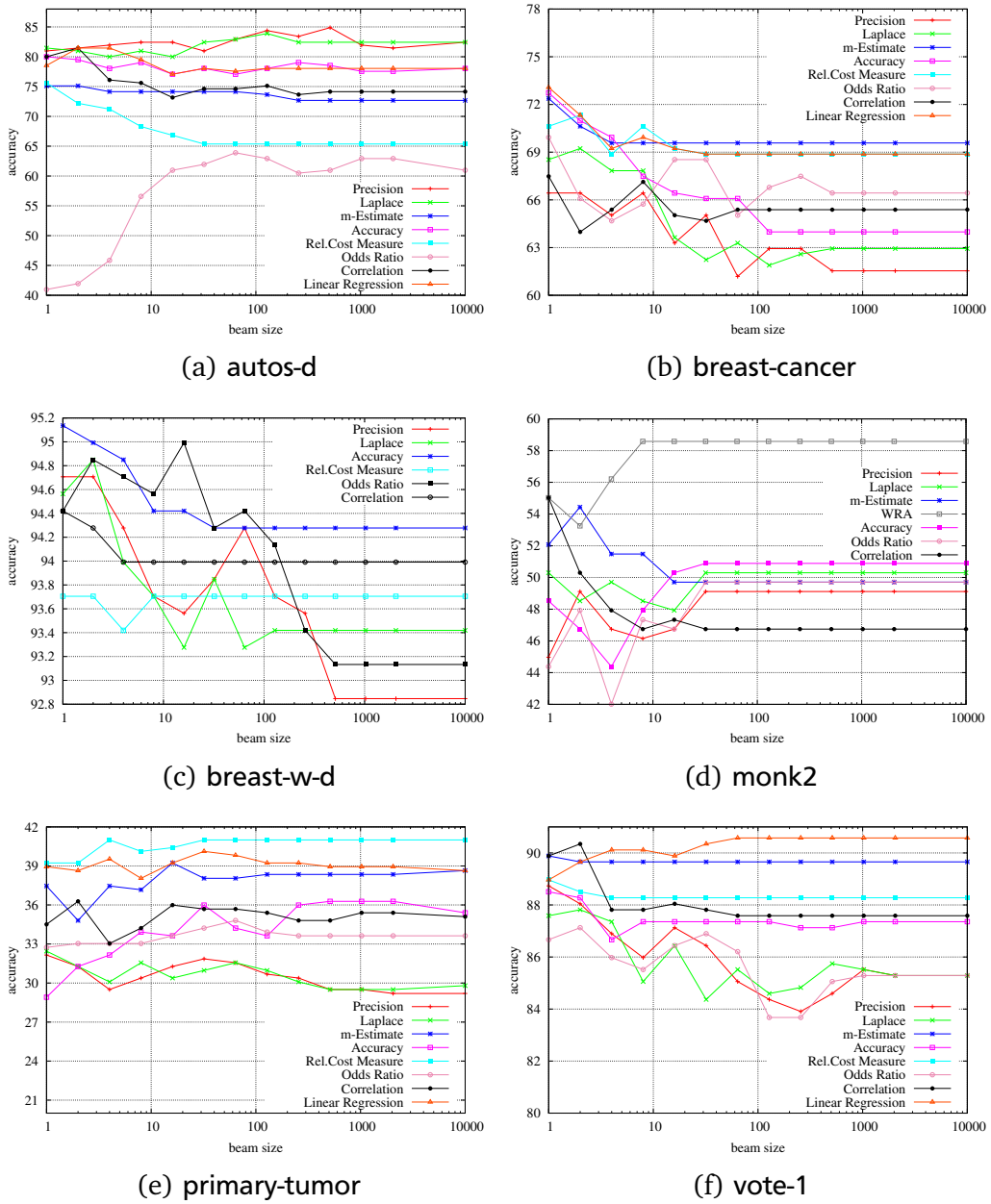
heuristic	beam	avg. acc. macro	avg. # conditions
precision	1	64.67	6.82
	∞	68.55	9.59
Laplace	1	66.13	6.86
	∞	68.78	9.64
m-estimate	1	70.87	7.23
	∞	71.32	7.36
wra	1	68.14	3.23
	∞	68.81	3.50
accuracy	1	72.50	6.64
	∞	73.45	9.55
rcm	1	72.31	5.64
	∞	72.69	6.00
odds ratio	1	63.99	4.05
	∞	65.57	5.05
correlation	1	72.41	7.14
	∞	72.72	8.95
Linear Regression	1	70.38	7.27
	∞	70.30	7.50

is contrarily to the intuition that rules with higher coverage tend to be more general and therefore shorter than rules with lower coverage.

This result is particularly surprising if we consider that this effect happens even though we implement a bias for more general rules by preferring rules with more coverage in the case of ties. We had expected that this strategy will particularly favor shorter rules because they tend to have a higher coverage, but as the results show, the found rules are considerably more complex, but nevertheless more general. In retrospect, however, it is maybe not surprising that complex high-coverage rules are overlooked by hill-climbing search in favor of shorter rules with somewhat lower coverage.

To ensure that our interpretation of the results are not influenced by the covering strategy, we also performed a second set of experiments, in which we only induce a single rule per class. The purpose of these experiments is to observe the change in terms of conditions per rule for growing beam sizes.

Again, Table 5.2 compares the performance of the hill-climbing search (first line) to the performance of exhaustive search (second line). The accuracies are obtained



(a) autos-d (b) breast-cancer

(c) breast-w-d (d) monk2

(e) primary-tumor (f) vote-1

Figure 5.4: Beam size vs. accuracy for single datasets

by using the rules as a complete model. Invariably, for all heuristics, exhaustive search finds rules that are longer, but more accurate, which conforms to the results of Table 5.1.

If we compare the results with those where complete models are learned in terms of accuracy, it is rather surprising that they achieve such high values. Most of the heuristics only lack about 10% accuracy on average, *accuracy* and *correlation* only decrease their performance by approximately 5%. Comparing their model size, both *precision* and *Laplace* are able to achieve this performance level with theories that are about seven times smaller than the complete theories, for the other heuristics the sizes are reduced to approximately half the size. This confirms the good results for decision stumps in comparison to decision trees [80].

5.4.3 Results for Individual Datasets

Of course, averaged accuracy over several datasets is a very crude and not very meaningful summary measure that we only used because of the lack of a better alternative (for discussion of that topic see Section 2.7). To illustrate the effects of the different search mechanisms without averaging the values, Figure 5.4 displays results of all interesting heuristics (those where some changes happen) on six selected datasets. The x-axis displays the beam size on a logarithmic scale (again, a beam size of $10,000$ denoting exhaustive search), and the y-axis depicts the cross-validated accuracy of the complete rule-based theory on this dataset. Most of the heuristics show some fluctuation between different beam sizes, but typically a clear trend can be recognized. However, some heuristics, primarily *wra*, remain quite constant over all sizes and data sets, for reasons we already discussed above. Thus, we only included this heuristic in the plot for `monk2` because there its performance was superior to the other measures. After a degradation of 1.78% in accuracy it achieves the highest value with a beam size of eight and from then on does not change any more.

Note, e.g., that in the two datasets in the bottom row, the best overall result has been achieved with exhaustive search, clearly outperforming the best results obtained with hill-climbing or beam search with low beam sizes. However, the oversearching phenomenon can be clearly observed for some heuristics (*Laplace* and *precision*) in the same plots. On other datasets, in particular `breast-cancer` and `breast-w-d`, all heuristics show strong oversearching. Among all plots, `breast-w-d` shows the strongest fluctuations. However, this is partly due to the different scale: note that the y-axis only displays about 2% of change in accuracy.

The dataset `autos-d` is one example for the constant performance gain of the heuristic *odds ratio*, but also of its bad performance in hill-climbing. Up to a beam size of 64, it profits from bigger beam sizes before the accuracy falls down again. High beam sizes > 512 show a comparable accuracy but the theory size drops by two conditions. Contrarily to the *odds ratio*, the *relative cost measure* loses performance. The performance of all other heuristics varies slightly, some ending up better others worse than hill-climbing.

Table 5.3: Runtimes (in sec.) of the heuristics with different beam sizes

beam size	*precision*	*Laplace*	*m*-estimate	*wra*	*accuracy*
1	884.2	808.4	495.6	299.2	741.3
2	1,405.1	1,243.7	632.3	362.7	1,024.6
4	1,447.4	1,292.6	684.7	386.0	1,037.3
8	1,520.8	1,423.8	729.4	431.3	1,063.1
16	1,586.9	1,571.1	817.0	439.9	1,141.1
32	1,721.7	1,787.1	879.1	481.3	1,211.2
64	1,891.5	1,950.7	935.6	503.1	1,288.1
128	2,102.4	2,098.5	1,066.1	577.2	1,443.1
256	2,466.0	2,518.8	1,279.7	679.0	1,660.5
512	3,380.5	3,341.6	1,670.2	786.9	2,061.3
1024	4,626.4	5,183.8	2,344.2	976.1	2,727.1
2048	8,227.3	7,638.2	3,395.8	1,473.0	3,904.4
∞	12,061.1	11,764.1	3,078.9	928.8	5,596.6

beam size	*rcm*	*odds ratio*	*correlation*	*Meta-learned*
1	234.5	615.0	586.3	457.5
2	334.7	793.2	836.8	627.9
4	360.6	832.3	938.2	710.6
8	359.0	890.9	1,007.5	764.8
16	363.0	920.0	1,074.7	826.8
32	376.8	981.6	1,170.4	924.1
64	394.2	1,112.6	1,291.8	1,074.0
128	412.0	1,387.6	1,495.1	1,218.1
256	436.2	1,651.1	1,861.2	1,445.1
512	480.9	2,042.8	2,379.7	1,871.4
1024	526.0	2,802.2	3,300.6	2,885.0
2048	576.3	4,076.6	5,174.1	3,903.2
∞	1,714.3	4,310.4	5,076.4	4,359.7

Another interesting plot is the one of `primary-tumor`. Initially, all heuristics show relatively strong fluctuations, but once the values stabilize, this is a good example for a dataset where most of the heuristics show a clear improvement when moving towards exhaustive search. Namely about four heuristics show improvements, two remain at the same level and two decrease performance. This dataset has 17 nominal attributes which have 43 values in total and 22 classes. Thus the search space for this data set is rather large which explains why we still can see changes when moving from beam search with $b = 2048$ to exhaustive search.

5.4.4 Runtime of the Search Methods

For completeness, Table 5.3 shows the runtimes in seconds for the different heuristics. Not surprisingly, the runtime generally increases for more exhaustive searches. However, it is interesting to look at the relative increase for consecutive beam sizes. Some measurements hardly change (e.g., *accuracy* with beam size two to 16) which means that the number of evaluated rules does not change much. This, again, reflects the fact that some heuristics prefer general (short) rules, and these are already found with low beam sizes.

On the other hand, both *precision* and *Laplace* have a strong bias towards overfitted rules which is also reflected in their runtimes. As the beam grows, the runtime of the algorithm increases. When changing the search to an exhaustive one the time grows again. *Precision* with over three hours takes the maximum amount of total runtime among all heuristics.

Note that the implementation of the exhaustive search sometimes is even more efficient than the beam search with $n = 2048$. But this holds mostly for heuristics that do not induce many candidate rules as described above.

5.5 Bidirectional Rule Learning

So far we have only focused on algorithms that employ a top-down strategy for the induction and the refinement of rules. Nevertheless, there are other strategies to guide the search. One example would be a bottom-up search where the refinement operator used to refine a candidate rule deletes conditions instead of adding them. Another technique is to combine the top-down approach and the bottom-up search yielding bidirectional search. In this section this strategy is examined in more detail. The relation of the bidirectional search and the optimization phase of RIPPER is emphasized here. The idea of the optimization phase of RIPPER essentially is to ensure that the rules are flexible enough to cope with the characteristics of many different datasets. RIPPER uses the I-REP strategy [61] to prune the current rule directly after it is learned. Already in this step a suboptimal previous decision (in this case the addition of a condition) can be undone. Essentially, this procedure also makes the rules more flexible and decreases the chance to get stuck in a local optimum.

An even stronger influence is imposed by the optimization phase. Here, each rule is examined whether it was a good choice in context of the whole rule set (which clearly is inaccessible during the search for a single rule). In other words, if the rule was found due to a local optimum that could not be discovered because hill-climbing search was used, it is checked whether a better rule was possible or not. This step could also be performed during the process of learning a rule. In this

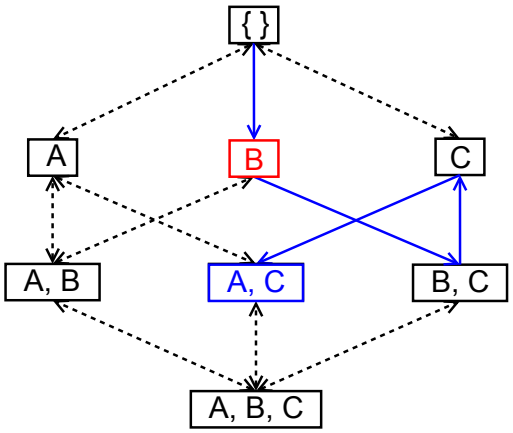


Figure 5.5: Bidirectional search, based on [95]

case, it would be necessary to evaluate the addition and the deletion of a condition when the candidate rule is refined.

The goal of this section is to gain an understanding whether the good results of RIPPER come from the use of I-REP and/or the optimization phase or if it would be sufficient to alter the search strategy in a way that will allow for a greater flexibility. Therefore, a bidirectional search was implemented in the SECO-Framework. Then, the new approach was compared against the regular top-down strategy and to RIPPER. This section is based on the work of Karpf [95].

5.5.1 Theoretical Considerations

When the learning algorithm is able to employ a bidirectional search some problems may occur that can be neglected as long as a top-down approach is utilized. The main problem is that the algorithm may get stuck in a cycle due to the undirected search by adding and deleting the same condition over and over again. For an example of this situation consider Figure 5.5. It is related to the Figures 5.1 and 5.2 above where a hill-climbing, a beam search, and an exhaustive search were displayed. Nevertheless, the semantics of Figure 5.5 are different. Here each capital letter (`A`, `B`, `C`) stands for a single condition (a refinement, e.g., $A_1 = x_{1,1}$) and not for an attribute.

The Figure displays the search space for a bidirectional learning algorithm. Note that the graph is undirected, i.e., a refinement can be conducted by adding or by deleting a condition. The solid arrows show a possible refinement path where the condition `B` (depicted in red) was a suboptimal choice, i.e., a local optimum. In a hill-climbing search the algorithm is unable to revert its wrong decision because `B` has the highest evaluation value in the first step. But in a bidirectional

search the algorithm is able to delete the condition `B` again and to navigate to the global optimal rule with conditions `A,C` (in the lower middle of Figure 5.5) which is impossible for the rule learners we have used so far.

Considering Figure 5.5 a cycle occurs if the best refinement of the candidate rule `B,C` is to delete `C` and the best refinement of candidate rule `B` is to add `C`. This problem and some possible solutions are described later.

When an exhaustive search is used the algorithm cannot get stuck in local optima. Already with a beam search the chance of finding a suboptimal rule decreases with an increase of the size of the beam. For these reasons, the bidirectional search may be viewed as a more efficient solution for omitting local optima. Later in this section we will evaluate whether the performance of a bidirectional search is comparable to an exhaustive search (cf. Section 5.5.3). A discussion of the different strategies, their advantages and disadvantages will be given in Section 5.5.4.

Note that, as described above, RIPPER also is able to reverse previous decisions by making use of the I-REP strategy and an optimization phase. This can either be done directly after the rule is learned because it is pruned on the pruning set or after the learning is finished via the optimization phase. To illustrate how a suboptimal choice during the learning process is reversed assume that the rule with the conditions `B,C` was added to the theory that the RIPPER algorithm has learned. Recall that during the optimization phase RIPPER creates the *Revision* and the *Replacement* (cf. Section 3.5.3). The first one is a rule that is built by refining the current rule and the second one is a rule that is learned completely new. These two rules and the original one are then evaluated by the MDL metric in the context of the current theory as described in Section 3.5.3 and the best one is selected. In case of the given example the algorithm would select the potential *Replacement* rule with conditions `A,C` and thus would also avoid getting stuck in a local optimum.

A problem of using a bidirectional search is that an infinite loop may occur when the same condition is deleted and added again. These cycles are impossible with directed graphs as those of a top-down or a bottom-up search are. We followed Karpf [95] and restricted the algorithm so that only refinements that have a strictly greater heuristic value than their predecessor are added. Other ways to prevent the algorithm from getting stuck in such a cycle could be for instance to mark all visited candidate rules but we will not consider them here. For a detailed discussion about the different approaches and their advantages as well as their limitations see [95].

5.5.2 Experimental Setup

The main goal of the experiments was to investigate whether the good performance of RIPPER can be reached by means of more flexible search algorithms. Following [95], the following 19 datasets were selected for the experiments. A broad variety of different characteristics of the datasets is given by this selection. All of them are described in Section 4.1.

Table 5.4: Results of the top-down and the bidirectional approach

search strategy	heuristic	avg. acc. macro	avg. acc. micro	avg. # rules	avg. # conds	avg. rank
bidirectional	*m*-estimate	86.52	93.58	18.68	54.95	3.47
top-down		86.43	93.65	18.78	55.32	3.42
bidirectional	*Laplace*	85.59	92.92	54.16	129.84	4.45
top-down		85.52	92.92	54.05	129.42	4.56
bidirectional	*correlation*	84.90	92.34	15.90	41.00	4.50
top-down		84.89	92.36	18.58	54.00	4.50
bidirectional	*wra*	83.20	90.23	6.05	20.16	5.63
top-down		83.22	90.29	6.11	20.47	5.47

anneal, audiology, auto-mpg, breast-cancer, breast-w, cleveland-heart-disease, credit, credit-g, glass, hypothyroid, ionosphere, krkp, mushroom, segment, sick-euthyroid, soybean, tic-tac-toe, titanic, vowel

The means for evaluating the algorithms are given in Section 2.7. For the comparison of the top-down strategy and the bidirectional approach the SimpleSeCo rule learner as described in Section 3.5.4 was used. The experiments were conducted using five heuristics, namely *correlation*, *foil gain*, *Laplace*, the *m*-estimate, and *wra* (all described in Section 2.6). This selection is based on the same motivations as stated above, i.e., to have a group of heuristics that employ different biases. These different heuristics excluding *foil gain* were used for the comparison of the top-down and the bidirectional approach. The main goal here was to give a fair comparison, so that the chance that one approach is better simply because it works better with a certain heuristic decreases.

For all algorithms the average rank, micro-, and macro-averaged accuracy is given. These averages are calculated with a 10–fold cross-validation (cf. Section 2.7.1).

5.5.3 Results

In this section an overview of the results is given. First, the two strategies top-down and bidirectional are compared, followed by a comparison of the bidirectional strategy and Ripper. A more detailed evaluation and some additional results is given by Karpf [95]. The section is completed by a comparison of the bidirectional search with a beam and an exhaustive search.

To identify whether or not the SimpleSeCo algorithm is able to benefit from a bidirectional search a comparison of both variants was done. The results are displayed in Table 5.4 (taken from [95]). The algorithm with the highest macro-averaged accuracy was the bidirectional approach used with the *m*-estimate, closely followed by the regular top-down approach with the same heuristic. The average rank of the latter was even better than that of the bidirectional algorithm. All other heuristics also show quite similar behavior except *correlation*. Practically, this means that the rule sets do not differ much. On the one hand this can mean that the SimpleSeCo generally cannot benefit from the more flexible bidirectional search. But on the other hand, the datasets may not contain many situations where local optima occur. For this reason, the advantages of the bidirectional search do not come to play.

The only major changes occur when the heuristic *correlation* is used. The results on three out of the 19 datasets were considerably different. These were `tic-tac-toe`, `credit-g`, and `breast-cancer`. The behavior of the two approaches is described below for these three datasets (results for `tic-tac-toe` and `breast-cancer` can be found in Table 5.6.

On the dataset `tic-tac-toe` the top-down approach with *correlation* finds ten rules with 44 conditions and is able to achieve an average accuracy of 82.78%. The bidirectional approach finds only five rules with 13 conditions but also lacks a bit behind in accuracy (81.52%). Interestingly, the first five rules are identical and then top-down finds a rather suboptimal rule containing seven conditions (the number of attributes in the dataset is nine) with a coverage of two positives and no negative example. The bidirectional approach seems to get stuck due to the requirement that the heuristic evaluation has to be greater for each refinement and returns a rule that covers more negative than positive examples. Recall that this is a hard-coded stopping criterion to stop the induction of additional rules (cf. Section 3.2.1). Therefore the theory is finished already at that point.

The dataset `credit-g` exhibits the strongest fluctuations among the two approaches. Here, the top-down approach learns 47 rules with 215 conditions and yields an accuracy of 69.7%. In contrast, the bidirectional approach finds only nine rules with 40 conditions and is able to increase the accuracy to 70.3%. The high number of rules and conditions found by the top-down approach might overfit the data due to the lower accuracy. The difference in the rule sets is noticeable right at the first rule. The top-down approach ends up in a rule with 13 conditions covering 79 positive and 27 negative examples (heuristic value=0.335). The bidirectional approach seems to suffer from the requirement that the heuristic value has to be strictly greater in each refinement step and finds a worse rule that at least still is able to reduce overfitting. Due to the different coverage statistics the whole learning process changes then.

Table 5.5: Results of the comparison between RIPPER and the bidirectional approach

algorithm	heuristic	avg. acc. macro	avg. acc. micro	avg. # rules	avg. # conds	avg. rank
RIPPER	*foil gain*	85.94	92.99	9.53	22.63	1.34
bidirectional	*correlation*	84.99	92.41	9.79	27.68	1.90
bidirectional	*foil gain*	67.02	79.21	2.95	5.32	2.76

On the dataset `breast-cancer` the same observations are valid that were discussed before for the dataset `tic-tac-toe`.

In this sense, it seems that the entire algorithm has to be adapted to a bidirectional search. A simple exchange of the search method seems not to be beneficial. On the other hand, major differences only occur for the heuristic *correlation*. The other heuristics exhibit small changes that are rather negligible. In summary, in the setup used for the previous experiment, a bidirectional search is unable to improve the algorithms performance, neither in terms of accuracy nor in theory size.

Comparison of RIPPER and the Bidirectional Approach

Table 5.5 displays the result on the 19 datasets of the regular RIPPER algorithm and of two bidirectional alternatives [95]. One of them uses the same heuristic as RIPPER and the other uses *correlation*. Note that the requirement of a refinement to be strictly greater than its predecessor interferes with the properties of the *gain*-heuristics. The premise that a refinement has to have a greater value than its predecessor is essential to omit that the search gets stuck in a cycle. The heuristic value of *foil gain* however, naturally decreases with each refinement (cf. Figure 3.1). In a top-down search where the last refinement is returned for *gain*-heuristics this has no effect. For bidirectional search this is a crucial issue and results in empty theories. Usually, no refinement can be found for which the heuristic value increases compared to its predecessor. Table 5.5 illustrates this effect because only about three rules with 5.32 conditions are found in average for the bidirectional search with the heuristic *foil gain*.

For this reason, we decided to include another heuristic, namely *correlation* because it proved to be a good choice before. Furthermore, it does not feature a parameter but uses a fixed trade-off. Since we do not want to bias the heuristic for this comparison it seems to be beneficial to employ a fixed trade-off. Nevertheless, both versions of the bidirectional search are worse than the regular RIPPER. In summary, the bidirectional search is unable to outperform RIPPER.

Table 5.6: Comparison of bidirectional and beam search for the heuristic *correlation*

dataset	algorithm or beam size (b)	avg. acc. macro	# rules	# conditions
breast-cancer	bidirectional	65.38	10	28
	$b = 1$	67.48	15	45
	$b = 2$	63.99	28	98
	exhaustive	65.38	25	83
soybean	bidirectional	92.39	32	62
	$b = 1$	90.48	31	67
	$b = 2$	90.63	29	62
	exhaustive	92.39	25	51
tic-tac-toe	bidirectional	81.52	5	13
	$b = 1$	82.78	10	44
	$b = 4$	82.46	11	41
	exhaustive	82.36	9	33

Comparison of Bidirectional, Beam, and Exhaustive Search

Another interesting question is how a bidirectional search behaves compared to the results of the different beam sizes and the exhaustive search of Section 5.4. The main focus lies on an analysis whether the bidirectional search outperforms the top-down approaches that refine many candidates simultaneously. On some datasets, and for some heuristics the exhaustive search was unable to improve the quality of the rule sets. Especially *wra* has shown no notable changes over the datasets used in Section 5.4 (cf. Figure 5.3 (d)). Hence, we decided to focus on the m-estimate and on *correlation*. Table 5.6 shows average accuracies and the number of rule and conditions for three datasets that were the most interesting. The algorithm was used with the heuristic *correlation*.

On the dataset `breast-cancer` the bidirectional search has the same accuracy as an exhaustive search but with only ten rules with 28 conditions (compared to 25 rules with 83 conditions). The theory is much smaller than the rule set of the exhaustive search. Still, the bidirectional search could not find the best theory. The simple hill-climbing search ($b = 1$) has a perspicuously higher accuracy but also with a few more rules.

The dataset `soybean` also shows the same accuracies for the bidirectional and the exhaustive search, but the latter finds a smaller theory. Other choices of the beam size are suboptimal.

Table 5.7: Comparison of bidirectional and beam search for the heuristic *m*-estimate

dataset	algorithm or beam size (b)	avg. acc. macro	# rules	# conditions
soybean	bidirectional	92.24	35	71
	$b = 1$	93.41	33	72
	$b = 2$	92.83	29	65
	exhaustive	92.68	27	66
tic-tac-toe	bidirectional	98.23	9	28
	$b = 1$	97.70	9	28
	$b = 2$	97.18	12	44
	exhaustive	96.45	10	32

The dataset `tic-tac-toe` illustrates that the bidirectional search is unable to guarantee that the best theory will be found. Here, also the hill-climbing search returns the best rule set.

Table 5.7 shows the same entries as Table 5.6 but for the heuristic *m*-estimate. The dataset `breast-cancer` is omitted because no notable differences between the algorithms could be noted. For the dataset `soybean`, basically the same observations as for the heuristic *correlation* are valid. The exhaustive search is even able to improve the accuracy in comparison to the bidirectional search. On the dataset `tic-tac-toe` the bidirectional search was the best algorithm which is in contrast to the results of the *correlation* heuristic.

5.5.4 Discussion

To sum up, the bidirectional approach is unable to improve over the basic top-down algorithm (cf. Table 5.4). It also is worse than RIPPER (Table 5.5). An extensive comparison to different beam searches and an exhaustive search was not performed and is subject to further research. Nevertheless, the direct comparison of some particularly interesting datasets showed that the results there are mixed (Table 5.6 and 5.7). Due to the good performance of a simple hill-climbing search, presumably mostly driven from the problem of oversearching (cf. Section 5.4), a notable improvement over different beam searches is in general rather implausible. The main focus of these experiments lies on theoretical considerations, i.e., whether a bidirectional search should be preferred over a beam search in general. The limited results given here do not allow a solid conclusion on that topic.

Some kind of processing of the induced rules seems to be necessary as the good performance of RIPPER illuminates. A bidirectional search serves as a means for adjusting the search algorithm to reach this objective in a more direct way. RIPPER separates the search for a good rule and the tuning of this rule. In a bidirectional search these two processes are interwoven. A beam search may be viewed as a naive approach that does not change the abstract search routine but simply executes it several times in parallel. Due to the massive runtime an exhaustive search needs, this strategy primarily was implemented to illustrate the oversearching problem empirically, rather than for practical purposes.

5.6 Related work

Most classification rule learning algorithms use hill-climbing as their method for greedily adding conditions to a rule and some use bidirectional search where, as we have seen, the deletion of a condition is also possible. Local pattern discovery algorithms, such as association rule or subgroup discovery often use some form of an exhaustive search. Beam search can be viewed as a means for trading off between hill-climbing and exhaustive search. In this chapter, the question we posed was how the quality of the found theories changes with increased search effort or with a bidirectional search.

Several authors have previously observed the phenomenon of oversearching, which essentially says that increased search effort may lead to a decrease in accuracy. For example, Murthy and Salzberg [123] have found that increasing the look-ahead in decision tree induction will typically no longer improve the results, and may also produce larger and less accurate trees. Specifically for inductive rule learning, it was shown by Quinlan and Cameron-Jones [134] that more search does not necessarily lead to better predictive accuracy. However, this work was limited to the use of a single heuristic for evaluating rules, the *Laplace* error. Segal [151] suggested to change the search heuristic to circumvent oversearching. To do so, *Laplace-depth* was suggested, a heuristic based on *Laplace* that includes a preference for short rules. However, *Laplace-depth* was able to reduce oversearching, but it was unable to eliminate it entirely. Similarly, a recent work also claims that the phenomenon of oversearching strongly depends on the evaluation function [139]. The proposed system called EXPLORE is able to achieve the same performance as other state-of-the-art rule learners, but needs less rules to do so. Contrarily to our approach, EXPLORE performs an exhaustive search over the whole rule set instead of single rules. Due to the vast computational demands, the rule sets currently are rather small but nevertheless show a good performance. Surprisingly, the authors report that oversearching did not occur in their approach. In the current version *accuracy* is used as evaluation function but some optimization constraints are also imposed such that effectively a pruning is realized. Additionally, the I-REP procedure is used to find the optimal length of the rule set in a prior phase. Then the

best rule set that fulfills these length constraints is searched. Actually, most often it took too long to find a rule set with the determined length. In effect, a rule set with seven conditions was the largest one in the experimental study [139].

Webb introduced OPUS, an efficient algorithm for unordered exhaustive search [170]. In the paper, an algorithm for optimization search, i.e., heuristic search is given. We used the proposed algorithm (called $OPUS^O$) to implement an exhaustive search algorithm.

Moreover, there are a number of papers dealing with bidirectional search for rule learning. The earliest reference to a bidirectional search is the VERSION SPACE algorithm [116]. The algorithm is an iterative one that processes the instances one by one. It maintains two sets, the G-Set containing the maximally general hypothesis (that may also be denoted by rules) and the S-Set containing the maximally specific hypotheses. It converges to a single concept if one exists (i.e., a single rule). In each step, if the instance is a positive one, the hypotheses in the S-Set are minimally generalized (i.e., the generalization step of a bidirectional search is employed). For each negative instance the hypotheses in the G-Set are minimally specialized. This can be seen as the regular specialization step in the FINDBESTRULE procedure sketched in Algorithm 3.2.

The SWAP-1 algorithm [172] works by regular rule refinement but is also able to employ a deletion or a replacement of a single condition. It uses an unordered list of rules (cf. Section 2.3.4).

Other algorithms make use of stochastic and deterministic local optimizations. The algorithm K-OPT for deterministic local search was introduced and empirically compared to decision tree learners [118]. It uses a binary vector which encodes the rules and examples by a fixed set of attributes of which each one has a fixed set of values. The parameter k then is used to determine how many bits of the binary vector are exchanged in each step. Another method is a stochastic search, e.g., based on simulated annealing. An algorithm using this technique is also presented [118].

The algorithm JOJO is also able to use a bidirectional search [42]. It was later extended to FROG that allows to change more than one condition at a time [43]. There exists also a version for first-order learning called JOJO-FOL [176].

5.7 Summary

As the above experiments unfold, oversearching is not a universal phenomenon. It rather depends on the used heuristic. An exhaustive search in general finds longer rules that cover more examples compared to the more simple search mechanisms. This perhaps is most obvious in the experiments where the rule learner only learns a single rule. Interestingly, these theories that merely contain a single rule are not much worse than the complete theories. The experiments with RIPPER have shown that the performance an optimization phase offers cannot be reached by employing

a bidirectional search. Often, a bidirectional search is even unable to improve over a simple top-down search. Interestingly, albeit only for some selected datasets, a bidirectional search may end up with a theory that is comparable with one that was found with exhaustive search. This can be of practical importance as usually the runtime of a bidirectional search is much less than the runtime of an exhaustive search.

6 A Metric-Based Approach to Regression Rule Learning

The accurate prediction of a numerical target variable is an important task in machine learning. There are several scenarios where an algorithm that is able to predict a particular continuous value instead of a discrete class is better suited. For example, in the domain of financial data, a crucial issue is to predict the volume of a credit. Where classification algorithms can only provide a decision whether or not a credit should be given, regression algorithms are able to predict the exact volume of the credit. In real-world scenarios, most often the target variable is numerical instead of categorical. However, while the statistical learning community has proposed a great variety of algorithms for solving this problem, it has not received so much attention in the data mining and inductive rule learning communities, where a strong focus lies on the comprehensibility of the learned models. The main advantage of a regression rule learning algorithm is that the induced rules can be interpreted where statistical models usually do not provide an intuitive way to be comprehended.

The goal of the work reported in this chapter and in Chapter 7 thus is the design of a heuristic rule learning algorithm that learns regression models in the form of a decision list consisting of simple regression rules that have constant models in the rule head. This model class is fairly restrictive, but we can show that both approaches yield very good results within this model class. However, the quality is still below the performance of statistical approaches that incorporate linear models or boosting [48].

In the following, a straight-forward metric-based implementation is sketched and the parameter of a novel regression rule learning heuristic is tuned in the same way as described before in Section 4.2.1. As the most obvious idea is to adapt the heuristic towards evaluating regression rules, the algorithm derived in this chapter is called metric-based. To implement the algorithm, in Section 6.1 the adaptations necessary to extend classification rule learning to regression are described. Then, in Section 6.2 a summary of the datasets and the experimental setup of this chapter is given. At this point, some of the regression algorithms used for comparison are also described. The adaptation of a classification rule learner to regression imposes new parameters that are described in Section 6.1 and then optimized in Section 6.4. Section 6.5 gives a summary of the results. In the following section related work is presented and Section 6.7 gives a summary of the research presented in this chapter.

The algorithm presented in this chapter is based on the paper [89].

6.1 Separate-and-Conquer Rule Learning for Regression

Several adaptations of the SIMPLESECO algorithms are necessary when it is used to learn data that have a numerical target attribute. In summary the required changes affect

- the head of the rule,
- the stopping criterion to stop the induction of rules, and
- the heuristic to evaluate the candidate rules.

First of all, the prediction given by a rule has to be changed. As noted above, most of the strong regression rule learning algorithms use linear models as prediction. Employing linear models indeed is powerful, but the interpretability of the rules suffers, i.e., a single value is easier to understand than a linear model that depends on many input variables. Consequently, we decided to use a single numerical value as prediction in all experiments. It can be calculated as the mean or the median of the examples covered by the rule. The first minimizes the mean squared error (*mse*) and the second minimizes the mean absolute error (*mae*). We experimented with both choices. For evaluation purposes we usually use the *rrmse* to get a domain-independent measure. Since the median is known to be robust against outliers we decided to use it for prediction in the rule's head in most cases.

In classification, the default rule at the end of the (decision) list usually predicts the majority class. In regression we do not have a majority class. The mean over all examples can be seen as a variant of the majority class, but still it is unclear how many examples should be covered to ensure a reasonable performance. In classification, all classes except the largest one are covered. In regression, however, the largest class is unknown. For this reason, we experimented with different percentages of examples that have to be covered before the induction of additional rules is stopped.

The last part of the SECO-Framework that has to be changed is the heuristic. Those that were used in this thesis so far rely on coverage statistics, i.e., positive and negative examples (cf. Table 2.3). However, in regression such statistics are unavailable. There are two ways to deal with this problem.

- Either by adapting the heuristic to the statistics available in regression (i.e., loss functions),
- or by reducing the regression problem to classification.

Table 6.1: Overview of the tuning databases for regression

dataset	# instances	# nominal attributes	# numeric attributes	# distinct values
abalone	4,177	1	7	28
auto-mpg	398	3	4	129
auto-price	159	1	14	145
breast-tumor	286	8	1	23
compressive	1,030	0	8	845
concrete-slump	103	0	9	83
cpu	209	1	6	104
delta-ailerons	7,129	0	5	35
echo-month	130	3	6	53
forest-fires	517	2	10	251
housing	506	1	12	229
machine	209	0	6	116
pbc	418	7	11	4
pyrim	74	0	27	63
quake	2,178	0	3	12
sensory	576	11	0	11
servo	167	4	0	51
strike	625	1	5	358
triazines	186	0	60	102
winequality-white	4,898	0	11	7

In this chapter the first approach is evaluated. It works by adapting the heuristic to the regression task. The resulting algorithm is called SeCoReg (Separate-and-Conquer Regression). The second objective is evaluated in Chapter 7 where the regression problem is reduced to classification. There, the basic algorithm that is derived in Section 6.3 is used again. The reduction has the advantage that the heuristics known from classification can be re-used for the task of regression.

6.2 Regression Datasets, Regression Algorithms, and Experimental Setup

Table 6.1 displays the datasets used for tuning the heuristic parameter and the other parameters of the algorithm discussed in Section 6.3. Note that we proceeded in the same manner as described in Section 4.2 by optimizing the parameters on tuning datasets and then test their validity on hold-out validation datasets. However, the process was slightly different because additionally the tuning datasets

Table 6.2: Overview of the validation databases for regression

dataset	# instances	# nominal attributes	# numeric attributes	# distinct values
auto93	93	6	16	81
auto-horse	205	8	17	59
cloud	108	2	4	94
delta-elevators	9,517	0	6	26
meta	528	2	19	436
r_wpbc	194	0	32	94
stock	950	0	9	203
veteran	137	4	3	101
winequality-red	1,599	0	11	6

were split into two folds of equal size. The main reason to proceed in this manner is that two parameter settings are yielded then, one for each of the two folds. These two configurations can then be compared. Whenever the parameter values of them are to a large extent similar it can be concluded that a stable parametrization of the algorithm has been found.

The validation datasets are shown in Table 6.2. The majority of the datasets are taken from the UCI Repository [46]. Some of the datasets are available from Luís Torgo's webpage under `http://www.liaad.up.pt/~ltorgo/Regression/DataSets.html` (visited 2011-05-07).

There are four other algorithms that are all implemented in *Weka* [177] which were used to compare our system with. Clearly, some of them are much more complex than our rather simple algorithm[1]. The algorithms we used to compare our configurations with were a multilayer perceptron (MLP) [41, 72, 76], the SVMREG [154], a linear regression (Linear Reg.) using the Akaike-criterion for model selection [2], and M5RULES [78]. The first two algorithms employ rather complex models whereas the linear regression is a simple algorithm that nevertheless has a reasonable trade-off between runtime and performance.

The M5RULES algorithm, as a representative for rule-based algorithms, applies the separate-and-conquer technique and generates a tree in each iteration and derives regression rules from these model trees or (using the option `-R`) from regression trees. In the first case a linear model is predicted by each of the rules and in the second case they use a single value as prediction. Clearly, the first variant is able to adapt more closely to the dataset (cf. Figure 2.1 and the accompanied

[1] Note that the algorithm neither has a pruning functionality nor an optimization phase.

discussion) whereas the second (constant) version is in the same model class as our approach.

REGENDER [30] is a rather new rule learning algorithm that is based on an ensemble of rules. The algorithm uses a forward stagewise additive modeling to build an ensemble. The ensemble consists of decision rules. But, indeed, all rules that cover the example contribute to the final decision by calculating the sum of the individual prediction values of all covering rules. The algorithm relies on the *mean absolute error* and the *mean squared error* (cf. Section 2.7.3). For the optimization based on the two error measures (or losses), the gradient boosting [49, 50] and the least angle technique [39] is used. Note that the number of rules is a user-given value, so it has to be predefined. This makes the algorithm less flexible, because it is unclear how to determine the right number of rules in advance. We used it with ten and 100 rules and also with the same number of rules the derived rule learners have found on each dataset (cf. Section 6.5.3).

All of the algorithms used for comparison are implemented in *Weka* [177] and all their parameters were left at default values. Note that in the default configuration of REGENDER 50 rules are used in the *Weka*-version. Interestingly, the version recommended by its authors [30] uses 200 rules, a different loss function, and a different optimization technique, instead. The default configuration was used in the first experiments (cf. Table 6.8) whereas the proposed version was used later in Chapter 7.

The reasons to select the abovementioned algorithms were that our implementation had to prove that its performance is comparable to other state-of-the-art systems. Another reason to select these particular algorithms for benchmark was the lack of freely available regression rule learning algorithms. The only free system we found was REGENDER and a comparison to this algorithm is given later in this section.

During the experiments a 10–fold cross-validation was used as described in Section 2.7.1. The main means for measuring the performance of the algorithms was the *rrmse* as defined in Section 2.7.3. Additionally, where applicable, the Friedman Test with a post-hoc Nemenyi Test was employed (cf. Section 2.7.5).

6.3 A Direct Adaption of the SIMPLESECO Rule Learner to Regression

A straight-forward approach to adapt the SIMPLESECO algorithm to regression is to adapt the heuristic. Given that no coverage statistics are available in regression, we decided to utilize the measures usually applied in regression. Due to the positive results of the parametrized heuristics (cf. Section 4.2.3), we decided to design a novel heuristic that also trades off between consistency and coverage. To measure consistency the *relative root mean squared error (rrmse)* was used. It was combined with the *relative coverage* (cf. Section 2.7.3) yielding the heuristic h_{cm}.

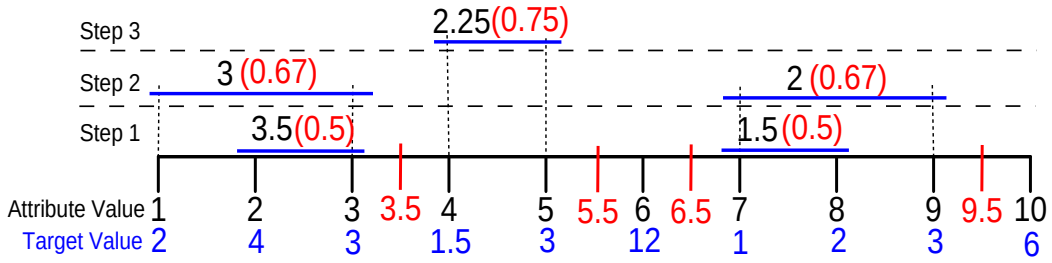


Figure 6.1: Example of the splitpoint clustering method

$$h_{cm} = \alpha \cdot (1 - rrmse) + (1 - \alpha) \cdot relcov \tag{6.1}$$

Here, the parameter α enables a trade-off between the error and the generality of the rule. For $\alpha = 1$ the relative coverage is ignored and thus the rules are solely evaluated by their error. This setting would yield a model that consists only of rules that cover a single example in the data and thus clearly would lead to overfitting[2]. The other extreme is to set $\alpha = 0$ which results in completely ignoring the error of the rule. A model built with this setting would only consist of the default rule, because its coverage is the highest that could be achieved by any rule. The optimal trade-off lies somewhere in between these two extremes.

Note that the heuristic h_{cm} is a direct adaption of the *relative cost measure* (cf. Section 2.6.3).

As described in Section 6.1 there are three adaptions necessary. Additionally, we derived a new method to compute the splitpoints for numerical attributes. In total, the algorithm has three parameters:

- The parameter of the heuristic,
- the number of splitpoints (*splitpoint*-parameter), and
- the percentage of examples that are left uncovered (*left-out*-parameter).

The parameter of the heuristic is optimized with the same greedy procedure known from Section 4.2.1. The number of splitpoints is crucial for the runtime of the algorithm. This value was optimized by testing different values. The last user-given parameter is the percentage of examples that are left uncovered by the outer loop of the algorithm. This parameter clearly depends on the dataset. If, e.g., half of the dataset is created by a certain function and the rest of the dataset encodes randomness, the best setting would be to learn rules for the part of the dataset that

[2] Note that this holds only in a scenario where a rule is allowed to cover a single example.

Algorithm 6.1 COMPUTESPLITPOINTS*(Examples, desiredSplitpoints)*

```
# initialize bestCluster and clusters
bestCluster ← null
clusters ← INITIALIZECLUSTERS (Examples)
# loop until desired number of clusters is found
while |clusters| ≥ desiredSplitpoints+1 do
    # build candidate clusters
    candidateClusters ← MERGEALLADJACENTCLUSTERS (clusters)
    # determine best one
    bestCluster ← LOWESTERROR (candidateClusters)
    # remove the two clusters the best one was build from
    clusters \ CLUSTERSUSEDIN (bestCluster)
    # add best cluster to clusters
    clusters ∪ bestCluster
# return splitpoint values (mean between adjacent clusters)
return SPLITPOINTS (clusters)
```

was created following some specification whereas the other part of the data would be described best by using the median over these examples.

Interestingly, during the experiments there was some evidence that we had included databases that basically encode randomness and for those learning anything results in bad performance (e.g., the dataset `quake`). On those datasets the best model will be given by one that simply predicts the median over all examples as the default rule does. For the sake of performance, the parameters of the algorithm could not be tuned on each single dataset but were tuned to yield an acceptable compromise for all datasets. In effect, a rule set is actually learned for the dataset `quake` where instead the prediction of the median over all examples would have been the better choice.

6.3.1 Splitpoint Processing

In rule learning algorithms, the generation of all possible splitpoints would be too costly. In classification, usually only the splitpoints between changes of the class (either from positive to negative or the other way round) are used. For regression a method for restricting the splitpoints for an attribute was developed. The basic idea comes from supervised clustering. Thus, we try to identify regions in the data of the current attribute that share a small error with respect to the target variable. The aim of the bottom-up clustering procedure is to yield partitions of the attribute that share a low error in the hope that the error of a rule that covers these regions will also be low. Clustering stems from the same motivation because it also guarantees that each cluster has the lowest possible error. Somehow this

is related to the mechanism known from classification. In this case, the dataset is partitioned into regions with the same target attribute (either positive or negative examples), because a splitpoint in between such a region would be suboptimal due to its bad discriminatory power.

Similarly to the problem of determining the best number of bags for a discretization of a numerical attribute or the number of binary classifiers (cf. Section 6.6), it is not simple to decide how many clusters should be used for the splitpoint clustering. The user has to predefine this number and hence how many splitpoints should be used. We experimented with different settings but surprisingly a rather low number of splitpoints seemed to be sufficient (cf. Section 6.4.1).

Figure 6.1 displays how the cluster algorithm (cf. Algorithm 6.1) works. In the example in Figure 6.1 the attribute has ten values moving equidistant from one to ten. The blue values depicted at the bottom line of the figure are those of the target attribute of the respective example. In the first step the attribute values are ordered ascending and each value becomes a cluster C_i containing exactly this value. Note that the subscript of C denotes its attribute value(s). Then two adjacent clusters are searched for which the error when using the mean of the two target values as prediction is the lowest. In the example these are the clusters two and three, seven and eight, and eight and nine. Though the objective in the first step is to join two adjacent clusters both C_2, C_3 (yielding C_{23}) and C_7, C_8 (resulting in C_{78}) are joined (its arbitrary whether to join C_7 and C_8 or C_8 and C_9). The mean of the first cluster is $3.5 = {}^{(4+3)}/_2$ and C_{78} has a mean of 1.5 (depicted above the line that connects the two values in Figure 6.1). If the mean absolute error is taken, both clusters have an error of 0.5, which is shown in red brackets in the corresponding figure. An error of $0.5 = {}^{(|4-3.5|+|3-3.5|)}/_2$ is also the lowest error that can be achieved given the exemplary data.

In the second step the procedure is executed recursively and again those clusters are joined that have the lowest error among all possible clusters. So, in this step, cluster C_1 is joined with C_{23} (yielding C_{123}) and cluster C_9 is joined with C_{78}. The error of both clusters grows to 0.67 because adding the respective example raises the error (i.e., $mae = {}^{(|2-3|+|4-3|+|3-3|)}/_3 = 0.67$ for the first cluster). Joining any of the untouched clusters leads to a higher error which means that the cluster with next lowest error is built in step three. After the second step six clusters were built and therefore five splitpoints exist. In the example the user given number (*desiredSplitpoints*) is set to four. Hence another cluster has to be built until the algorithm is finished. This last cluster is derived by joining C_4 and C_5. The resulting cluster C_{45} has an error of 0.75.

After the third step five clusters are built and the splitpoints are simply derived by taking the mean between the values of two adjacent clusters or two values if the cluster contains only one example. The four splitpoints are 3.5, 5.5, 6.5, and 9.5 (depicted between two adjacent values in Figure 6.1 at the number ray).

We have evaluated the effectiveness of the clustering method by comparing it to another splitpoint method where a user-given number of splitpoints is selected equidistant. The results of this comparison are shown in Section 6.5.1.

For the computation of the error in procedure LOWESTERROR the *mean absolute error* was used. This choice is arbitrary but experiments with the *root mean squared error* did not yield any performance difference.

6.4 Optimizing Several Parameters

For the optimization of the three parameters, we proceeded in the same way as in Section 4.2 by using the tuning datasets displayed in Table 6.1 to optimize all parameters. The difference here is that for all parameter optimizations the datasets were split into two equal sized folds. Therefor, all datasets were randomized in advance using the unsupervised randomize function[3] of *Weka* [177]. All evaluation measures were computed using a 10–fold cross-validation. The parameters were optimized on the first subsets (S_1) and afterwards the algorithm was evaluated on the second subsets (S_2) of all datasets and vice versa which yields two configurations of the algorithm, i.e., two different setups of all parameters. If these two configurations are more or less equal, this would be solid evidence that the parameters are stable among many different learning problems. After the tuning, the two configurations are additionally evaluated on the nine validation datasets shown in Table 6.2.

There are many proposals how to optimize a set of parameters (for an overview covering nearly all optimization-related details see [171]). These include grid-search or evolutionary tuning. We decided to use a simpler method. Under the assumption that the parameter of the heuristic has the biggest impact on the performance, the other two were tuned first. This assumption seems to be justified as all candidate rules are evaluated by the heuristic and the effects a change of the parameter would have are huge. The *left-out*-parameter was fixed to zero, thus all examples are covered. In this case the default rule is built by using the median of all examples. In other cases, where examples remain uncovered, it is built by using the median of all uncovered examples.

In contrast, it is not obvious what parameter value can be used for the heuristic. For this reason, five different values were used during the optimization. To make a choice, the two extremes were included ($\alpha = 0$, and $\alpha = 1$), and some values in between, namely 0.4, 0.5, and 0.6. These values were chosen to include different preferences of the heuristic. Clearly, using only two parameters would be suboptimal because in Section 4.2.2 we have seen that the optimal parameter rather lies somewhere in the middle of the domain than at the beginning or end of it. We expected the curve yielded by plotting the parameter over the error to be shaped

[3] `weka.filters.unsupervised.instance.Randomize`

Table 6.3: Results for the splitpoint computation and left-out-parameter (average *rrmse* over the five parametrizations of the heuristic, S_1 stands for the first subsets and S_2 for the second, respectively)

parameter (splitpoint)	S_1	S_2
1	1.0675	1.0540
3	**0.9929**	1.0256
5	1.0132	1.0261
7	1.0067	1.0245
9	0.9992	**1.0209**
11	1.0126	1.0427
19	1.0163	1.0240

parameter (left-out)	S_1	S_2
0	0.9929	1.0209
0.01	0.9787	1.0221
0.02	0.9776	1.0182
0.03	0.9759	1.0156
0.05	0.9739	0.9940
0.1	**0.9704**	0.9835
0.2	0.9736	**0.9701**

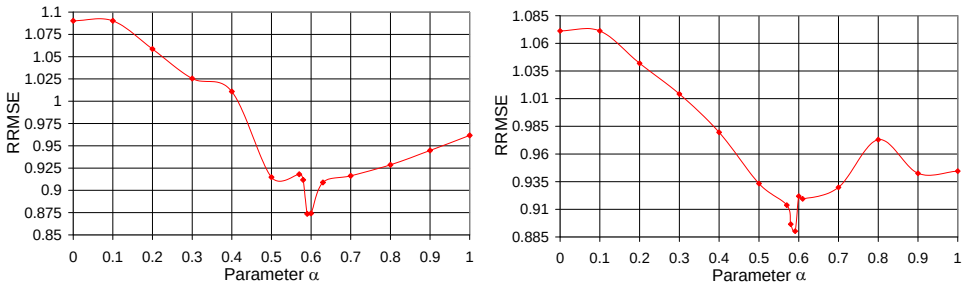
like a U, where the two extreme values would results in a rather bad performance and the optimal value lies somewhere around 0.5. Note that the shape is turned upside down now because smaller values are better (*accuracy* vs. *error*). To have a combined error measurement for the optimization procedure the mean of the *rrmse* of these choices, estimated by the cross-validation, was taken.

6.4.1 Splitpoint and Left-Out-Parameter

In the beginning, the *left-out*-parameter was fixed to a value of zero yielding a starting point for the optimization of the *splitpoint*-parameter. To find the best value some intuitive values (1, 3, 5, 7, 9, 11, and 19) were used. All values bigger than 19 were skipped because a clear gain in runtime performance should be achieved. Using huge values would result in practically using all possible splitpoints and thus would not improve the algorithm's runtime[4].

Table 6.3 (left table) shows the results for the optimization procedure (for the two subsets S_1 and S_2 of the tuning datasets). As can be seen the best number of splitpoints was three on the first subset and nine on the second subset (the lowest error is depicted in bold in the figure). On S_1, however, using nine splitpoints yields the second best *rrmse* which lacks only 0.0063 behind the best performing number of splitpoints. On S_2, using nine splitpoints performed best followed by using 19 splitpoints. The best performing number of three from the first subsets, lacks 0.0047 in terms of *rrmse* behind the best one and therefore is the third best method. Nevertheless, the gap between different parametrizations seem to be big-

[4] Note that the number of disjunct values for an attribute in the data is rather small as can be seen in Table 6.1 in the rightmost column.



(a) tuned on S_1 (b) tuned on S_2

Figure 6.2: Parameters over *rrmse* for both subsets of the tuning datasets

ger on S_1 than on the second subsets. Regarding the split of all tuning datasets into two folds of equal size, these results seem to reflect the randomness in splitting the datasets.

After the optimization of the splitpoint parameter, the same procedure was employed to the *left-out*-parameter of the algorithm. Here, the splitpoints were already fixed to three for the algorithm tuned on S_1 and to nine for the variant tuned on S_2. To find the best value again some intuitive parameters were used. Thus, the values 0, 0.01, 0.02, 0.03, 0.05, 0.1, and 0.2 were evaluated during this optimization. To cover all examples was included to make sure that it is more effective to leave some examples uncovered. Clearly, this depends on the given dataset. But it also is affected by the quality of the induced rules. For numerical target variables it can be useful to cover only those parts of the data that share some common characteristics. For the rest of the data it could be beneficial to treat them independently from their characteristics, i.e., by assigning them the same target value. Values bigger than 0.2 are not a good choice, because leaving more than 80% of the examples uncovered results in suboptimal rule set.

As can be seen in Table 6.3 (right table) two different parameters performed best on the two subsets. Practically this can be attributed to the same reasons that were already discussed during the optimization of the splitpoint parameter. Thus, on the one hand the randomized split of the data into two subsets of equal size could have manipulated the characteristics of the datasets (i.e., the distribution of the target attribute). On the other hand it could also be possible that there is no unique best value for leaving examples uncovered. However, the results clearly show that leaving examples uncovered is mandatory for a good performance of the algorithm.

Table 6.4: Best configurations of the SECOREG learner

identifier	tuned on	parameter of h_{cm}	splitpoint parameter	*left-out* parameter
SECOREGS_1	S_1	0.59	3	0.1
SECOREGS_2	S_2	0.591	9	0.2

6.4.2 Parameter of the Regression Heuristic

For the optimization of the heuristic parameter the search procedure introduced in Section 4.2 was used. It employs a binary search to find the best parameter and proved to yield stable ones for classification heuristics as observed in Section 4.2.2.

The search is started with a range of intuitively appealing values. Thus, the two extremes of zero and one are evaluated together with some values in between $(0.1, 0.2, ..., 0.9)$. All settings are evaluated by taking the average of the *rrmse* on the 20 datasets presented in Section 6.2. Then the best performing parameter is used for further inspection. Therefore, an area around this value is inspected in more detail. There are several choices to do this, but we decided to evaluate six parameters around the best one. Those are distributed equidistant around the best parameter with decreasing the step size from 0.1 to 0.01. This procedure is executed recursively, so in the next step the six parameters around the next best value are evaluated. The search stops if the *rrmse* improvement falls below a threshold of $t = 0.0005$. This choice was arbitrary but we believe that the effort that has to be made to narrow down the parameter in the next step of the search procedure is too high compared to the performance gain this execution may yield.

Figure 6.2 shows a graphical interpretation of the search for both experiments. For either of them low parameter settings result in suboptimal performance. When the parameters are increased the performance becomes better as long as the optimal setting is reached. After that it decreases again.

For the parameters that are optimized on the first subsets of the datasets (Figure 6.2 (a)) the curve shows some fluctuations in the part located left of the best parameter. In spite of this behavior the curve depicted in Figure 6.2 (b) is monotonically decreasing in this area. For parameter settings that are bigger than the best parameter the curve in the left figure is now showing a monotone increase whereas it shows more fluctuations when the parameter is tuned on the second subsets of the partitioned datasets.

Interestingly, the best parameters are very similar in both experiments. This means that they are stable among different subsets of the datasets. On S_1 the best parameter lies at 0.59 and on the second subsets it was 0.591. For the first

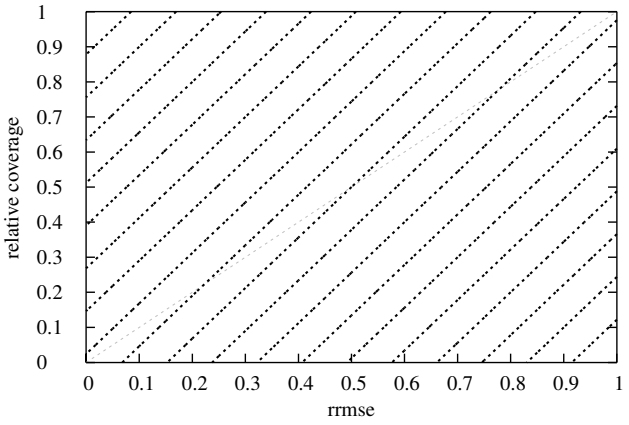


Figure 6.3: Isometrics of the regression heuristic for $\alpha = 0.59$

subsets the optimum of 0.591 lacks only 0.007 behind in terms of *rrmse*. For S_2 the difference in performance was 0.001.

In the end, the optimization yields two configurations of the algorithm which are summarized in Table 6.4. The configurations are abbreviated with SECOREGS$_1$ for the setting derived on S_1 and SECOREGS$_2$ for the values found on S_2 respectively.

Assumed that the best parameter lies somewhere around the region of 0.6, consistency should be preferred over coverage for regression rules. This also holds for classification rules where the preference of consistency is even stronger than in regression. This can be observed for the *cost measure* and the *relative cost measure* (cf. Section 4.2.2) where the best parameter was 0.437 for the first heuristic and 0.342 for the second one. For the *cost measure* the consistency was only slightly preferred over the coverage, but for the *relative cost measure* it was even more heavily weighted. Nevertheless, both cost measures for classification show a clear trend of preferring consistency over coverage which is also notable for the regression heuristic.

Figure 6.3 displays the isometrics in a space where the *rrmse* is displayed on the x-axis and the relative coverage is shown on the y-axis. Note that the graph is plotted in the same aspect ratio as before albeit it actually is a quadratic one (both axis moving from zero to one). This format was chosen to ease a comparison to the isometrics of the two classification cost measures (isometrics of the *relative cost measure* are given in Figure 4.3 (d)). The isometrics show the same preferences as are observable for the *relative cost measure*. This is a reminiscent of the fact that the same basic heuristics are traded off. Interestingly, the slope of the parallel isometrics is also quite similar compared to the *relative cost measure*. The reason is that the parameter setting of both of them is nearly the same (0.342 for the *relative cost measure* and $1 - 0.591 = 0.409$ for the regression heuristic h_{cm}). Nevertheless,

Table 6.5: Runtime of different splitpoint methods on the test set

method	runtime (in sec.)
3 equidistant splitpoints	2,625.4
3 clustered splitpoints	1,234.3

the number of positive examples is not exactly the same as the relative coverage because the latter measures total coverage whereas the first only computes the positive coverage. The number of negative examples also does not correspond exactly to the *rrmse*.

6.5 Results

In the previous sections, the tuning of the different parameters was described. In the following, some runtime statistics of the algorithm are given when it is used with various numbers of splitpoints. Then the two configurations are compared against other well-known algorithms.

6.5.1 Using Different Numbers of Splitpoints

Table 6.5 shows a comparison of the runtime of two different splitpoint methods. At first, three equidistant splitpoints per attribute were used. Then, three clustered splitpoints were employed. Evaluating all splitpoints was too costly. All runtimes depicted in Table 6.5 are the averages of ten independent runs on a dual Pentium 4, 2.8 GHz processor with 2 GB RAM on the nine datasets used for testing (cf. Section 6.2).

As can be seen in Table 6.5 the clustered splitpoint computation is more efficient than the equidistant method. At first sight this may appear contrarily to what could be expected. Due to the much more simpler computation of equidistant splitpoints this method should be faster than the clustering method. But note that this evaluation was done by letting the entire algorithm run on the nine test datasets. Not surprisingly the quality of the equidistant splitpoints is worse compared to the clustered splitpoints. This results in a notably higher number of candidate rules that have to be evaluated during the search which can be drastically reduced by using clustered splitpoints as the best rule is found much faster.

Table 6.6: Results in terms of average *rrmse* for different algorithms on the two folds of the tuning datasets (S_1 and S_2)

algorithm	S_1	S_2
M5Rules	0.7425	0.8058
Linear Reg.	0.8145	0.9116
MLP	1.0154	1.389
SVMreg	0.7917	0.85
SECOREGS$_1$	*0.8736*	0.9291
SECOREGS$_2$	0.8976	*0.8903*

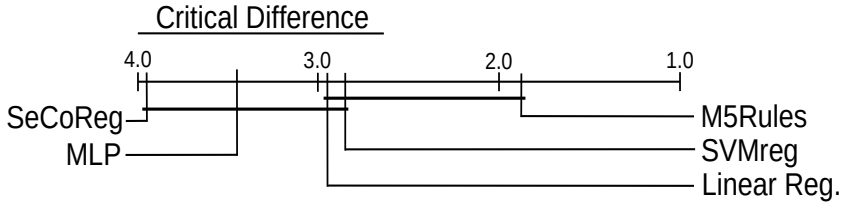
6.5.2 Comparison with other Systems on the Tuning Datasets

The main reason for the comparison on the tuning datasets is to determine how good the algorithms perform against other regression algorithms both on the subsets used for tuning and on the hold-out subsets. Table 6.6 gives an overview of the different algorithms compared to each other on the two subsets using the *rrmse*. The configurations were compared against the four other regression algorithms described in Section 6.2.

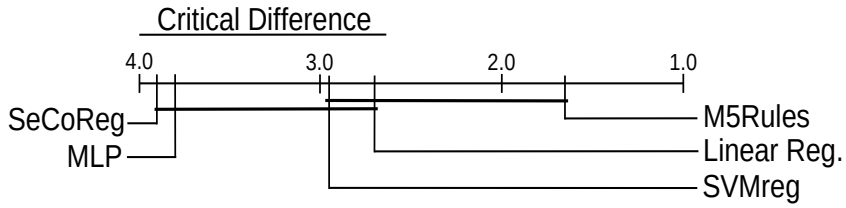
In Table 6.6 the results of all algorithms on the two subsets of the tuning datasets are displayed. Results of both derived SECOREG-algorithms are shown together with their performance on the datasets on which they were tuned (in italics). Not surprisingly both variants of the algorithm that were tuned on the respective subsets perform better than using them on the other subsets. The ranking of the algorithms is similar on both experimental variants. The best one was the M5RULES algorithm followed by the SVMREG and the linear regression. The SECOREG was ranked on the fourth place in both experiments, only slightly behind the linear regression (lacking 0.0831 behind on the first subsets and 0.0175 on S_2). The multilayer perceptron had the worst performance with a rather big gap to the next better algorithm.

Figure 6.4 shows CD-charts (cf. Section 2.7.5) for both experiments. Only M5RULES was significantly better than the SECOREG algorithms in both cases.

In summary, the performance of the two SECOREG-configurations is solid without beeing really convincing. The configurations are worse than most of the other algorithms and even do not outperform the linear regression. The reasons could be that it is hard in general to improve over the given selection of algorithms given the restrictions of our rule learner. The linear regression, as we will see later, has a good performance in general (cf. Section 7.3). M5RULES uses linear models as prediction which makes the algorithm much more flexible because all examples



(a) subsets S_1



(b) subsets S_2

Figure 6.4: Comparison of all algorithms against each other with the Nemenyi test on the two subsets. Groups of algorithms that are not significantly different (at $p = 0.05$) are connected.

that are covered by a rule are mapped to several different values instead of a single one. Nevertheless, there is a considerable high number of datasets (six of 22) where no rule is learned at all. In these cases the model of the M5RULES-algorithm is similar to the output of the linear regression. Hence, it is questionable whether we still have a rule-based model as it does not contain a rule at all. In general, a rule learner that uses linear models in the head of the rules can never be worse than the linear regression.

In addition, it is surprising that the performance of the SECOREG-configurations on the subsets where they were tuned also is not overwhelming. The difference between the error of the two algorithms among the two subsets is 0.024 for S_1 and 0.0388 for S_2. It is obvious that this difference indeed is not that large compared to the magnitude of the *rrmse* of all algorithms. The picture changes slightly when we evaluate the algorithms on the hold-out test sets as can be seen in the next section.

Note that some of the algorithms show a *rrmse* greater than one. Effectively, this means that they are worse than a random prediction[5]. Usually this situation occurs whenever the dataset encodes randomness, i.e., when learning is not beneficial at all. Due to the split into two folds of equal size and the cross-validation that is employed afterwards, some of the folds of the cross-validation may become very small and also rather skewed concerning the distribution of target values compared

[5] A random prediction in the case of regression is a model that uses the median over all training examples as prediction.

Table 6.7: Results in terms of *rrmse* for different *Weka* algorithms on the test set

dataset	SVMreg	M5Rules	Linear Reg.	MLP	SECOREGS$_1$	SECOREGS$_2$
auto-horse	0.32 ±0.08	0.37 ±0.14	0.32 ±0.11	0.34 ±0.10	0.52 ±0.18	0.61 ±0.11
auto93	0.66 ±0.12	0.58 ±0.19	0.67 ±0.20	0.57 ±0.19	0.65 ±0.17	0.85 ±0.29
cloud	0.39 ±0.12	0.42 ±0.16	0.40 ±0.13	0.62 ±0.33	0.61 ±0.19	0.67 ±0.15
delta-elevators	0.61 ±0.01	0.60 ±0.01	0.61 ±0.01	0.63 ±0.01	0.78 ±0.03	0.77 ±0.03
meta	0.92 ±0.08	1.86 ±1.58	2.33 ±1.72	1.40 ±0.90	1.00 ±0.02	1.01 ±0.03
r_wpbc	1.03 ±0.16	1.14 ±0.19	1.04 ±0.13	2.20 ±0.56	1.35 ±0.20	1.27 ±0.18
stock	0.37 ±0.05	0.14 ±0.03	0.36 ±0.04	0.20 ±0.04	0.25 ±0.03	0.26 ±0.04
veteran	0.93 ±0.15	1.23 ±0.61	1.07 ±0.36	3.01 ±1.78	1.09 ±0.22	1.21 ±0.33
wine-quality-red	0.82 ±0.03	0.81 ±0.03	0.81 ±0.03	0.95 ±0.08	0.98 ±0.09	0.95 ±0.04
averages	0.6739	0.7942	0.8456	1.1017	0.8040	0.8438

to the distribution in the whole dataset. Also, in regression there is no stratified cross-validation (cf. Section 2.7.1), i.e., the distribution of values is purely random which may also be the reason that a model is learned that is worse than random prediction.

6.5.3 Comparison with other Algorithms on the Test Sets

To validate the results on completely different datasets, the algorithm was also tested on nine independent test sets (cf. Section 6.2). This step is necessary to make sure that the tuning datasets, even though they were split into two disjunct subsets, were not overfitted during the parameter tuning phase. Table 6.7 displays the results in terms of *rrmse* on the test databases for all of the four *Weka* algorithms and the two configurations of the SECOREG-learner. The ranking of the algorithms slightly differs compared to the results on the 20 datasets. Hence, on the test sets the SVMREG performs best followed by the M5RULES-system. On the third place SECOREGS$_1$ appears. It was only slightly worse in performance compared to M5RULES. The next best algorithm is SECOREGS$_2$ which has achieved a marginal better *rrmse* than the linear regression. As in the previous experiments the multilayer perceptron was the worst algorithm.

Thus, on the test sets the tuned SECOREG-algorithm achieved better results than in the previous experiment. Here, the best configuration of the algorithm is ranked on the third place. Note that the dataset `meta` shows huge standard deviations for some algorithms (M5RULES, linear regression and MLP) which also contributes

Table 6.8: Results in terms of *rrmse* compared to RegENDER on a subset of seven datasets of the test set

algorithm	avg. rrmse	avg. rank
SeCoRegS_1	0.82	3.0
SeCoRegS_2	0.85	3.13
RegENDER (10 rules)	0.90	3.88
RegENDER (100 rules)	0.93	3.88
RegENDER (rules from S_1)	0.92	3.75
RegENDER (rules from S_2)	0.90	3.38

highly to the averaged results. We attribute this to the separation of the data into the ten folds of the cross-validation. However, all algorithms had to deal with this situation.

Additionally to the error measurements a Friedman-Test was employed. Contrarily to the previous results, the Friedman-Test was not rejected at a p-value of 0.05 (the critical F-value was 2.196 but to be rejected it had to be bigger than 2.492). It would have be rejected at a p-level of 0.1, but this was not significant enough to include these results in the chapter. For this reason the Nemenyi-Test could also not be done on the test sets. Practically, at the confidence level of $p = 0.05$, the algorithms do not differ significantly.

Table 6.8 shows a comparison to RegENDER [30]. The dataset `auto-horse` contains missing class values which cannot be handled by RegENDER. Therefore, this dataset was skipped. In addition the results on the dataset `meta` showed strong fluctuations as mentioned before. For this reason this dataset was also not included. RegENDER has a parameter to specify the number of rules in the ensemble. To make a choice the algorithm was tested with ten and 100 rules and with the same number of rules the two SeCoReg variants had found on the test sets[6]. Clearly, using more rules will result in a lower error but, for reasons of comparison, we think it is fair to run the algorithm with the same number of rules as used in the SeCoReg-learner.

The SeCoReg-algorithms were slightly better in average *rrmse* and their average ranks were also superior. Hence, a Friedman Test was rejected ($p = 0.05$) but the Nemenyi Test showed that all algorithms were in the same equivalence class (the critical distance extends over the average ranks of all algorithms) and therefore they do not differ statistically significant. In a later experiment, the best configuration of RegENDER that was proposed by the authors [30] was also included in a comparison, albeit on different datasets (cf. Section 7.4).

[6] Note that we used the default version of RegENDER instead of the one proposed by the authors.

6.6 Related Work

There are only a few previously published attempts that try to adopt separate-and-conquer rule learning to regression. Examples include the FRS system [31], which is a reimplementation of the earlier FORS algorithm [94], and *predictive clustering rules* (PCR) [168]. Predictive clustering rules are generated by modifying the search heuristic of CN2 [20] to use a heuristic that is based on the dispersion of the data. This algorithm also follows a different route by joining clustering approaches with predictive learning. The R^2 system [163] works to some extent analogously as other separate-and-conquer algorithms by selecting an uncovered region of the input data. But this selection differs from the mechanism used in regular separate-and-conquer learning. However, it also allows for rules to overlap and the rules predict linear models instead of a single target value.

More popular are techniques that learn regression trees [7, 135, 169], which can then be converted into decision rules as in the M5RULES algorithm [78]. The key idea of these approaches is also to replace the purity heuristic of the decision tree algorithm with a heuristic that measures the reduction in variance.

Other rule-based regression algorithms are based on ensemble techniques. For example, RULEFIT [51] performs a gradient descent optimization, allows the rules to overlap, and the final prediction is calculated by the sum of all predicted values of the covering rules.

A related algorithm is described by Steger [155]. Here, also an ensemble of decision rules is learned with the difference that the right number of rules is determined by the algorithm itself. The procedure of classification is similar to the one described for REGENDER. The learning, instead, is different. The algorithm uses the I-REP strategy by dividing the training set into a growing and pruning set in the same matter as RIPPER proceeds (cf. Section 3.5.3). Then, rules are learned on the growing set and already during the learning also pruned on the pruning set. The examples that are covered by a rule are not removed, but their target value is reduced by the value of the prediction of the found rule in order to reduce the variance of the whole ensemble. The evaluation of the candidate refinements of a single rule is done on the pruning set, which can be seen as an implicit pruning mechanism because a rule only grows if its evaluation on the pruning set is sufficient. The *mean squared error* was used to evaluate the rules, a rule has to cover at least three examples, and the induction of rules stops whenever the overall performance of the next five rules the algorithm would find does not increase. If so, the rules are discarded and not included in the rule set. All parameter choices are results of a solid experimental setup. For further information see [155].

Another popular technique to deal with a continuous target attribute, different from all the above ones, is to discretize the numeric target values as a preprocessing step and apply conventional classification algorithms on the discretized

dataset. As this can be seen as reducing regression to classification, work following this path is discussed later in Section 7.5.

6.7 Summary

To summarize the chapter, a new rule learning algorithm for the task of regression was presented. The modifications of the original classification algorithm to make it suitable for regression were shown. These include a new splitpoint generation method, a stopping criterion, and an optimal setting for a new parametrized heuristic for regression. It can be concluded that both tuned variants of the presented algorithm are unable to outperform state-of-the-art systems. They are rather situated in the middle of the other algorithms (this holds at least for the test sets). Especially on the nine test sets it became clear that the SECOREG rule learners are able to achieve a performance comparable to the results of the four *Weka* algorithms and REGENDER. Due to the rather simple design of the SECOREG learner, the algorithm can be seen as a first step towards effective regression rule learners that predict single target values and are based on decision lists.

7 Regression via Dynamic Reduction to Classification

The goal of this chapter is the design of a heuristic rule learning algorithm that learns regression models in the form of a decision list consisting of simple regression rules that have constant models in the rule head just as before. But here, we do not adapt the heuristic, but instead we introduce a method that enables the use of classification heuristics. One of the benefits of this approach is that we are able to use the heuristics presented in Chapter 2 and 4. In this sense, our hope is that the good results of the parametrized heuristics carry over when they are used in the regression rule learner.

Again, it should be mentioned that the chosen model class, i.e., using a decision list with constant values in the rule head, is fairly restrictive. However, we can show that our approach yields a good performance within this model class, even a better quality than the algorithms presented in Chapter 6. Nevertheless, statistical methods that are not restricted in any way still work better.

Several strategies to induce a set of regression rules have been proposed in the literature (cf. Section 6.6). Some of them rely on the gradient-descent algorithm for finding a rule ensemble that optimizes some loss function. Others convert given trees into sets of rules. One of the most popular strategies for learning classification rules is the separate-and-conquer algorithm as shown in Algorithm 3.1 and 3.2 and as discussed also in Section 3.1, because, due to its simplicity and its good performance in classification (cf. Chapter 4), we decided to use this strategy as the basis for our approach. As previously described in Chapter 6, we adapt the algorithm by using constant values in the head of the rule (instead of a nominal value as in classification) and by defining a stopping criterion when to prevent the addition of subsequent rules (cf. Section 6.1).

In Section 7.1, we introduce our approach for repeatedly converting regression problems into classification problems. Its key idea is to dynamically define a region around the target value predicted by the rule, and considering all examples within that region as positive and all examples outside that region as negative. The experimental setup of our approach and the key results are described in Sections 7.2 and 7.3, respectively. In Section 7.4 the approach is compared to the two SeCoReg learners derived in the previous chapter. The following section comprises related work with a focus on reduction schemes and Section 7.6 concludes the chapter by providing a summary.

Note that the algorithm for reducing the regression problem to classification was published in the paper [90].

7.1 Dynamic Reduction to Classification

As mentioned in Chapter 6, there are two principal approaches for adapting heuristic rule learning algorithms to numerical target variables. One possibility is to change the heuristic function in a way that the goal of discriminating between positive and negative examples is replaced with the goal of reducing the variance in the target value. This approach has been taken by most decision-tree based approaches [7, 135, 169], by several regression rule learning approaches [94, 163], and it formed also the basis of the metric-based approach discussed in the previous chapter (cf. Section 6.3).

An alternative approach is to derive the necessary statistics, i.e., p and n, the number of positive and negative examples that are covered by the rule, and P and N, the total number of positive and negative examples in the dataset, from numerical data, so that heuristics that are known to work well for classification can be directly used on regression problems. A simple strategy that employs this approach is to discretize the target value using equal-width, equal-frequency, or a search-based approach [164].

However, the use of class-discretization as a pre-processing step makes it harder to fine-tune the class values to the context provided by the examples, just as a pre-discretization of attributes is less flexible than the dynamic discretization that is commonly used in rule learning or decision-tree algorithms. Therefore, we propose an alternative approach that allows to dynamically derive positive and negative examples from regression data.

Each (complete or incomplete) rule r is associated with a numerical target value y_r which will be predicted for all examples that are covered by this rule. Obvious candidates for determining y_r are the mean and the median of the covered examples. We chose the median because it is more robust against outliers and proved to be a good choice in Chapter 6. Our goal is that the covered examples are close to this predicted value. In fact, this is the key motivation behind all approaches that evaluate candidate conditions by the reduction of the variance among the covered examples. We implement this idea in a different way by not directly using the standard deviation as a measure for the rule quality, but by labeling all examples that are within the standard deviation as positive, and all examples that are outside the standard deviation as negative. This allows to evaluate the rule using standard classification heuristics such as those defined in Section 2.6 for guiding the learner. More precisely, if the distance between the target value and the predicted value is below a certain threshold, this example is considered to be positive and if it is above the threshold it would be a negative example, i.e.,

$$class(\mathbf{x}) = \begin{cases} positive & \text{if } |y - y_r| \leq t_r \\ negative & \text{if } |y - y_r| > t_r \end{cases} \tag{7.1}$$

where $\mathbf{x}$ is the current example, y is the true value of the example $\mathbf{x}$, y_r is the value predicted by rule r, and t_r is a threshold. Thus, the total number of positive and negative examples is defined as

$$P_r = \sum_{i=1}^{t} \mathbf{1}(|y_i - y_r| \leq t_r); \quad N_r = t - P_r \tag{7.2}$$

where t = number of examples, and $\mathbf{1}(.)$ is the indicator function.

All possible conditions g are then evaluated by forming the refined rule $r' = r \cup g$ and counting how many positive ($p_{r'}$) and negative ($n_{r'}$) examples are covered by r'. Among them, the condition that maximizes a given rule learning heuristic is chosen.

Note that all counts are indexed with the rule r. This is because both the predicted value y_r and the threshold t_r may change with each refinement step. As a consequence, the labels of the examples and thus any of the above counts may change after r is replaced with r'. Note, however, that the positive examples provide a focus towards the median of a rule, which makes it more likely that the label of examples changes from positive to negative than in the opposite direction.

One can think of different ways for defining the threshold t_r. As motivated above, the standard deviation σ_r of the examples that are currently covered by the rule is a natural choice. We also experimented with a factor that slightly reduces or enlarges the standard deviation. Section 7.3 shows the results for the following three thresholds

- $t_r = 0.95 \cdot \sigma_r$,
- $t_r = \sigma_r$, and
- $t_r = 1.05 \cdot \sigma_r$.

We also considered asymmetric distributions where we have two different thresholds, one for $y_r < y$, and one for $y_r > y$, but this did not improve the results.

The key idea of the reduction approach is reminiscent of *ϵ-insensitive* loss functions which form the basis of several support-vector regression algorithms [154]. A key difference is that we adapt the width of the insensitive region dynamically after each refinement step.

As in the classification setting, the refinement of a candidate rule continues until no more negative examples are covered, and the best rule encountered in this

process is returned as the final rule. Depending on the selected heuristic, this is not necessarily the last rule (cf. the refinement graph given in Figure 3.1).

Finally, we have to define a stopping criterion that stops the induction of rules in general. A naive method to do so is to stop the induction of additional rules when a certain amount of examples is covered by rules. We decided to stop learning rules as soon as 90% or more of the examples are covered by rules. While we did not yet perform a thorough evaluation of this parameter, it was chosen according to our experience with the metric-based regression rule learner (cf. Section 6.4.1). All remaining examples will be covered by a default rule, which will predict the median value of all uncovered training examples.

7.2 Experimental Setup

In the following, we compare our algorithm to a variety of other algorithms, both state-of-the-art regression rule learners and statistical regression learners, all in their implementations in the *Weka* data mining library [177]. The rule learning algorithms include M5RULES (also using the option -R), and REGENDER [30]. Note that we used the version recommended by the authors [30] (cf. Section 6.2). A description of the used algorithms can also be found in Section 6.2. As representatives for statistical regression algorithms we used a linear regression, a *multi-layer perceptron* (MLP), and *support-vector regression* (SVMREG) in the same setup as before.

In addition, we also compare our dynamic approach to regression by classification to a static approach. Here, the numerical target variable of a regression problem is transformed into a nominal class. This process of discretizing the target variable and afterwards use standard classification algorithms can also be seen as a reduction. Research following this path can be found in [164, 173, 174]. The main problem here is that the number of bags for the discretization process is unknown in advance. For this reason the performance of this technique strongly depends on the choice of the number of classes. To compare the reduction approach with the static discretization, we used the standard method implemented in *Weka* using the class `weka.classifiers.meta.RegressionByDiscretization`. However, we used an equal-frequency instead of the default equal-width discretization of the class variable, because the former seemed to work better. As a learning algorithm we used the same rule learning algorithm upon which our dynamic approach is based, namely the SIMPLESECO described in Section 3.5.4, with the same four learning heuristics described below.

As noted before there are many different heuristics to navigate the search. All of them are trying to maximize the coverage of positive examples (p) and to minimize the negative coverage (n), but differ in the way in which they trade off these two objectives [60]. Experimenting with all known heuristics was out of the scope for this chapter, so the decision was to use a heuristic that is known to underfit the

Table 7.1: Evaluation of dynamic regression by classification (top), static regression by classification (bottom), and two other rule-based learning algorithms.

Dynamic Regression by Classification					
factor	heuristic	*rrmse*	avg. rank	avg. # rules	avg. # conds
0.95	*wra*	0.752	8.63	15.06	38.31
0.95	*Laplace*	0.784	11.19	11.25	13.88
0.95	*correlation*	0.726	6.50	10.13	24.63
0.95	*relative cost measure*	0.780	9.81	19.06	34.25
1.00	*wra*	0.764	10.06	17.06	47.81
1.00	*Laplace*	0.774	10.63	10.19	12.50
1.00	*correlation*	0.753	8.38	9.25	22.06
1.00	*relative cost measure*	0.767	9.50	19.06	35.75
1.05	*wra*	0.780	13.13	13.19	34.19
1.05	*Laplace*	0.772	10.19	9.69	11.81
1.05	*correlation*	0.796	12.88	10.25	33.31
1.05	*relative cost measure*	0.775	9.75	19.44	37.56
Static Regression by Classification					
# classes	heuristic	*rrmse*	avg. rank	avg. # rules	avg. # conds
5	*wra*	0.883	18.25	5.63	20.75
5	*Laplace*	0.857	14.75	84.56	197.44
5	*correlation*	0.844	15.13	28.06	84.00
5	*relative cost measure*	0.852	16.63	22.88	68.00
10	*wra*	0.930	18.69	6.06	23.13
10	*Laplace*	0.872	17.00	138.44	339.25
10	*correlation*	0.864	15.88	49.31	167.25
10	*relative cost measure*	0.901	17.94	20.75	67.31
20	*wra*	0.965	20.81	10.06	36.56
20	*Laplace*	0.872	18.06	177.44	423.63
20	*correlation*	0.862	17.81	95.13	295.00
20	*relative cost measure*	0.928	19.13	33.19	102.13
Other Rule-Based Regression algorithms					
algorithm		*rrmse*	avg. rank	avg. # rules	avg. # conds
RegENDER (50)		0.768	9.38	50.00	190.00
M5Rules -R		0.773	10.44	6.19	14.94

data (*wra* [162]), one that overfits the data (*Laplace*), and two that are known to perform well in a large variety of datasets (*correlation* and *relative cost measure*). The latter features a parameter c_r, which we set to the value of 0.342 (cf. Section 4.2.2). The definition of these four heuristics and a description of them is given in Section 2.6.

Note that in our implementation, we only consider refinements that ensure that a minimum number of examples is covered. For all experiments (cf. Section 7.3) we fixed the minimum coverage to three examples. All other types of handling anomalies were kept (cf. Section 3.2.1).

We used 16 datasets in the experiments. For the selection, we focused on datasets that have a reasonable number of different values in the class variable, in order to avoid any unfair advantage that rule-based methods might have on datasets with a numerical data variable with only a few different values. Apparently, a majority of the datasets used before do not have a big number of distinct values. A result is, among other implications, that some of these datasets are basically not learnable at all (cf. the discussion about the dataset `quake` given in Section 6.3).

The 16 selected datasets were

> *auto-mpg, auto-price, auto93, cloud, compressive, concrete-slump, cpu, diabetes, echo-month, housing, machine, meta, pyrim, strike, triazines, veteran*

and are described in Section 6.2. The dataset `diabetes` is neither described in Table 6.1 nor in Table 6.2. It contains no nominal and two numerical attributes. In total it has 43 instances that have 20 distinct values.

The primary method to evaluate the algorithms is the *rrmse* obtained by a 10–fold cross-validation implemented in *Weka* [177]. Note that the regression algorithms are not yet implemented in the SECO-Framework. We do not report results on individual datasets here. Instead, we report average results over all 16 datasets. We are aware of the problems that come with averaging results over many different domains (i.e., some databases may be outliers with huge variance compared to the majority of the other datasets) and hence also report the average ranks of the algorithms in the tables. Where applicable, we also test for statistical significance using a Friedman-Test with a post-hoc Nemenyi-Test as previously suggested [32] and described in Section 2.7.5. The resulting CD-charts give insights how good the algorithms perform by evaluating their ranking independently from using average accuracy.

7.3 Results

Table 7.1 shows the results of various parametrizations of our approach in comparison to the static regression by equal-frequency discretization.

We tried different variants using five, ten, and 20 classes. For comparison, we also show the results of two other regression-based rule learning algorithms, namely M5RULES [78] used with the option `-R` so that it predicts constant values in the rule head, and REGENDER [30] in its default configuration which learns

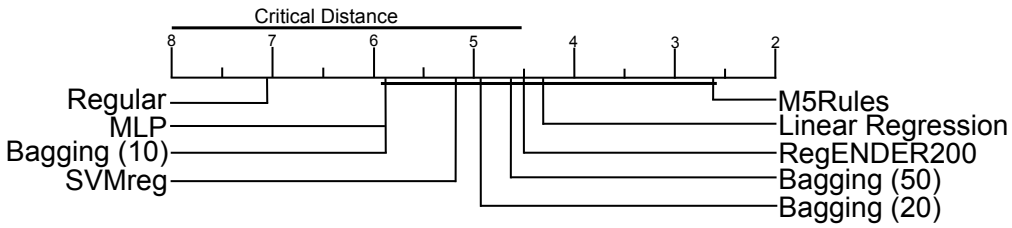


Figure 7.1: Comparison of the algorithms shown in Table 7.2 against each other with the Nemenyi test. Groups of algorithms that are not significantly different (at $p = 0.01$) are connected.

50 rules, a number that is still comparable to the number of rules learned by our algorithms. Other settings for these algorithms are evaluated further below.

It can be clearly seen that all dynamic versions outperform all static versions in all configurations. The best static version has an average rank of 14.75, whereas the worst dynamic version has an average rank of 13.13. A Friedman test can reject the null hypothesis that all algorithms are equal with high certainty ($p \ll 0.01$). The post-hoc Nemenyi test shows that the best configuration (dynamic regression using *correlation* as a search heuristic and a factor of 0.95) is significantly better than all but two static configurations at a confidence level of 0.1 ($CD = 9.354$), and better than seven at the confidence level of 0.01 ($CD = 11.125$). However, it should be noted that our rule learning algorithm does not produce probability distributions, which could improve its performance.

Somewhat surprisingly, the versions using more classes tend to have a worse performance than the versions with fewer classes. Apparently, the advantage of a finer grained discretization of the target variable is outweighed by the disadvantage that problems with multiple classes are harder to learn for the underlying rule learner. This is also confirmed by the number of rules the algorithms learned. Here, when learning 20 classes the number of rules is the highest among all configurations.

With respect to the different configurations of the dynamic approach, it seems that lower versions of the factor that is applied to the standard deviation seem to yield a somewhat better performance.

Among the four heuristics, *correlation* proves to be the best choice. This confirms our previous results on classification datasets (cf. Section 4.2). However, it seems that the good performance of the *relative cost measure* does not materialize in this setting (or at least not with the parameter setting that performed well in classification). In this context, a surprising observation is that the complexities of the regression theories do not correspond to their known behavior: while the expectations that *Laplace* overfits and *wra* over-generalizes are clearly met in classification, this cannot be observed in the regression rule sets. In fact, with the dynamic approach, *Laplace* finds fewer rules and conditions than *wra*.

Table 7.2: Comparison of a bagged version to other types of regression algorithms

algorithm	*rrmse*	avg. rank	avg. # rules	avg. # conditions
Regular	0.726	7.06	10.13	24.63
Bagged (10)	0.671	5.88	97.94	245.81
Bagged (20)	0.659	4.94	186.75	451.25
Bagged (50)	0.658	4.63	465.88	1,146.56
linear regression	0.651	4.31	—	—
MLP	0.746	5.88	—	—
SVMREG	0.673	5.19	—	—
REGENDER (200)	0.679	4.50	200.00	1,163.63
M5RULES	0.604	2.63	2.94	5.38

Interestingly, the *rrmse* monotonically decreases when the factor attached to the standard deviation is increased for *Laplace*, while it increases for *wra*. This means that evaluating rules with *wra* seems to decrease performance with a growing number of positive examples, whereas the performance of *Laplace* seems to improve. A possible explanation could be that a narrower range of positive examples may help to prevent overfitting.

Overall, the performance of the dynamic class discretization is comparable to the performance of other rule-based learning algorithms, such as M5RULES with constant predictions or REGENDER with low numbers of rules. With the best-performing heuristic, *correlation*, a slight advantage is noticeable.

However, it must be noted that the overall performance of the algorithm does not match the performance of other standard regression algorithms, which learn more elaborate models than the piece-wise constant functions that are learned by the algorithms evaluated in Table 7.1. In order to see this effect, we can compare the performance of M5RULES with the parameter setting `-R`, which predicts constant values in its rule head, to the performance of its default configuration which is able to predict linear models in the rule heads (shown at the bottom of Tables 7.1 and 7.2 respectively). In order to maximize predictive performance, an adaptation of our approach for the induction of linear models seems necessary, and is subject to future work.

A different approach to weaken the bias of the piece-wise constant prediction functions is to rely on ensemble techniques, which is the key idea of the approach taken by REGENDER. In order to evaluate the potential of our approach in such a setting, we used bagging with the best setting determined in the previous experiment (using *correlation* as a heuristic and a threshold of $t_r = 0.95 \cdot \sigma_r$). Table 7.2 shows the results for learning ten, 20, and 50 rule sets. More than 50 iterations

Table 7.3: Comparison of the two approaches with each other including other types of regression algorithms

algorithm	*rrmse*	avg. rank	avg. # rules	avg. # conditions	$\frac{\#\text{ conds}}{\#\text{ rules}}$
SeCoRegS_1	0.764	9.05	58.90	345.14	5.86
SeCoRegS_2	0.783	9.76	62.10	391.57	6.31
dynamic (*correlation*)	0.737	8.48	13.71	37.10	2.70
Bagged (10)	0.669	6.43	121.57	334.38	2.75
Bagged (20)	0.658	5.33	235.81	641.57	2.72
Bagged (50)	0.657	5.19	592.10	1,639.52	2.77
M5Rules (-R)	0.753	8.86	9.33	29.38	3.15
M5Rules	0.600	2.86	3.33	7.24	2.17
RegENDER (200)	0.656	4.71	200.00	1,259.90	6.30
linear regression	0.641	4.71	—	—	—
MLP	0.762	6.95	—	—	—
SVMreg	0.659	5.67	—	—	—

did not yield further improvements. We compared this setting to RegENDER learning 200 rules, linear regression, support-vector regression, multi-layer perceptrons, and M5Rules. While the linear or piece-wise linear models seem to be somewhat better suited for these problems, no significant difference between the models can be observed.

The value of the Friedman statistic is 3.879 which exceeds the critical value of 2.663 for $p = 0.01$. The post-hoc Nemenyi test yields critical distance values of 3.476 for $p = 0.01$ and 2.764 for $p = 0.1$ (Figure 7.1 illustrates this situation for $p = 0.01$). Thus, no significant difference can be shown between the best algorithm M5Rules and the bagged versions of our algorithms with 20 or more theories. Conversely, the test also does not show a significant difference between our regular dynamic regression by classification approach and any of the other regression algorithms except the model-based rule-learning algorithm M5Rules. However, this result should be put into perspective, because in Figure 7.1 we only included the best-performing parameter setting of Table 7.1. It seems obvious that the performance of the algorithm without bagging is substantially worse than the performance of the other algorithms.

7.4 Comparison of the Metric-Based Algorithms and the Dynamic Reduction

In this section the two different approaches to regression that were described in Chapter 6 and in this chapter are compared against each other. For the experiments, the datasets of Section 7.2 in union with those displayed in Table 6.2 were

used. Table 7.3 shows the results for the algorithms of Table 7.2 and the two tuned variants of the SECOREG rule learner. Note that these two algorithms were optimized on eleven of the 21 datasets of the displayed evaluation[1]. Nevertheless, it seems fair to do such a comparison, because the SECOREG learners were tuned only on 50% of the instances of these eleven datasets. For comparison the other algorithms used in this chapter are also included.

As can be seen in Table 7.3 the dynamic approach was better than the two metric-based algorithms of Section 6.3. Interestingly, the dynamic reduction allows the algorithm to find much less rules and conditions than the SECOREG algorithm. The number of conditions each rule has is also lower. This once again emphasizes the importance of the rule learning heuristic in separate-and-conquer algorithms. In spite of the simple way to reduce the regression problem to classification, the *correlation* heuristic is able to improve the heuristic search drastically. The c_{cm} heuristic shown in Equation 6.1 uses regression statistics directly. The results show that this direct adaptation of the heuristic does not work as well as the dynamic reduction. Here, the classification statistics (i.e., positive and negative examples) are estimated and then used to compute the heuristic evaluation of each candidate rule. Providing these statistics to a regular classification heuristic works better than the adaption of the heuristic to regression. This is confirmed in two ways:

- The average *rrmse* of the dynamic approach is better than that of the two direct approaches, and
- the number of rules and conditions and also the ratio of conditions per rule is much lower for the dynamic approach (13 rules vs. about 60 rules, 37 conditions vs. over 300 conditions, and a ratio of only 2.7 conditions per rule for the dynamic method vs. nearly six for the other two approaches).

Regarding the results displayed in Table 7.2 the ranking of the algorithms in terms of average *rrmse* remains stable. In both experiments M5RULES was the best one, followed by the linear regression. In Table 7.3 REGENDER ranks on the third place where it was fourth in Table 7.2. The SVMREG has switched its place with REGENDER. The difference in terms of *rrmse* is negligible in both tables. The dynamic approach ranks on the fifth place in both tables, followed by M5RULES-R and MLP. Thus, the advantage of the dynamic method over the M5RULES-R learner is consistent with Table 7.3. The dynamic approach finds more rules than M5RULES-R but the performance of them is better.

Where in Table 6.7 the direct approach tuned on the first subsets was ranked on the third place, its performance drastically decreases in the experiment displayed in Table 7.3. Both of the tuned variants are ranked on the last places there. Interestingly, the SVMREG had its best performance in the experiments summarized in

[1] These datasets were `auto-mpg`, `auto-price`, `compressive`, `concrete_slump`, `cpu`, `echo-month`, `housing`, `machine`, `pyrim`, `strike`, and `triazines`.

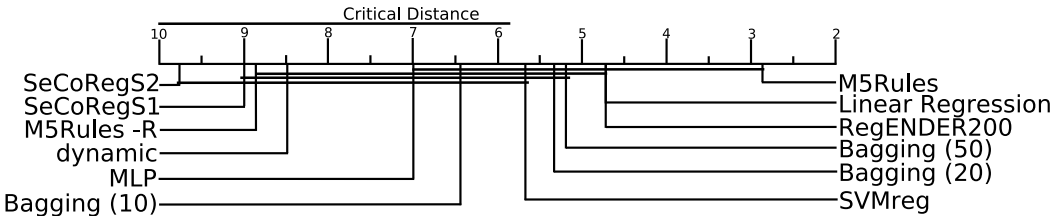


Figure 7.2: Comparison of the algorithms shown in Table 7.3 against each other with the Nemenyi test. Groups of algorithms that are not significantly different (at $p = 0.01$) are connected.

Table 6.7, where it was clearly ranked on the first place. Both the linear regression and the M5RULES algorithm had a rather bad performance in these experiments. On the one hand the reason could be the selection of datasets, but on the other hand the difference in performance could also result from a different seed used for the cross-validation in these early experiments. As mentioned in Section 6.5.3 the dataset `meta` has shown strong fluctuations in all experiments and this may have also influenced the results.

Figure 7.2 displays the critical distance diagram for all algorithms. The Friedman Test yielded a value of 10.839 which exceeds the critical value of 2.33 for $p = 0.01$. The Nemenyi Test has a critical distance of 4.16. M5RULES, ranked on the first place, is significantly better than the two SECOREG algorithms, the configuration where the M5RULES algorithm predicts single values, and the dynamic approach. This confirms the previous results given in Figure 7.1 where M5RULES was also significantly better than the dynamic approach. As discussed before, the main reason is that the latter uses piecewise linear models whereas the rules found by M5RULES are not restricted to predict a single value.

Furthermore, the linear regression and REGENDER, both ranked on the second place, outperform the two SECOREG algorithms significantly. At least for the linear regression these results are in contrast to those observed on the seven test sets used in Section 6.5.3. Bagging with 50 rules is significantly better than the SECOREG learner tuned on the subsets S_2. The SVMREG is not significantly different from all other algorithms. Note that all bagged variants of the dynamic approach perform very well given the selection of algorithms.

7.5 Related Work

Work concerning regression rule learning in general was previously presented in Section 6.6. In this section, related work specific to reduction schemes is given. Most of the work that concentrates on reduction schemes is based on ordinal regression or ranking. Here, each example is labeled by a certain rank. A framework

exists that is able to reduce ordinal regression to binary classification [108]. The framework operates in three steps where in the first one extended examples are extracted, in the second one a binary classifier is learned on these examples, and in the last step a ranking rule from the binary classifier is constructed. Detailed information on that topic can also be found in the literature [109].

There is some work on reducing regression level set estimation to classification [150]. The goal here is to find a regression function for which the value is above a certain threshold. This can then be reduced to cost-sensitive classification.

A regression problem can also be converted into a set of binary classification problems, where each classifier essentially tests whether an example is above or below a certain threshold [103]. In the paper, plots are given that relate the error to the number of binary classifiers. The curves are converging, but still, the question how to determine the optimal number of classifiers which, in some sense, is related to finding the right number of bags for a discretization process, is not solved there. For instance, for the dataset `adult`, the best number of binary classifiers is around 40, whereas for the dataset `boston` the highest performance is reached with approximately ten binary classifiers [103].

In Section 7.2 the approach of discretizing the numerical target variable and afterwards using classification algorithms was mentioned. In the experiments this method was used with the equal-frequency approach where in each interval approximately the same number of instances reside. There are other methods to perform the discretization. Often domain knowledge can be used which means that suitable discretizations are already known. Weiss and Indurkhya introduced the algorithm P-CLASS [174]. It essentially works like a K-MEANS clustering algorithm [75] where the number of classes or k, the number of clusters, has to be known in advance. Other discretization methods include search-based methods that work by using different parameter settings and choosing the one that yields the best predictive accuracy, and a method that is based on the misclassification cost [165]. As mentioned before, the main drawback of this kind of reduction scheme is that the number of classes has to be known in advance and indeed is hard to determine.

7.6 Summary

In this chapter a reduction scheme was derived that is also applicable to other algorithms that operate on coverage statistics. In this way, classification rule learning heuristics can directly be used to solve the regression problem. Some heuristics behave differently than known from classification when they are used in the presented algorithm. Most interestingly the behavior of *wra* and *Laplace* is contrarily to what is known from classification (cf. Chapter 4). Moreover, the only parametrized heuristic utilized in the reduction algorithm, namely the *relative cost measure*, was outperformed by *correlation*. These results suggest that the trade-off has to be re-

adjusted for the regression task. However, the presented algorithm was able to compete with state-of-the-art regression rule learning algorithms that use aggregated predictions and was able to clearly outperform algorithms that reside in the same model class, i.e., that predict single target values and are based on decision lists.

8 Experiments on Real-World Data

In this chapter the algorithms derived in the previous chapters of the thesis are applied to real-world data. In total, there were two external projects in which it was tried to solve different real problems with our rule learning algorithms. The first problem resides in the medical domain, the second problem was to predict whether or not a student has to be invited to a personal discussion based on a questionnaire. This session has the purpose to guide the student such that he or she will continue his or her studies with more success. Both projects were conducted in cooperation with external members either from other universities or with physicians.

In the first part, we focus on predicting skin cancer. This work was a cooperation with a dermatologist. The goal of the project was to analyze a big number of questionnaires that were filled out by patients during a period of several years. The main challenge was the pre-processing of the data as well as an assessment whether machine learning can be applied successfully in this domain. The necessary steps to answer these questions were done by Fischer [44].

The second part describes experiments that had the goal to induce a set of rules that is able to assist in categorizing students. The data were also extracted from questionnaires. In a practical project, all necessary pre-processing was conducted. The main challenge was the extraction and conversion of the web-based questionnaires so that an .arff-file is yielded that could then be used for learning.

The chapter starts with Section 8.1 where the skin cancer experiments are summarized. Section 8.2 is concerned with the categorization of students. Both sections are organized as follows: They begin with an introduction to the domain. Then the datasets are described. The sections are finished with a description of the results, including a presentation of interesting rules. In Section 8.3 related work of both domains is given and in Section 8.4 a summary of results of both topics is given.

8.1 Using Rule Learning Algorithms to Predict Skin Cancer

In 2011, we conducted a collaborative study in cooperation with the "Qualitätsgemeinschaft Südhessischer Dermatologen"[1]. One of the members of this institution is Dr. Matthias Herbst, a dermatologist who has his office in Darmstadt. He was the contact person concerning questions about the domain of skin cancer.

[1] `http://www.qsd-ev.de/` (visited 2012-03-30)

As the cases that are newly infected with skin cancer have increased steadily over a period of over 30 years now [8] and the chance of being cured is strongly related to a successful cancer screening program, the importance of auspicious incidence discovery, i.e., detecting patients that have a high risk of getting skin cancer, is immanent. A major problem is the identification of persons for whom a cancer screening would be beneficial. As, for financial reasons, not all persons can participate in such a preventive program, it has to be assessed which patients have a higher risk of getting skin cancer and which are more likely to stay healthy. For this reason, the "Qualitätsgemeinschaft Südhessischer Dermatologen" has gathered data from questionnaires that were filled out by a large number of patients. This program is not restricted locally but spans over entire Germany and was run for about four years.

The goals of the collaboration are twofold: First, it has to be assessed whether these questionnaires can constitute the basis to learn a good classifier and if so, a rule-based system should be designed. The main reasons to rely on rules is that the interest to have an understandable model usually is high in the medical domain. The described project is no exception as one of the benefits of rules would be to compare them to existing rules that are indirectly used by the doctors to classify their patients. Gaining a deeper understanding of the potential relationships of the attributes and the class is also one of the goals. In this sense, rules are a good choice to provide the required interpretability. The project is summarized in the paper [44] and most of this section is based on this work.

In the next section, an introduction to the domain of skin cancer is given. The different types of skin cancer are explained. Then, the datasets that were created from the questionnaires are described (Section 8.1.2). Note that the questionnaires consist of additional evaluations of the corresponding physician itself and therefore different types of datasets are created. These are also summarized there. In Section 8.1.3 the results are presented.

8.1.1 Introduction to the domain

Under the term "skin cancer" the most common types of skin cancer are summarized. These can be categorized by the fact whether or not they emerge from the melanocytes[2]. One of the most dangerous types, the *malignant melanoma* is indeed emerging from the melanocytes. This type of cancer is likely to cause metastatic spread which makes it extremely dangerous. On the positive side, this type of skin cancer is quite rare. More common types of the second category, i.e., the non-melanoma ones, are the *basalioma* and *spinalioma*. The first one is the most frequent skin cancer with a relative frequency of about $75 - 80\%$. It has a low risk of metastatic spread and only about 0.1% of the patients actually die. As the

[2] Melanocytes are pigment cells of the human skin that are able to synthesize melanin.

Table 8.1: The attributes given by the patient and by the dermatoligist

category	attribute name	abbreviation	attribute type
patient	age	age	numerical
	sex	sex	binary
	time spent in intense sun	$hours_{sun}$	nominal
	sunburns	sb_{total}	nominal
	sunburns in childhood	sb_{child}	nominal
	sunburns as a juvenile	sb_{juv}	nominal
	sunburns as adult	sb_{adult}	nominal
	sun protection	$protect$	nominal
	usage of solarium	$solarium$	numerical
	outdoor sports	$sport$	binary
	skin cancer history	$sc_{history}$	binary
	skin cancer history in family	sc_{family}	nominal
dermatologist	number of pigments	$pigm$	numerical
	dysplastic nervus cell nevi	dys	binary
	skin type education	edu_{skin}	binary
	education about solar protection	edu_{sp}	binary
	education about $ABCD$-rule	edu_{ABCD}	binary

abovementioned three types of skin cancer have a cumulative relative frequency of about 95% of all new cases [138], other types of skin cancer are not taken into account here.

Surprisingly, the number of new cases steadily increases while the mortality rate is decreasing [8]. It was suggested that the reasons mostly are a much better spread of methods to detect skin cancer among the public [8]. These methods can usually be performed without consulting a doctor. Most often skin cancer develops from moles or liver spots. Usually, it can be self-detected whether or not a certain mole is at risk to develop into cancer. The main method for the prevention of the development of skin cancer is the so-called *ABCD*-rule. Interestingly, this rule itself was derived from a multivariate analysis of 31 dermatoscopical criteria [156]. In the rule, *A* stands for asymmetry (the shape and the delimiters of the mole), *B* refers to boundary (especially the boundary at the borders of the mole), *C* means color (the darker the mole the worse it can develop), and *D* is the diameter (moles that have a diameter greater than two centimeter are involving a high risk). As skin cancer in an early stage can most often be healed, a prevention in terms of detecting possible risk-carrying moles certainly is beneficial.

Table 8.2: Statistics of the three datasets

dataset	# attributes	type of class	distribution	accuracy of baseline
D_1	12	risk estimation (binary)	high risk: 2,531 low risk: 4,192	61.91
D_2	17	risk estimation (binary)	same as above	same as above
D_3	17	traffic light model (3 classes)	green: 5,939 yellow: 778 red: 54	87.71

Despite the good progress made in the prevention of skin cancer, a detailed analysis carried out by a dermatologist can never be replaced. Thus, it still is necessary to assess whether or not a patient has a high risk of developing skin cancer. Self-prevention is one approach, but, inevitably, mechanisms to categorize patients based on certain habits or medical records are required. As the number of possibly endangered persons is huge but capabilities of physicians are limited, an automated method would be of great value. The problem is also related to financial issues because health insurance companies can save money if only those patients are invited to preventive cancer screenings that are actually affected. For this reason, the research on a system that is able to categorize patients to decide whether or not such a screening is necessary is justified.

8.1.2 The datasets

During several years approximately 7,000 questionnaires were collected. The questions were separated into two parts: The first part has to be filled out by the patient and the second part contains a short evaluation of the dermatologist. Here, some additional information towards a first diagnosis are given. These mostly include details about the number of pigments a patient has as well as some educational issues, i.e., whether or not some obvious relations between certain causes of skin cancer were clarified. In Table 8.1 the attributes used in the study are displayed.

Note that based on these attributes three preliminary datasets were created $(D_1, ..., D_3)$. These datasets are then processed by several pre-processing steps (see below). The first distinction is whether or not the additional attributes of the dermatologist are used and the second one concerns the class that has to be predicted. One choice is to use the risk of getting skin cancer as target value and the other is a so-called traffic light model where the risk is separated into three

different classes. Here, the green class contains uncritical cases, the yellow class involves those patients that have a moderate risk, and the red class incorporates the patients that have a high risk. Note that D_1 and D_3 are the interesting datasets as D_2 already incorporates opinions and estimates of the dermatologist. Note that D_3 also includes such evaluations but here it is harder to predict three classes instead of only two. Not surprisingly, the results on D_2 are much better than those determined on the other two datasets. Table 8.2 gives statistics about the three datasets.

Most of the attribute value ranges are related to frequencies. Hence, $hours_{sun}$ has the possible values "often", "occasional", "rarely", or "avoided". The three attributes concerning the sunburns (sb_{child}, sb_{juv}, and sb_{adult}) have four possible values: "often", "frequently", "rarely", and "never". As pointed out [44], these categories are rather vague as, e.g., the distinction between "frequently" and "often" is not intuitively measurable. The attribute $protect$ can have the following values: "consequently", "rarely", or "no statement possible". Finally, the attribute sc_{family} has additionally to the binary decision a value "unknown" as often the skin cancer history in the family is unclear.

Due to a large list of data-driven problems and inconsistencies these three datasets had to be processed in several ways which are described in the next section.

Data Pre-Processing, Missing Values, and Inconsistencies

The datasets created above have several problems that have led to extensive preprocessing. In this section, only parts of all the necessary steps are described. For a detailed description see [44]. First of all, 167 questionnaires did not contain a risk estimate and therefore were removed. Thus, the datasets have 6,771 instances in total. Unfortunately the datasets contain many inconsistencies that mostly come from medical examinations of one and the same patient conducted by different dermatologists, some of which classified patients into the high risk class while others ascertained a low risk. For solving this problem, the majority vote was used. Another severe problem is that the number of missing values of some attributes is higher than average. The three attributes that describe sunburns had the largest number of missing values among all attributes[3]. It was suggested to replace the missing values by a nearest neighbour algorithm using the Euclidean distance and a value of $k = 17$ [44]. Four different variants of the replacement of missing values were evaluated [44]:

- Replace all missing values,

[3] The percentage of missing values for the attribute "sunburn in childhood" was 30%, for "sunburn as a juvenile" 26%, and for "sunburns as adult" 15.7%.

Table 8.3: Theory size and performance of different *Weka* algorithms

dataset	algorithm	# rules	# conds	accuracy	relative to baseline
D_1	RIPPER with pruning	5	13	64.26	+2.35
	RIPPER without pruning	11	55	63.47	+1.56
	PART with pruning	292	1,457	61.76	−0.15
	PART without pruning	1,418	9,089	59.07	−2.84
D_2	RIPPER with pruning	10	28	73.7	+11.79
	RIPPER without pruning	44	234	69.27	+7.36
	PART with pruning	302	1781	71.13	+9.22
	PART without pruning	1,551	12,614	67.7	+5.79
D_3	RIPPER with pruning	5	17	87.74	+0.03
	RIPPER without pruning	14	88	87.73	+0.02
	PART with pruning	128	639	86.86	−0.85
	PART without pruning	1,001	8,059	83.98	−3.73

- replace only those missing values that have the highest frequency[4],
- keep missing values, or
- remove the instances that have missing values.

We only present results for the first method, i.e., the one that replaces all missing values. One may argue that the performance usually relates to the method of replacement which is partly confirmed [44]. Nevertheless, this is the most intuitive way to deal with such high numbers of missing values.

Recall that we used two different scenarios of prediction: In the first one, the risk of developing skin cancer is used as target value, and in the second one a traffic-light model was employed. The risk is described by one field in the questionnaire. Here, the dermatologist has evaluated whether or not the patient has a high risk of developing skin cancer. For the second type of prediction, the patients had to be categorized into three risk classes (green, yellow, and red). Based on the mortality rate of the different skin cancer types, it was decided to use the red class for the malign melanoma and the spinalioma [44]. The latter itself is not that dangerous but the risk of metastatic spread is very high. The yellow class contains patients where a basalioma or an actinic keratosis (prestage of skin cancer) was diagnosed. The green class contains patients that are classified as safe. Whether or not this

[4] The percentage of the other four attributes that contained missing values was only 1.4% to 3.5%.

Table 8.4: Theory size and other statistics for selected configurations of SIMPLESECO

dataset	heuristic	*minCov*	# rules	# conds	performance relative to baseline
D_1	h_{acc}	1	16	41	+2.84
D_1		2	3	6	+3.17
D_2		1	76	249	+10.84
D_2		2	4	10	+11.64
D_3		1	276	1,112	−2.58
D_3		2	2	5	+0.04
D_1	h_{corr}	1	1	2	+3.07
D_2		1	4	11	+11.5
D_3		1	30	198	−0.74
D_1	h_{LR}	2	1	2	+3.08
D_2		2	4	15	+11.16
D_3		2	1	8	−0.01

categorization is meaningful is not a question that can be answered in this thesis, in that case we trust in the expertise of the dermatologist. Table 8.2 comprises an overview about the distribution of the different classes and the accuracies of the baselines as well as the number of attributes of the three final datasets.

8.1.3 Results

In this section the results are presented. Firstly, we evaluate some *Weka* rule learners, i.e., RIPPER [22] and PART [47]. Here, we concentrate on the comprehensibility of the rule set by measuring the number of rules and conditions. Performance is also an issue, but as pointed out [44], most of the algorithms are unable to outperform the baseline (at least not when no evaluation of the dermatologist is given). In the following, some interesting statistics about some configurations of the SIMPLESECO rule learner are given. As we evaluated a large number of different heuristics, only worthwhile results are listed. The section is concluded by the presentation of interesting rules. The focus here lies in the differences of single rules found by different algorithms as one goal is to show that the SECO-Framework is capable of producing various different types of rules for a given domain by using a wide variety of heuristics.

Table 8.3 lists the results in terms of comprehensibility for the *Weka* [177] algorithms. Both algorithms were evaluated with and without pruning on the three datasets described in Section 8.1.2. The results are mixed. As RIPPER is capable

r_1 $sc_{history} = \text{yes} \rightarrow$ high risk=yes [67,247]

r_2 $age \geq 27 \wedge$ protect=consequently $\wedge$ $sc_{family} = \text{yes} \rightarrow$ high risk=yes [155,160]

r_3 $age \geq 83.5 \wedge sb_{juvenile} = \text{never} \rightarrow$ high risk=yes [2,15]

r_4 $sc_{history} = \text{yes} \wedge age \geq 26 \rightarrow$ high risk=yes [64,244]

r_5 $sc_{history} = \text{yes} \wedge age \geq 27.5 \rightarrow$ high risk=yes [64,244]

Figure 8.1: Selected rules on dataset D_1

of inducing compact rule sets, PART clearly fails in this task. Even for the variant that uses pruning, the rule sets are by far too large, peaked by the unpruned version of dataset D_2 with over $1,500$ rules and over $12,000$ conditions. Clearly, such rule sets are suboptimal in this domain as they cannot be interpreted any more. Interestingly the performance of PART also lacks behind RIPPER in all cases. For two of the datasets (D_1 and D_3) the performance is even worse than the baseline. Thus, PART even fails to achieve any of the two main goals: inducing a compact comprehensible rule set, and maintaining a certain level of performance. Instead, RIPPER is able to achieve both goals, albeit the interpretability of the theory found on D_2 with the unpruned version is questionable (44 rules with 234 conditions).

The results of the SIMPLESECO configurations are also rather mixed (Table 8.4 gives an overview of some selected configurations). Although all configurations are able to outperform the baseline by a small margin on D_1, four configurations ended up with an empty theory (those are not included in Table 8.4. The best performance was achieved by *accuracy* with a minimum coverage of two on all of the three datasets. Interestingly, on all the three datasets, the theories found by this configuration were those that were most comparable to RIPPER's output. In summary, *Laplace*, the *relative cost measure*, and the m-estimate had the highest number of rules and conditions (up to 167 rules with 713 conditions). Four configurations yielded the same single rule (rule r_4 or r_5 in Figure 8.1, which are semantically the same as their statistics are similar), these were *correlation* and the *Linear Regression* with a minimum coverage of one and two, respectively.

The dataset D_2 exhibits even more variance. The number of rules scales up to 458 and the highest number of conditions was $2,070$ (*Laplace* with a minimum coverage of one). In between are five configurations that had about 80 to 160 rules with about 250 to 500 conditions. Then, again, four configurations had found only a single rule, which interestingly is a refinement of the first rule RIPPER has found (cf. rule r_6 and r_9 in Figure 8.2). Note that as mentioned above the performance is the best on this dataset as the task for the learner is simplified because the dataset has only two classes and some additional attributes were given by the dermatologist.

r_6 dys=yes → high risk=yes [198,791]

r_7 $sc_{history}$ = yes → high risk=yes [65,225]

r_8 dys=yes ∧ edu_{skin} = yes ∧ $pigm \geq 35$ ∧ edu_{ABCD} = yes → high risk=yes [0,92]

r_9 dys=yes ∧ $age < 85$ → high risk=yes [197,791]

r_{10} dys=yes ∧ $age < 84$ → high risk=yes [197,791]

r_{11} $sc_{history}$ = yes ∧ $age \geq 23.5$ → high risk=yes [62,224]

r_{12} dys=yes ∧ $age \geq 30$ ∧ $age < 81$ ∧ edu_{skin} = no ∧ edu_{ABCD} = yes
→ high risk=yes [0,119]

Figure 8.2: Selected rules on dataset D_2

D_3 is the most extreme case. A total of seven configurations ended up with no rule, for which of the three classes rules are learned varies considerably, and the theory size ranges from a single rule with eight conditions to 322 rules with 1,456 conditions. Note that a big problem of the dataset D_3 seems to be that the three classes are hard to separate. For this reason, most often the algorithms only induce rules for one class and not for two of the three classes as desired.

In the following we will have a closer look on individual rules. In this analysis, no rules for the PART algorithm are presented as the theories were far too big, so that no single interesting rule could be found (cf. the statistics given in Table 8.3). As the three datasets capture different aspects, rules are presented in turn for each of them. Figure 8.1 shows some selected rules for dataset D_1. The first two rules (r_1 and r_2) are found by RIPPER in default configuration. SIMPLESECO with *accuracy* and $minCov = 2$ also finds rule r_1. Rule r_1 seems to be an obvious one as metastatic spread is more likely when someone had cancer before (the patient had screen cancer history, i.e., $sc_{history}$ = yes). The statistics of the rule are reasonable albeit far from perfect. In the next iteration RIPPER finds rule r_2 whereas the sketched SIMPLESECO configuration finds rule r_3. Clearly, statistically and semantically, rule r_2 is suboptimal. Covering 160 positives and 150 negatives is inadequate. The condition that someone who applied sun protection consequently is likely to get skin cancer is counter-intuitive. Interestingly, rule r_3 seems to be a better alternative at least when the coverage statistics are inspected. However, the condition that no sunburns as a juvenile are registered usually is also not an indication for a possible skin cancer risk.

The rules r_4 and r_5 which were found by three SIMPLESECO configurations, are semantically the same given the dataset D_1. Both of them have the same coverage statistics, reducing the false positives by three compared to r_1 but also lose three true positive. Given that the costs of a true and a false positive are equally weighted, this rule can be seen as an alternative to the one RIPPER has found.

r_{13} $age \geq 60 \wedge edu_{ABCD} = \text{no} \wedge \text{sex=male} \wedge pigm \leq 15$
$\rightarrow$ risk class=yellow [34,67]
r_{14} $age \geq 71 \wedge edu_{ABCD} = \text{no} \rightarrow$ risk class=yellow [53,60]
r_{15} $age \geq 78 \wedge edu_{ABCD} = \text{no} \wedge \text{sex=male} \wedge sb_{adult} = \text{never} \wedge sb_{total} = \text{rarely}$
$\wedge$sport=yes $\rightarrow$ risk class=yellow [0,10]
r_{16} $age \geq 84 \wedge age < 86 \wedge sc_{family} = \text{no} \rightarrow$ risk class=yellow [9,23]
r_{17} $age \geq 84 \wedge \text{protect=consequently} \wedge edu_{ABCD} = \text{no} \wedge edu_{skin} = \text{yes}$
$\rightarrow$ risk class=yellow [0,7]
r_{18} $age \geq 83.5 \wedge age < 91.5 \wedge sc_{family} = \text{no} \rightarrow$ risk class=yellow [15,28]

Figure 8.3: Selected rules on dataset D_3

Figure 8.2 summarizes the most interesting observations on dataset D_2. As stated above, this dataset is much easier to learn, but still it is of interest what characteristics the rules have. First of all, in terms of consistency and coverage, the rules are much more accurate compared to dataset D_1. RIPPER (default) at first finds rule r_6, which only concentrates on the feature "dysplastic nervus cell nevi (*dys*)". After consulting the dermatologist, it became clear that this feature is strongly related to the skin cancer risk. Again, a total of five configurations (*accuracy* with $minCov = 2$, the F-measure, and *weighted relative accuracy* with $minCov = \{1, 2\}$) find two semantically similar rules (r_9 and r_{10}). However, these rules outperform the RIPPER rule by excluding an additional false positive. The second rule found by RIPPER in its default mode is rule r_7 which also is promising. Clearly, the skin cancer history is correlated with the skin cancer risk. The coverage statistics also are of high quality. Nevertheless, the SIMPLESECO learner instantiated with *accuracy* and $minCov = 2$, is able to outperform RIPPER again. By testing also on the age of the person, rule r_{11} is able to only lose one true positives while excluding three false positives.

Interestingly, the variant of RIPPER where pruning is disabled, finds a consistent rule with high coverage (rule r_8). Covering 92 true positives and no false positive is near perfect. Two of the four conditions are intuitively correct but the relation of a solid education that results in a higher skin cancer risk is somewhat unusual. SIMPLESECO with *Linear Regression* and $minCov = 2$ is able to induce the best rule given the constraint that no negatives have to be covered. Semantically, rule r_{12} has five conditions, containing the age, using the *dys*-feature and two of the educational attributes. All of these make sense, only that education about the *ABCD*-rule does not prevent skin cancer is counter-intuitive.

Figure 8.3 summarizes interesting rules found on dataset D_3. As may eventually be noted, all rules predict "risk class = yellow". The main reason for this is that most of the rule learner omit learning rules for the "red" class while the "green"

class is covered by the default rule. Some of the configurations of SIMPLESECO with $minCov = 1$ induce rules for the class "red", but nearly all of them are so-called small disjuncts [79], i.e., covering a few positives and no negative example. The first two rules (r_{13} and r_{14}) are found by RIPPER in default configuration. Both of them are meaningful rules, only the conditioning on the number of pigments seems unusual. One would expect that the higher the number of pigments is, the higher the risk of getting skin cancer should be (as one previous rule also confirms, cf. rule r_8). Nevertheless, the coverage statistics of the first rule are satisfying, but those of the second one are clearly suboptimal. In contrast, rule r_{15} found by RIPPER without pruning is promising by covering ten true positives and no negative. Its test for outdoor sports is essentially a valid indicator for high skin cancer risk.

In the following, some rules are presented that are found by SIMPLESECO. For all of them $minCov = 2$ holds. While r_{16} and r_{18} cover negative examples, r_{17} is a pure rule that was found with *Laplace*. Albeit not covering as many positives as r_{15} does, the attribute tests of the rule are promising. Only the test for the education on skin cancer is counter-intuitive. The other two rules test the same attributes, but use different ranges. The first one covers 23 positives and nine negatives and was found by *accuracy*. The latter was induced by employing the m-estimate and loses five positives, but also covers six negatives less than r_{16}.

8.2 Using Rule Learning to Identify Students Who Need Assistance

The second problem was to categorize students based on characteristics extracted from questionnaires. As publicly available rule learning systems as RIPPER often lack sufficient configuration options, the SECO-Framework is one choice to overcome this problem. In RIPPER, e.g., the only options are how many examples should be covered by each rule, whether or not a pruning should be used, and how many optimizations should be performed. Other, more substantial options as, e.g., the heuristic cannot be configured in the *Weka*-implementation of RIPPER. On the other hand, as described in Chapter 3, nearly every dimension of a rule learner can be configured in the SECO-Framework which suits it perfectly when different types of rule sets should be learned. Consequently, the SECO-Framework was our main tool to induce rule sets for this domain. This section summarizes the research of the project with a focus on the usability of the induced rules rather than on their accuracy. The main reason for doing so is that there is not much training data yet which makes it complicated to obtain solid performance estimates.

The next section consists of a detailed description of the domain. Here, a motivation is given and the different institutions that were involved in the project are described. Then, the dataset is shown and some problems mostly concerned with the statistics of the dataset are revealed (Section 8.2.2). In the following, the results of different rule learning algorithms are presented (Section 8.2.3).

8.2.1 Introduction to the domain

At the Technical University Darmstadt[5], a system to assist students to study trouble-free is installed since the winter term 2006–2007. Since the winter term 2007–2008 it is done in cooperation with the "Hochschuldidaktische Arbeitsstelle" (hda)[6]. This system is called "Mentorensystem Informatik"[7] and is particularly designed to assist students of computer science. The students are not only supported during their entire studies, but an essential part of the assistance is to evaluate which students may encounter problems particularly during the first two semesters. The main goal is to reduce drop-out rates. Thus, after the first semester, it is considered which students have accomplished only one of up to four exams successfully. Those are invited to a counseling session where experienced employees of the university try to identify the reasons why the students had problems during the first semester. Usually one session is about half an hour long. In the following such a session is also called "personal session", "special session", or simply "session".

As the number of students that start to study computer science at the Technical University Darmstadt is constantly high or even increasing for a while now, it is of interest to identify students that will pass less than two exams in advance. It is also important what criteria students may have that will lead to suboptimal results during the first two semesters. Currently, the students have to fill out a questionnaire which then is analyzed manually. Depending on some predefined criteria which mostly stem from experience, a certain number of students is identified that is invited to such a personal session. As the number of students in question often exceeds 50, it takes a long time to identify students for whom a personal session would be beneficial. Also, the experience that is the basis for such a selection, usually is hard to formalize and sometimes rather resembles an intuitive procedure. For this reason, during a practical course[8], it was evaluated whether a machine learning approach to this problem is feasible. The main focus here lies in the interpretability of the model because it is of importance to notice why an automated machinery would assign a student to participate in the personal session. Hence, using rules seems to be a natural choice.

[5] `http://www.tu-darmstadt.de` (visited 2012-05-07)

[6] `http://www.hda.tu-darmstadt.de` (visited 2012-05-07)

[7] `https://www.informatik.tu-darmstadt.de/de/studierende/studiengaenge/bachelor-informatik/mentorensystem/` (visited 2012-05-07)

[8] The practical course was conducted by Maxi Neubacher. The report on the project can be found under `http://www.ke.tu-darmstadt.de/lehre/arbeiten/studien/2012/Neubacher_Maxi.pdf` (visited 2012-05-07).

Table 8.5: The 44 attributes used for prediction

category	attribute name	abbreviation	attribute type
generals	distance to university	$dist$	nominal
	job	job	binary
	hours per week for job	$hours_{job}$	nominal
	hours per week for friends	$hours_{fr}$	nominal
	hours per week for computer	$hours_{c}$	nominal
	hours per week for social networks	$hours_{sn}$	nominal
	hours per week for tv	$hours_{tv}$	nominal
	text in field private issues	$issues$	binary
school related	school grade	$grade_{sc}$	nominal
	grade in mathematics	$grade_{m}$	nominal
exam related	# registered exams	no_{reg}	numerical
	# de-registered exams	no_{dereg}	numerical
	# permitted exams	no_{per}	numerical
	# exams written	no_{wr}	numerical
	# exams ill	no_{ill}	numerical
	# exams passed	no_{pass}	numerical
	# exam inspections	no_{insp}	numerical
lecture related	frequency participated in lecture lec	$freq_{l}(lec)$	nominal
	frequency participated in exercise lec	$freq_{e}(lec)$	nominal
	frequency homework done for lec	$freq_{h}(lec)$	nominal
study related	hours for lecture	$hours_{l}$	nominal
	hours for private studies	$hours_{pr}$	nominal
	study group	$group$	binary
self evaluation	subject matter too complex	$subj_{comp}$	nominal
	subject matter too much	$subj_{much}$	nominal
	subject matter different than expected	$subj_{diff}$	nominal
	spent too few time	$time$	nominal
	undertaken too much	$undertaken$	nominal
	no plan how to learn	$plan$	nominal
	exam different than expected	$diff$	nominal
	lecture does not fit exam	fit	nominal
	too nervous during exam	$nervous$	nominal

8.2.2 The dataset

In the scope of the *Mentorensystem Informatik* described above, students have to fill out a questionnaire via the internet. The questionnaire contains 52 questions

in total, where some of them have either nominal or numerical values and some of them can be answered with free text. The free text fields are usually converted into binary attributes, i.e., *true* for a filled text field and *false* if it is left empty. In a first step, questions are identified that seem to be important and that are beneficial when included in the dataset[9]. Thus, the dataset features a total of 44 attributes, where 34 are nominal, three are binary, and seven are numerical ones. Table 8.5 summarizes the attributes by categorizing them into six types. The target value is a binary attribute that decides whether or not a student has to participate in a counseling session.

Most of the nominal attributes in Table 8.5 can be seen as discretized numerical attributes. The reason is that the answers are pre-given in the questionnaire based on certain ranges. For example, the distance to the university is given by four intervals {less than 15 minutes, between 15 and 30 minutes, up to 60 minutes, more than 60 minutes}. The grades are also discretized in certain intervals because it is more important to know in which region a student is situated than to know the exact grade. All attributes that are measured in hours per day reside in half an hour intervals up to two hours. The attributes that are given in hours per week are between less than five and $25 - 30$ hours, in steps of five hours. For the lecture related attributes, the variable lec is a placeholder for a particular lecture, i.e., {*Introduction to Computer Science 1* (GDI1), *Formal Fundamentals of Computer Science 1* (FGDI1), *Human Computer Systems* (HCS), *Technical Fundamentals of Computer Science* (TGDI1), *Mathematics 1* (Ma1)}. For all of them, there exist three attributes shown in Table 8.5. The possible values are {nearly all the time, often, irregularly, rarely, refuse}.

For the eight attributes of the category *self evaluation* the values were {fully applies, rather applies, rather not applies, does not apply}.

The project was focused on pre-processing the data rather than to evaluate and select certain algorithms. To this end, a dataset in the *.arff*-format was created. Currently, it has 123 instances that were derived by the manually labeled questionnaires of the winter term 2010–2011 and the summer term 2011. Thus, the ratio between the number of instances and attributes is rather skewed. To overcome this problem, a natural choice is to perform a feature selection (for an overview see [111] or [73]). Due to time restrictions this was not conducted in the project. Nevertheless, a rule learning algorithm performs its own feature selection (by selecting promising conditions), thus, it may not be so severe to omit an external feature selection.

[9] Among others, the first name and last name or the matriculation number were removed as they have no impact on the class variable.

Table 8.6: Theory size and other statistics for different algorithms

algorithm	heuristic	*minCov*	# rules	# conds	# *fn*
RIPPER with pruning	h_{fgain}	2	1	2	28
RIPPER without pruning	h_{fgain}		11	34	22
PART with pruning	entropy & h_{cov}		10	20	28
PART without pruning	entropy & h_{cov}		23	49	26
SIMPLESECO	h_{acc}	1	13	28	24
	h_{corr}		6	16	26
	h_F		11	26	24
	h_ω		16	31	29
	h_{lap}		20	37	28
	h_{LR}		5	13	24
	h_m		5	16	22
	h_{prec}		18	35	24
	h_{c_r}		4	9	20
	h_{wra}		0	0	30
SIMPLESECO	h_{acc}	2	2	4	28
	h_{corr}		2	8	26
	h_F		6	16	24
	h_ω		3	6	32
	h_{lap}		2	4	34
	h_{LR}		5	13	24
	h_m		5	16	22
	h_{prec}		0	0	35
	h_{c_r}		4	9	20
	h_{wra}		0	0	30

8.2.3 Results

In the following some rule sets that were found by different algorithms on the created dataset are shown. In total, we tested the algorithms RIPPER [22] and PART [47] that are provided by *Weka* [177]. For both of them pruning was enabled and disabled. Additionally, we ran several configurations of the SECO-Framework, specifically variations of the SIMPLESECO algorithm (cf. Section 3.5.4). The algorithm was tested with the heuristics *accuracy*, *correlation*, F-Measure, *Klösgen measure*, *Laplace*, *Linear Regression*, m-estimate, *relative cost measure*, and *weighted*

r_1 $no_{pass} \leq 0 \wedge no_{wr} \leq 0 \rightarrow$ session=yes [12,16]

r_2 $no_{pass} \leq 0 \wedge no_{reg} \leq 1 \wedge hours_{pr} =$ 5-10 $\rightarrow$ session=yes [0,5]

r_3 $no_{pass} \leq 0 \wedge no_{per} \leq 3 \wedge subj_{much} =$ 'rather not applies' $\rightarrow$ session=yes [0,5]

r_4 $no_{pass} > 0 \wedge$ group=yes $\rightarrow$ session=no

r_5 $freq_l(HCS) =$ rarely $\rightarrow$ session=no

r_6 issues=yes $\wedge$ nervous='fully applies' $\rightarrow$ session=yes

r_7 $no_{reg} < 0 \wedge$ undertaken='does not apply' $\rightarrow$ session=yes [0,4]

r_8 $no_{per} < 1.5 \wedge subj_{much}$='rather not applies' $\rightarrow$ session=yes [0,3]

r_9 $no_{reg} < 0 \wedge$ fit='does not apply' $\rightarrow$ session=yes [1,7]

r_{10} group=no $\wedge$ issues=yes $\wedge$ $no_{insp} < 0 \rightarrow$ session=yes [2,7]

Figure 8.4: Selected rules

relative accuracy. For all of them the minimum coverage was set to one and to two example(s). For the parametrized heuristics, the optimal values derived in Section 4.2.2 were used, and the *Linear Regression* heuristic is the one derived in Section 4.3. Table 8.6 gives a summary of the size of the rule sets in terms of numbers and conditions. The focus lies on the rules themselves, as the main goal of the project is to construct a rule set that enables humans to get an idea what characteristics a student exhibits to fall into the critical category.

A Brief Discourse on Performance

Unfortunately, none of the algorithms was able to outperform the baseline. The dataset has 35 positive (`session = yes`) and 88 negative examples (`session = no`), thus when predicting "no" in all cases, the accuracy would be 71.54%. The best algorithm so far (SimpleSeCo with *Laplace* and mimimum coverage two) achieved 69.11% accuracy in a 10–fold cross-validation. Nevertheless, every term more data will be gathered which should improve the quality of the rule sets.

Note that currently the performance is not the main issue. More interesting at this point in time is what rules are found by the algorithms, and consequently what factors are indicative for a student's failure. The costs in this setting are also not uniformly. As it is more severe to miss a student who would benefit from a personal session than to invite a student to such a session who actually is not in need to be counseled, a false positive is not as serious as a false negative. The number of false negatives (*fn*) is inspected in more detail in the last column of Table 8.6.

As can be seen there, the heuristic *relative cost measure* has the lowest number of false negatives among all algorithms, but ranks at bottom for the predictive accuracy (only three algorithms were worse).

Interpretation of Selected Rules

In Figure 8.4 some interesting rules for the algorithms are shown. The coverage of the rules, where applicable, is shown in square brackets as known from Chapter 3. The rules presented in respect of the SimpleSeCo learners were found with a minimum coverage of two but note that the interesting ones do not differ to the configurations where the minimum coverage is set to one.

A single rule as found by Ripper in default configuration (rule r_1) clearly is not enough to describe the data sufficiently. When we inspect the coverage statistics it is obvious that this rule is suboptimal (covering 16 positive and twelve negative examples). The rules r_2 and r_3 are two examples of the theory of the unpruned version of Ripper, which contains eleven rules in total (where the rest of rules has a lower but at least positive coverage). Both rules seem to make sense. Interestingly, seven out of the eleven rules consist of the condition $no_{pass} \leq 0$ which emphasizes the importance of the number of passed exams.

Rule r_4 and r_5 are examples for the PART algorithmus in default configuration. The first rule makes sense as a special session usually is not necessary when the student has passed at least one exam and is already in a study group. Interestingly, the second rule is somewhat surprising as it states that students who have rarely participated in the *Human Computer Systems* lecture are no candidates for a special session. Due to coverage changes related to the separate-and-conquer strategy, rules may reflect characteristics that are not present in the original dataset, but still this is a somewhat counter-intuitive rule. Unfortunately, the coverage statistics given by PART are not directly comparable to the format used in this thesis. Note that the theory PART learns is different in that both classes are predicted by the rules where for all other algorithms only the smaller class (in this case `session=yes`) is used for prediction.

Another interpretation of rule r_5 is that the best students have no need to participate at the lecture. In general, this observation holds as often outstanding students do not attend lectures. The sixth rule found by an unpruned PART states that when the student has private issues and was nervous during the exam a special session is necessary. This can also be confirmed to be valid by experience.

To illustrate how different configurations of the SeCo-Framework may assist in finding different types of rules, some of the induced rules are listed in Figure 8.4 (rules $r_7, ..., r_{10}$). The advantage is that due to the configurability of the SeCo-Framework, rules with diverse characteristics can be found. Thus, as argued before, different heuristics yield different rules. Those may be biased towards shorter or longer rules or the total number of rules in a rule set may be influenced. Hence,

in Figure 8.4 some example rules are shown. Note that all of them have a minimum coverage of two. Rules r_7 and r_8 are found with the *Laplace* heuristic. The theory consisted only of these two rules. Essentially, the coverage of seven positive examples can also be achieved by a single rule. Using the heuristic *Klösgen measure*, a rule could be found that covers seven positive and one negative example. Interestingly, rule r_9 was the first rule in the theory, i.e., the *Klösgen measure* favors coverage more over consistency than *Laplace* does. Recall that a parameter setting of 0.4323 for the *Klösgen measure* effectively encodes a stronger bias towards *weighted relative accuracy* (cf. the isometrics of the *Klösgen measure* in Figure 2.13) which is known to underfit, i.e., to prefer coverage over consistency (cf. Section 4.2.3). Rule r_{10}, found with the *Linear Regression* is the fourth rule in the list. Still, it is an interesting rule as the properties "study group" and the text field "personal issues" proved to have a huge impact on the decision whether to invite students to a personal session or not.

8.3 Related Work

While there are many papers about cancer detection and prediction in general, including also a lot of work on using machine learning in the skin cancer domain, there is less research on dealing with the behavior of students in universities. Nevertheless, there are some papers and even an International Journal of Artificial Intelligence in Education[10] exists. Most of the published work focuses on student modeling. Nonetheless, the categorization of students into those that need assistance and those that do not is to some extent similar to student modeling as effectively also a model of the student is constructed albeit this model is much less complex. Most of the research on that topic is concerned with tutor systems or other systems that assist students during their studies (cf., e.g., the systems that can be found in the literature [14, 100, 5]). To reach that goal the main task is to derive or learn a student model [153]. Note that the main difference to existing research in both areas is that our work relies on questionnaires.

It was found out that the main research in the area of cancer has been focused on the detection of cancer rather than the prediction of it [25]. In that work, one observation was that about 1,500 papers were published regarding cancer detection while less than 120 papers deal with cancer prediction (as we did in Section 8.1). Despite the majority of work in this area is focused on cancer in general, there is research that is concentrated on skin cancer. Applying machine learning in the cancer domain has a long history (cf., e.g., [152, 112], or for an overview on the history and a summary of state-of-the-art [18, 101]). Interestingly this domain seems to be hard to learn as dermatologists are only slightly above 60% accuracy in detecting skin cancer [70]. Nevertheless, most of the learning algorithms rely on

[10] `http://iaied.org` (visited 2012-05-07)

image data [28, 40, 34, 92]. Also, it was shown that symbolic algorithms (in this case decision trees) do not perform well when relying on features based on images [25]. The authors ascribed that to the large number of numerical attributes. Another paper also includes some evidence that symbolic learning is not the right way [183]. Nevertheless, we think that the interpretability outweighs the performance (cf. Section 8.1.3).

8.4 Summary

The first case study deals with the problem of predicting skin cancer. Some of the rules proved to be interesting for the dermatologist. The SECO-Framework was able to serve as a toolkit for finding diverse rules, so that a great number of different rules were available during the project. For reasons of interpretability, rules were a perfect instrument to work with in this medical area. Despite, as the results also reveal, there is still a lot of work to do. Future research should be concentrated on improving the accuracy and especially the false positive rate has to be reduced. These steps are necessary to guarantee that the system can be used in practice.

In the second case study, rules describing the necessity to invite a student to a counseling session were presented. It was shown that algorithms provided by *Weka* [177] do not allow to be configured adequately. Instead, the SECO-Framework provided a large number of different configurations where the rule sets were also capturing different biases. For example, to minimize the false negatives in this domain, the *relative cost measure* proved to be the best choice. Nevertheless, the research can only be seen as a first step towards a system that can be actually employed. The main reason is that the predictive performance of all algorithms is poor. The lack of training data and the process of labeling the instances manually seem to be the main causes. The project still runs which means that in each new semester more data is available. Thus, we are confident that the system will improve over time.

9 Discussion of the Results

In this chapter we give a unifying view of the entire research that was conducted in this thesis and wrap up the different chapters. We identify several criteria a good rule learning heuristic has to fulfill. Then we describe how we have addressed these topics in this thesis.

The crucial step for the performance of a greedy covering algorithm is the choice of the rule evaluation heuristic. In this thesis, we have tried to empirically determine a good trade-off between consistency and completeness for classification and regression. Consistency and completeness are the most important criteria for evaluating the quality of a rule. Interestingly, the preference structure of heuristics for classification and regression is to a large extent similar (cf., e.g., the isometrics of the cost measure (Figure 4.3) and those of the regression measure depicted in Figure 6.3). We have investigated how heuristics that employ such an optimal trade-off support additional components of a rule learning algorithm such as an optimization phase or pruning (cf. Section 4.2.3). We have introduced a novel heuristic that was learned via meta-learning. The key difference of this approach was that no bias towards existing measures was imposed. The derived heuristic has a similar preference structure as the previous heuristics.

However, it is also important to keep in mind that a good rule must optimize or trade off various criteria simultaneously. Among them are:

Consistency: How many negative examples are covered?

> In concept learning or decision list learning, whenever a rule covers negative examples, these misclassifications cannot be corrected with subsequent rules. When sets of unordered rules are learned for multi-class problems, such corrections are possible, but obviously one should nevertheless try to find pure rules.

Completeness: How many positive examples are covered?

> Even though subsequent rules may cover examples that the current rule leaves uncovered, rules with higher coverage are typically preferred because they bring the learner closer to the goal of covering all positive examples. Additionally, rules with high coverage tend to be more reliable.

Gain: How good is the rule in comparison to other rules (e.g., default rule, predecessor rules)?

A rule with high consistency may nonetheless be bad if its predecessor had a higher consistency and vice versa. Thus, many heuristics, such as *wra*, or *foil gain* relate the quality of a rule to other rules.

Utility: How useful will the rule be in the context of the other rules in the theory?

A rule with high consistency and completeness may nevertheless be a bad addition to the current theory if it does not explain any new examples.

Bias: How will the quality estimate change on new examples?

It is well-known that estimates obtained on the training data will be optimistically biased. A good heuristic has to address this problem so that the algorithm will not overfit the training examples.

Potential: How close is the current rule to a good rule?

An incomplete rule, i.e., a rule that is encountered during the search for a good rule, should not be evaluated by its ability to discriminate between positive and negative examples, but by its potential to be refined into such a rule.

Simplicity: How complex or comprehensible is the rule?

In addition to its predictive quality, rules are often also assessed by their comprehensibility, because this is one of the key factors for preferring rule learning algorithms over competing inductive classification algorithms. As comprehensibility is difficult to measure, it is often equated with simplicity or rule length.

Our experiments in this thesis addressed many of this issues directly (consistency, completeness, gain, bias, potential). Some of these criteria are also addressed, at least to some extent, algorithmically. Utility, e.g., is addressed in part by the covering loop which removes examples that have been covered by previous rules. Thus, the context of past rules is taken into account. However, it is harder to take the context of rules that will be subsequently learned into account. This is particularly the case because most rule learning heuristics only focus on the examples covered by the rule, and not on the examples that are not covered by a rule. This is contrarily to decision tree learning heuristics, which consider all possible outcomes of a condition simultaneously.[1] The PART [47] algorithm may be viewed as an attempt to use the best of both worlds. Similarly, RIPPER's global optimization phase, where rules in a final theory are tentatively re-learned in the context of all previous and all subsequent rules, may be viewed as an attempt to address this issue.

[1] Consider, e.g., the difference between the information gain heuristic used in ID3 [131] and the information gain heuristic used in FOIL [132].

As we have seen above, exhaustive search often finds rules that have a higher coverage than hill-climbing. While the individual rules tend to have a higher quality than the low-coverage rules found by hill-climbing, as can be seen from Table 5.2, they also often perform worse in the context of a complete theory. We attribute this to two reasons, which we discuss in more detail below:

- the good performance of individual rules is often due to more generalization, and
- it is a fundamental problem that rules are optimized locally.

As we are working in a concept-learning scenario where we only learn rules for the positive class, negative examples that are covered by one rule cannot be excluded by learning additional rules. Many of the differences from theories learned by hill-climbing and exhaustive search are due to such high-coverage rules. While we have not thoroughly investigated this issue yet, the situation is reminiscent of the *small disjuncts problem* [79], which denotes the problem that on the one hand, a large part of the error of a rule-based theory can be attributed to rules with low coverage, but that, conversely, these rules are also necessary to maintain a certain minimum performance. For all of the regression rule learners we fixed the minimum coverage to three examples. While we did not concentrate on the small disjuncts problem there, restricting the rules to cover at least three examples in general seems to be a good choice. Then, small disjuncts with a coverage lower than three are omitted anyway.

A related issue is that for hill-climbing search it is beneficial to weight consistency more heavily than completeness. All of the three parametrized heuristics for classification do so (cf. Section 4.2). Note that the regression heuristic h_{cm} derived in Section 6.4.2 also prefers consistency over completeness. Albeit consistency is defined differently in this case, the general trend that false positives (or rules with a high *rrmse* in regression) should be omitted is also valid for regression. Nevertheless, when the regression task is reduced to classification, the weighting factor for regular classification should not be re-used for regression. This is confirmed by the rather bad performance the *relative cost measure* achieves when used for the regression task (cf. Section 7.3). Interestingly, the characteristics of consistency and completeness themselves seem to change when the statistics are derived by the reduction scheme. A detailed inspection of this phenomenon is subject to further work.

Rule learning heuristics are designed to induce a single rule that optimizes the quality criterion locally in the context of previously found rules. They do not aim to induce a rule that performs optimally in the context of the final rule set. The locality of this choice is maybe best visible in the stopping criterion described in Section 3.2.1, which only allows to add a rule to the theory if it covers more positives than negatives. The idea behind this criterion is that adding a rule that covers

more negatives than positives will decrease the global accuracy of the theory (on the training set). However, this does not imply that accuracy is additive in the sense that optimizing accuracy on each individual rule will optimize global accuracy. Thus, our exhaustive search algorithms are still greedy with respect to the goal of optimizing global performance of the whole rule set. A simple approach to address this problem is the optimization phase implemented in the RIPPER rule learning algorithm [22]. As the experiments in Section 5.5.3 have unfolded, sometimes the effects caused by such a stopping criterion can be severe. For example, on some datasets the bidirectional search ends up in considerably smaller theories than an exhaustive search because one suboptimal rule was found during the search. The interplay between the requirement that each refinement has to have a greater evaluation value than its predecessor and the global stopping criterion is the main reason for those situations.

Certainly, the most important conclusion is to realize that all the abovementioned criteria are not independent. For example, comprehensibility and simplicity are correlated with completeness: simple rules tend to be more general and to cover more examples. Thus, a bias for completeness will automatically correlate with a bias for shorter rules. This is at least true in a hill-climbing scenario. As we have observed in Section 5.4.2, when more sophisticated search algorithms are used, the picture changes. Here, contrarily to the observation for hill-climbing, longer rules may cover more examples and therefore can be also more general. To omit low coverage rules, the idea of the *Laplace*-correction as introduced in CN2 [19] was to on the one hand correct a too strong bias for consistency over completeness (by, e.g., penalizing pure rules that cover only single examples), and on the other hand also to try to provide more accurate probability estimates.

That these criteria are not independent was already noted by the authors of CN2, who observed that the importance of its rule significance test greatly diminished when *Laplace* is used as a search heuristic because, compared to entropy, it tends to favor general rules anyways [19]. In fact, in the README file to the implementation of the algorithm, one can read that the *Laplace* heuristic of the second version directly tries to estimate what the combination of entropy and significance test indirectly estimated, namely the expected performance of a rule on new test data, and that thus the *Laplace* heuristic is intended to replace both, the original entropy heuristic and the significance test. As we have seen in Section 5.4.2 another way to find general rules with high accuracy would be to employ more complex search algorithms. The results given in Table 5.2 also show that a single rule per dataset performs very well on its own. This mostly comes from the higher coverage these rules have in comparison to single rules found via a hill-climbing search.

Another result is that when the regression problem is reduced to classification (cf. Chapter 7), the behavior of the classification heuristics themselves changes. As noted in Section 7.3, biases of heuristics known from classification do not carry over to regression. Hence, *Laplace* has found smaller theories than *weighted rela-*

tive accuracy. This is most obvious when we compare the results of the dynamic approach with the static regression by classification (cf. Table 7.1). Here, *Laplace* has found approximately twelve times larger rule sets with the static approach compared to about three times smaller theories with the dynamic method. In this sense, requirements (and their interplay) that result from the analysis on classification problems may not be valid for regression scenarios and thus have to be re-evaluated. As the results for the *relative cost measure* show, it seems to be necessary to re-adjust the consistency/coverage trade-off derived in the classification scenario to make it suitable for regression.

As the experiments with the regression heuristics have shown (cf. Section 6.5), consistency and completeness are also different when they are weighted against each other in a single heuristic. Albeit the derived heuristic implements quite a similar preference structure as the tuned heuristics (cf. Figure 6.3 and 4.3, but also Figure 4.5), the two objectives are not exactly the same. In regression, the issue of completeness was also treated differently by fixing the number of examples that have to be covered by the rule set. A solid evaluation of this parameter is subject to future work.

In Chapter 8 the heuristics were used on real-world problems. The most important criterion there was simplicity as our primary interest was to have interpretable rule sets. Interestingly, most of the found rules were short and had appropriate accuracy. As expected, there is no unique best heuristic. This mostly results from the no free lunch theorem [179] and is even more evident in the real-world setting. As sketched above, heuristics have to incorporate many goals simultaneously. Usually, each domain has its own peculiarities. Where accuracy was the main objective in the empirical experiments of the previous chapters, the characteristics of the rules themselves are more important in real-world scenarios. Hence, an optimal setting of the parameter of the heuristic is different than before as the domain and the objectives have changed. In essence, as claimed by the no free lunch theorem, an unique optimal setting is never achievable. Nevertheless, the presented combinations of the criteria in form of various derived heuristics have enabled the induction of comprehensible and accurate rule sets in real-world domains.

As a conclusion of the discussion we think that it is unclear whether all the abovementioned criteria can or should be addressed simultaneously with a single rule learning heuristic. Answering this question is even beyond the scope of this thesis (in fact, we do believe that these problems should be separated and addressed individually). However, it is the case that common rule learning algorithms essentially assume that all these objectives can be captured into a single heuristic function (only overfitting is frequently addressed by using a separate criterion). Thus, it is a valid and interesting question how good a greedy rule learner can get under this assumption. The work reported in this thesis may be viewed as a contribution to answer this question.

10 Conclusions and Future Work

In this chapter the thesis is concluded. Future work on all the different aspects addressed in this thesis is also given here.

10.1 Conclusions

In Chapter 3 the rule learning framework SECO was introduced. This framework is based on a generalization of the separate-and-conquer strategy given in [55]. It was shown that some well-known separate-and-conquer rule learning algorithms can be instantiated in the SECO-Framework by simply specifying some of the general building blocks via an *XML*-file. These building blocks are the main components of the framework that enable the generalization. Due to the versatility of the framework most rule learning algorithms that are based on the separate-and-conquer strategy can be easily implemented in the framework. All models of these algorithms can be re-used which greatly diminishes the effort that is necessary to implement new algorithms. In this thesis, the framework was also used to evaluate the different algorithms using the evaluation package of the SECO-Framework. To our knowledge, the SECO-Framework is the first concrete implementation of a general framework for rule learning.

The experimental study reported in Chapter 4 has provided several important insights into the behavior of greedy inductive rule learning algorithms.

First, we think that this has been the most exhaustive experimental comparison of different rule learning heuristics to date. We tested five parameter-free heuristics, five parametrized heuristics with a large number of parametrizations, and several different meta-learning scenarios. The results confirm various previously known findings (e.g., *precision* and *Laplace* overfit, whereas *accuracy* and *weighted relative accuracy* over-generalize), but also yielded new insights into their comparative performance. In particular, we have determined suitable default values for commonly used parametrized evaluation metrics such as the m-estimate. This is of considerable practical importance, as we showed that these new values outperformed conventional search heuristics and performed comparably to the RIPPER rule learning algorithm. On the other hand, however, we have also seen some indication that these values capture aspects of our algorithm that may not fully transfer to other rule learning algorithms.

Second, our results also let us draw important conclusions about what factors influence a good performance of a rule learning heuristic. For example, we found

that heuristics which take the *a priori* class distribution into account (e.g., by evaluating relative coverage instead of absolute coverage) will in general outperform heuristics that ignore the class distribution (e.g., the F-measure which trades off *recall* and *precision*). This is also confirmed by the high weight that this parameter receives in our meta-learned heuristics. *Gain*-heuristics, which take this one step further by considering the distribution of the predecessor rule as a prior class distribution for each individual addition to a rule, may be even more preferable, but this still needs to be investigated more thoroughly.

We also found that for a good overall performance, it is necessary to prefer consistency over coverage, i.e., to weight the false positive rate more heavily than the true positive rate. We can most clearly observe this bias towards minimizing the false positive rate in the optimal parameter value for the *relative cost measure*, but it can also be observed in other well-performing heuristics whose isometrics have a very steep slope in the important regions. In the experiments with meta-learning and in the good performance of *correlation* we can also observe that heuristics perform better if they increase the emphasis on this aspect for rules with high consistency.

This result may also be interpreted as evidence that a good heuristic has to adapt to the characteristics of the algorithm in which it is used. In our case, this bias towards consistency seems to be a desirable property for a heuristic that is used in a covering algorithm, where incompleteness (not covering all positive examples) is less severe than inconsistency (covering some negative examples), because incompleteness can be corrected by subsequent rules, whereas inconsistency cannot (at least not in a concept learning scenario). This dependency on the dynamics of the algorithm is also confirmed by one of the results of the meta-learning study, in which we observed that training on the test-set performance of the candidate rule is somewhat less efficient than training on the performance of its best refinement. Finally, the results of the transfer of the heuristics to other rule learning implementations also seem to confirm that they capture individual characteristics of the algorithm.

The main conclusion that we draw from the experiments reported in Chapter 5 is that oversearching is not a universal phenomenon. Depending on the search heuristic and the characteristics of the dataset, we may also observe that increasing the search effort may considerably improve the performance of a rule learner. We have observed oversearching primarily for heuristics where hill-climbing performed generally well, such as for the three heuristics that were optimized for hill-climbing search [85, 88]. For those, hill-climbing performed best, and the performance decreased steadily and substantially when used with increasing beam widths, as is predicted by the oversearching phenomenon. It should be noted that among the heuristics that we looked at, there was no counter-part that performed as well in exhaustive search as these heuristics performed in hill-climbing. However, the

performance of other heuristics (such as *precision* or *odds ratio*) did not suffer from exhaustive search or even improved considerably with increased search efforts.

These diverse results are maybe most obvious when we compare the performance of *precision* and *Laplace*, which differ greatly in their performance in a hill-climbing search, but perform almost identically when used in exhaustive search. Thus, it seems that the prime motivation that lead to the introduction of the *Laplace* heuristic in CN2 [19, 125], namely the ability to correct for noise in the data, does not seem to be the key factor for its improved performance (otherwise we should be able to observe this performance difference with exhaustive search as well).

A key finding that helps to understand these results is that exhaustive search learns fewer, but substantially longer rules. This means that the longer rules found by exhaustive search are nevertheless often more general than the shorter rules found by a hill-climbing search. In some cases, this leads to a better performance, whereas in other cases it leads to over-generalization. In particular, we observed for all heuristics that the first rule found by exhaustive search is typically both, longer and more accurate than the first rule found by hill-climbing search. Thus, the main problem that causes the loss in accuracy seems to be in the interplay of rule induction and the covering loop.

Note that our experiments have not addressed a fundamental problem in separate-and-conquer rule learning, namely that rules have to be learned by optimizing a local quality criterion, whereas they will be used in the context of an entire theory. While this holds for both, hill-climbing and exhaustive search, it may nevertheless have affected our results: more general rules will often have a higher local evaluation than more specific rules, but those weaker rules may perform better in a complete theory (e.g., because they cover fewer negative examples). It is still an open question how to propagate a global quality criterion back to local optimization problems [98].

An approach to adjust the learned rules to the context of the whole theory is implemented in the RIPPER algorithm. It uses an optimization phase to re-learn all rules in the theory at a stage where the rule set is already induced. As the experiments of Section 5.5.3 have shown, a bidirectional search is unable to yield theories that are comparable to those found by RIPPER. Albeit suboptimal decisions can be reversed with a bidirectional search, the optimization phase of RIPPER is a better choice. Interestingly, the bidirectional search also is unable to outperform a simple top-down approach in most cases (cf. Table 5.4). On the other hand, there are particular datasets where a bidirectional search works comparable to an exhaustive search but with notably less computational effort. Nevertheless, the performance achieved by the bidirectional search is somewhat in contrast to what we have expected. We are still investigating why this is the case.

In Chapter 6 a new rule learning algorithm for the task of regression was presented. It was shown that the algorithm performs comparable to different state-of-the-art algorithms implemented in *Weka* and REGENDER, a rather new algorithm.

A new splitpoint generation method was introduced. This method improves the quality of candidate rules and even results in lower runtimes compared to naive methods as the generation of equidistant or all splitpoints. Nevertheless, the number of generated candidate rules directly depends on the number of splitpoints. But as shown in the experiments, at least for one configuration of the algorithm, a number of three splitpoints per numerical attribute was already sufficient.

Additionally, a novel rule learning heuristic was introduced that clearly improves the algorithms performance due to its flexibility in weighting the error of a rule with its coverage. An optimal setting for this regression rule heuristic was presented and it proved to be stable since the parameter values obtained in two different optimizations were nearly the same. An interesting observation is that, as known from classification, in regression the rule's consistency should be preferred over its coverage.

In Chapter 7, we proposed a new technique for the heuristic learning of regression rules. The key idea is to dynamically transform the regression approach into a classification setting by defining positive examples near the current mean as positive, and those further away as negative. In this way, classification rule heuristics can be directly used for learning regression rules. The algorithm clearly outperforms a static discretization using the same rule learner, and performs at least equally to other state-of-the-art regression rule learning algorithms. While it does not reach the performance of other algorithms that use more expressive models, we also showed that the use of bagging results in an algorithm that is en par with other rule-based regression ensembles and statistical regression techniques.

In Chapter 8 two real-world problems were introduced. It was shown that by using the SECO-Framework, diverse rules could be found which proved to be meaningful in both domains. As often the rules themselves are the interesting elements of the system and only partly the performance of them, one of the main objectives of rule learning algorithms should be to allow for configuration of the system and therefore offering a great variety of different rules. Most of the rule learners rather provide only a small set of configurations that results in merely a few different rules sets (even if these have a good performance). Hence, as existing rule learning algorithms often only have a constrained configurability (cf. RIPPER [22], or PART [47] in their *Weka*-implementations), the SECO-Framework nearly can be configurated in all possible dimensions a rule learner consists of.

In Section 8.1 the goal was to predict skin cancer risk. Albeit the rules are not precise enough to be actually used in this sensitive domain, some interesting ones could be found. One of the main advantages during this project was that the dermatologist was able to compare the rules he uses himself for categorizing patients to those found by the SECO-Framework.

The second case study dealt with the problem of identifying students that need assistance. Unfortunately no algorithm was able to outperform the baseline. The main reason is that there is not enough data available at this point in time. Nevertheless, the presented work can be seen as a first step towards a system that can be actually used in the future.

10.2 Future Work

In the following, directions for future work on the topics presented in this thesis are given.

10.2.1 SeCo-Framework

Certainly, the framework currently does not cover all separate-and-conquer algorithms. For example, stochastic search algorithms are unsupported right now. But, due to the modular design of the software, these could be realized in a simple way. For instance, to include a stochastic search only two interfaces have to be implemented. In the future we plan to integrate additional search algorithms as well as a graphical user interface.

Currently, the evaluation package is mainly used to evaluate rule learning algorithms. In the future we plan to enrich this package so that it becomes the main means for evaluation purposes for different data mining problems. Among them are multilabel classification, label/object ranking, ordinal classification, hierarchical classification, and preference learning. Some of them can also be combined, e.g., one can perform a label ranking for instances that have multiple classes assigned (multilabel). The key step here is to define proper interfaces so that an extension of the framework could be easily achieved without changing the whole structure of the framework. Then, all necessary features can be implemented one by one.

10.2.2 Classification Heuristics

The results presented in Chapter 4 also have their limitations. For example, we have only evaluated the overall performance over a wide variety of datasets. Obviously, we can expect a better performance if the parameter values are tuned to each individual dataset. We think that the good performance of Ripper is due to the flexibility of post-pruning, which allows to adjust the level of generality of a rule to the characteristic of a particular dataset. We have deliberately ignored the possibility of pruning for this set of experiments, because our goal was to gain a principal understanding of what constitutes a good rule evaluation metric for

separate-and-conquer learning. It is quite reasonable to expect that pruning strategies could further improve this performance. In particular, it can be expected that the performance of parameter values that result in slight overfitting can be considerably improved by pruning (whereas pruning can clearly not help in the case of over-generalization). We are currently investigating the interplay of pruning and learning in more detail.

10.2.3 Search Algorithms

As we might conclude from the results of Chapter 5, search heuristics for rule learning algorithms have to address several goals simultaneously: on the one hand, they have to estimate a rule's predictive quality, on the other hand, they have to evaluate the rule's potential for being refined into a rule that has a high predictive quality. However, with increasing search efforts, the importance of the latter point decreases, because the chances that high-quality rules will be found without guidance of the search increase. Thus, we think that good heuristics for exhaustive search have different requirements than good heuristics for hill-climbing search. Most of the efforts in inductive rule learning have been devoted only to the latter problem, whereas we would argue that finding a suitable metric for exhaustive rule induction is still an open problem. We plan to address this in a similar way as had been previously performed for hill-climbing search in Chapter 4. As soon as this has been solved, we can address the second step, which is to clearly separate the search heuristic and the rule evaluation metric in inductive rule learning algorithms.

An interesting open question is how these results relate to rule algorithms as CBA [110] that employ an APRIORI-like exhaustive search for exhaustively searching for *rules that satisfy certain constraints*. For example, it seems reasonable to expect that the minimum support constraint that is typically used will avoid some overfitting behavior, but a systematic exploration of this issue is still pending.

10.2.4 Regression Rule Learning

The runtime of the algorithm presented in Chapter 6 still needs a lot of improvement. One way to do this could be to introduce error bounds that make a so-called forward pruning possible (cf. Section 3.2.1). This method enables the algorithm to stop the refinement of a rule when it is clear that it could not outperform a virtual best refinement. In classification this can be easily implemented, in regression however it is unclear how this could be done.

Another simple improvement would be to use some kind of pruning functionality to keep the theory size small. Strategies like I-REP (cf. [61]) that is used in the RIPPER-algorithm [22] seem to be promising possibilities to do so. The method for

prediction could also be altered by using an unordered mode of the algorithm that makes it comparable to algorithms like REGENDER [30], which also uses several rules to predict a single numerical target value.

The dynamic reduction presented in Chapter 7 can also be improved in several ways. We expect that a more systematic investigation of its parameters such as the factor applied to the standard deviation, or the percentage of examples that need to be covered by rules can yield a better performance. The current setting of 90% is rather intuitive and perhaps not the best choice. Similarly, a more systematic evaluation of classification rule heuristics in the context of regression tasks that are tackled by employing the reduction scheme could lead to better performance. Other works [104, 60] show that different heuristics are suited for different applications, and it is still not well explored which heuristic is the best choice. In the case of parametrized heuristics, such as the *relative cost metric*, the parameter can also be tuned to optimize the performance on the datasets in the same way as suggested in Section 4.2.1.

A promising path to optimize the predictive performance of the two algorithms would be to replace the constant predictions in the rule heads with linear models. The results of M5RULES, where we included a version with constant predictions and a version with linear models into the experimental evaluation, show that this should lead to drastic performance improvements. However, much of the interpretability of the rule set would be lost when doing so.

10.2.5 Real-World Applications

For the prediction of the skin cancer risk the main challenge is to design rule learning algorithms that are able to omit predicting a person as healthy when the person actually has skin cancer. There is some work in the area of cost-sensitive classification in medical domains. A fruitful combination of these approaches could be a beneficial next step.

For the student assistance system more data have to be gathered. Solid statements about the performance can only be done when the variance is reduced. However, as each semester new data are available, the rules quickly will get better.

Bibliography

[1] R. Agrawal, T. Imieliński, and A. Swami. Mining Association Rules between Sets of Items in Large Databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data (COMAD-93)*, pages 207–216, New York, NY, USA, 1993. ACM.

[2] H. Akaike. A new Look at the Statistical Model Selection. *IEEE Transactions on Automatic Control*, 19(6):716–723, 1974.

[3] E. L. Allwein, R. E. Schapire, and Y. Singer. Reducing Multiclass to Binary: A Unifying Approach for Margin Classifiers. *Journal of Machine Learning Research*, 1:113–141, December 2000.

[4] B. Apolloni, A. Esposito, D. Malchiodi, C. Orovas, G. Palmas, and J. G. Taylor. A General Framework for Learning Rules from Data. *IEEE Transactions on Neural Networks*, 15(6):1333–1349, November 2004.

[5] P. Baffes and R. Mooney. Refinement-Based Student Modeling and Automated Bug Library Construction. *International Journal of Artificial Intelligence in Education*, 7:75–116, January 1996.

[6] R. J. Bayardo,Jr. and R. Agrawal. Mining the Most Interesting Rules. In *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-97)*, pages 145–154, 1999.

[7] L. Breiman, J. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth & Brooks, Pacific Grove, CA, 1984.

[8] E. W. Breitbart, A. Wende, P. Mohr, R. Greinert, and B. Volkmer. Hautkrebs. *Gesundheitsberichterstattung des Bundes*, Heft 22, 2004.

[9] S. Brin, R. Motwani, and C. Silverstein. Beyond Market Baskets: Generalizing Association Rules to Correlations. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (COMAD-97)*, pages 265–276, 1997.

[10] J. S. Bruner, J. J. Goodnow, and G. A. Austin. A Study of Thinking. *Science Editions, New York*, 1967.

[11] C. A. Brunk and M. J. Pazzani. An Investigation of Noise-Tolerant Relational Concept Learning Algorithms. In *Proceedings of the 8th International Workshop on Machine Learning (ML-91)*, pages 389–393, Evanston, Illinois, 1991.

[12] W. Buntine and T. Niblett. A Further Comparison of Splitting Rules for Decision-Tree Induction. *Machine Learning*, 8(1):75–85, 1992.

[13] S. Burges. Meta-Lernen einer Evaluierungs-Funktion für einen Regel-Lerner. Diploma thesis, TU Darmstadt, 2006. In German (English title: *Meta-Learning of an Evaluation Function for a Rule Learner*).

[14] R. R. Burton. Diagnosing bugs in a simple procedural skill. In D. H. Sleeman and J. S. Brown, editors, *Intelligent Tutoring Systems*, pages 157–184. Academic Press, 1982.

[15] B. Cestnik. Estimating Probabilities: A Crucial Task in Machine Learning. In L. Aiello, editor, *Proceedings of the 9th European Conference on Artificial Intelligence (ECAI-90)*, pages 147–150, Stockholm, Sweden, 1990. Pitman.

[16] C.-C. Chang and C.-J. Lin. LIBSVM: A Library for Support Vector Machines. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm` (visited on 2012-05-03), 2001.

[17] Y. Chevaleyre and J.-D. Zucker. A Framework for Learning Rules from Multiple Instance Data. In *Proceedings of the 12th European Conference on Machine Learning (ECML-01)*, pages 49–60. Springer-Verlag, 2001.

[18] D. V. Cicchetti. Neural Networks and Diagnosis in the Clinical Laboratory: State of the Art. *Clinical Chemistry*, 38(1):9–10, 1992.

[19] P. Clark and R. Boswell. Rule Induction with CN2: Some Recent Improvements. In *Proceedings of the 5th European Working Session on Learning (EWSL-91)*, pages 151–163, Porto, Portugal, 1991. Springer-Verlag.

[20] P. Clark and T. Niblett. The CN2 Induction Algorithm. *Machine Learning*, 3(4):261–283, 1989.

[21] W. W. Cohen. Efficient Pruning Methods for Separate-and-conquer Rule Learning Systems. In *Proceedings of the 13th International Joint Conference in Artificial Intelligence (IJCAI-93)*, pages 988–994, Chamberry, France, 1993.

[22] W. W. Cohen. Fast Effective Rule Induction. In *Proceedings of the 12th International Conference on Machine Learning (ICML-95)*, pages 115–123, Tahoe City, California, USA, 1995. Morgan Kaufmann.

[23] W. W. Cohen and Y. Singer. A Simple, Fast, and Effective Rule Learner. In *Proceedings of the 16th National Conference on Artificial Intelligence and the 11th Innovative Applications of Artificial Intelligence Conference (AAAI-99/IAAI-99)*, pages 335–342, Menlo Park, CA, USA, 1999. American Association for Artificial Intelligence.

[24] C. Cortes and V. Vapnik. Support-Vector Networks. *Machine Learning*, 20(3):273–297, 1995.

[25] J. A. Cruz and D. S. Wishart. Applications of Machine Learning in Cancer Prediction and Prognosis. *Cancer Informatics*, 2:59–77, 2007.

[26] L. De Raedt. *Interactive Theory Revision: An Inductive Logic Programming Approach*. Academic Press Ltd., London, UK, 1992.

[27] L. De Raedt and M. Bruynooghe. Indirect Relevance and Bias in Inductive Concept-learning. *Knowledge Acquisition*, 2:365–390, November 1990.

[28] C. Decaestecker, I. Salmon, O. Dewitte, I. Camby, P. Van Ham, J. L. Pasteels, J. Brotchi, and R. Kiss. Nearest-neighbor Classification for Identification of Aggressive versus Nonaggressive Low-grade Astrocytic Tumors by Means of Image Cytometry-generated Variables. *Journal of Neurosurgery*, 86(3):532–537, 1997.

[29] K. Dembczyński, W. Kotłowski, and R. Słowiński. A General Framework for Learning an Ensemble of Decision Rules. In *From Local Patterns to Global Models (LEGO-08), ECML/PKDD Workshop*, 2008.

[30] K. Dembczyński, W. Kotłowski, and R. Słowiński. Solving Regression by Learning an Ensemble of Decision Rules. In *Proceedings of the 7th International Conference on Artificial Intelligence and Soft Computing (ICAISC-08)*, pages 533–544, Berlin, Heidelberg, 2008. Springer-Verlag.

[31] D. Demšar. Obravnavanje numericnih problemov z induktivnim logicnim programiranjem. Master's thesis, Faculty of Computer and Information Science, University of Ljubljana, Slovenia, 1999. In Slovene.

[32] J. Demšar. Statistical Comparisons of Classifiers over Multiple Data Sets. *Machine Learning Research*, Jan(7):1–30, 2006.

[33] J. Demšar and B. Zupan. From Experimental Machine Learning to Interactive Data Mining. White paper available at `http://orange.biolab.si/wp/orange.pdf` (visited on 2012-05-03), 2004.

[34] N. J. Dhinagar and M. Celenk. Non-invasive Detection and Classification of Skin Cancer from Visual and Cross-sectional Images. In *Proceedings of*

the 4th International Symposium on Applied Sciences in Biomedical and Communication Technologies (ISABEL-11), pages 94:1–94:7, New York, NY, USA, 2011. ACM.

[35] T. G. Dietterich. Approximate Statistical Tests for Comparing Supervised Classification Learning Algorithms. *Neural Computing*, 10:1895–1923, October 1998.

[36] T. G. Dietterich and G. Bakiri. Solving Multiclass Learning Problems via Error-Correcting Output Codes. *Journal of Artificial Intelligence Research*, 2:263–286, January 1995.

[37] P. Domingos. The Role of Occam's Razor in Knowledge Discovery. *Data Mining and Knowledge Discovery*, 3(4):409–425, 1999.

[38] J. Du. Iterative Optimization of Rule Sets. Master's thesis, TU Darmstadt, 2010.

[39] B. Efron, T. Hastie, L. Johnstone, and R. Tibshirani. Least Angle Regression. *Annals of Statistics*, 32(2):407–499, 2004.

[40] E. A. el Kwae, J. E. Fishman, M. J. Bianchi, P. M. Pattany, and M. R. Kabuka. Detection of Suspected Malignant Patterns in three-dimensional Magnetic Resonance Breast Images. *Journal of Digital Imaging: The Official Journal of the Society for Computer Applications in Radiology*, 11(2):83–93, 1998.

[41] L. Fausett. *Fundamentals of Neural Networks: Architectures, Algorithms, and Applications*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1994.

[42] D. Fensel and M. Wiese. Refinement of Rule Sets with JoJo. In *Proceedings of the 6th European Conference on Machine Learning (ECML-93)*, pages 378–383, 1993.

[43] D. Fensel and M. Wiese. From JoJo to Frog: Extending a bi-directional Search Strategy to a more flexible three-directional Search. In *Beiträge zum 7. Fachgruppentreffen Maschinelles Lernen, Kaiserslautern*, 1994.

[44] D. Fischer. Maschinelles Lernen zur Hautkrebs-Vorhersage. Bachelor's thesis, TU Darmstadt, 2011. In German (English title: *Using Machine Learning to predict Skin Cancer*).

[45] P. A. Flach. The Geometry of ROC Space: Understanding Machine Learning Metrics through ROC Isometrics. In T. Fawcett and N. Mishra, editors, *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 194–201. AAAI Press, 2003.

[46] A. Frank and A. Asuncion. UCI Machine Learning Repository. Repository available at `http://archive.ics.uci.edu/ml` (visited on 2012-05-03), 2010.

[47] E. Frank and I. H. Witten. Generating Accurate Rule Sets Without Global Optimization. In J. Shavlik, editor, *Proceedings of the 15th International Conference on Machine Learning (ICML-98)*, pages 144–151, Madison, Wisconsin, 1998. Morgan Kaufmann.

[48] Y. Freund and R. 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.

[49] J. H. Friedman. Greedy Function Approximation: A Gradient Boosting Machine. *The Annals of Statistics*, 29(5):1189–1232, 2001.

[50] J. H. Friedman. Stochastic Gradient Boosting. *Computational Statistics And Data Analysis*, 38(4):367–378, 2002.

[51] J. H. Friedman and B. E. Popescu. Predictive Learning via Rule Ensembles. *Annals Of Applied Statistics*, 2(3):916–954, 2008.

[52] J. Fürnkranz. FOSSIL: A Robust Relational Learner. In F. Bergadano and L. De Raedt, editors, *Proceedings of the 7th European Conference on Machine Learning (ECML-94)*, volume 784 of *Lecture Notes in Artificial Intelligence*, pages 122–137, Catania, Italy, 1994. Springer-Verlag.

[53] J. Fürnkranz. Top-Down Pruning in Relational Learning. In A. Cohn, editor, *Proceedings of the 11th European Conference on Artificial Intelligence (ECAI-94)*, pages 453–457, Amsterdam, The Netherlands, 1994. John Wiley & Sons.

[54] J. Fürnkranz. Pruning Algorithms for Rule Learning. *Machine Learning*, 27(2):139–171, 1997.

[55] J. Fürnkranz. Separate-and-Conquer Rule Learning. *Artificial Intelligence Review*, 13(1):3–54, February 1999.

[56] J. Fürnkranz. Round Robin Classification. *Journal of Machine Learning Research*, 2:721–747, 2002.

[57] J. Fürnkranz. Modeling Rule Precision. In J. Fürnkranz, editor, *Proceedings of the ECML/PKDD-04 Workshop on Advances in Inductive Rule Learning*, pages 30–45, Pisa, Italy, 2004.

[58] J. Fürnkranz. From Local to Global Patterns: Evaluation Issues in Rule Learning Algorithms. In K. Morik, J.-F. Boulicaut, and A. Siebes, editors, *Local Pattern Detection*, pages 20–38. Springer-Verlag, 2005.

[59] J. Fürnkranz and P. A. Flach. An Analysis of Stopping and Filtering Criteria for Rule Learning. In J.-F. Boulicaut, F. Esposito, F. Giannotti, and D. Pedreschi, editors, *Proceedings of the 15th European Conference on Machine Learning (ECML-04)*, volume 3201 of *Lecture Notes in Artificial Intelligence*, pages 123–133, Pisa, Italy, 2004. Springer-Verlag.

[60] J. Fürnkranz and P. A. Flach. ROC 'n' Rule Learning—Towards a Better Understanding of Covering Algorithms. *Machine Learning*, 58(1):39–77, January 2005.

[61] J. Fürnkranz and G. Widmer. Incremental Reduced Error Pruning. In W. Cohen and H. Hirsh, editors, *Proceedings of the 11th International Conference on Machine Learning (ICML-94)*, pages 70–77, New Brunswick, NJ, 1994. Morgan Kaufmann.

[62] B. R. Gaines. System Identification, Approximation and Complexity. *International Journal of General Systems*, 3:145–174, 1977.

[63] B. R. Gaines. An Ounce of Knowledge is worth a ton of Data: Quantitative Studies of the Trade-Off between Expertise and Data based on Statistically Well-Founded Empirical Induction. In *Proceedings of the 6th International Workshop on Machine Learning (ML-89)*, pages 156–159. Morgan Kaufmann, 1989.

[64] B. R. Gaines. Structured and Unstructured Induction with EDAGs. In *Proceedings of the 1st International Conference on Knowledge Discovery and Data Mining (KDD-95)*, pages 124–129, Montreal, Canada, 1995. AAAI Press.

[65] B. R. Gaines. Transforming Rules and Trees into Comprehensible Knowledge Structures. In *Advances in Knowledge Discovery and Data Mining*, pages 205–226. MIT Press, 1996.

[66] D. Gamberger and N. Lavrač. Confirmation Rule Sets. In D. A. Zighed, J. Komorowski, and J. Żytkow, editors, *Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD-00)*, volume 1910 of *Lecture Notes in Artificial Intelligence*, pages 34–43, Lyon, France, September 2000. Springer, Berlin.

[67] S. Geman, E. Bienenstock, and R. Doursat. Neural Networks and the Bias/Variance Dilemma. *Neural Computing*, 4(1):1–58, 1992.

[68] T. V. Gestel, J. A. K. Suykens, B. Baesens, S. Viaene, J. Vanthienen, G. Dedene, B. D. Moor, and J. Vandewalle. Benchmarking Least Squares Support Vector Machine Classifiers. *Machine Learning*, 54(1):5–32, 2004.

[69] A. Giacometti, E. K. Miyaneh, P. Marcel, and A. Soulet. A Framework for Pattern-Based Global Models. In *Proceedings of the 10th International Conference on Intelligent Data Engineering and Automated Learning (IDEAL-09)*, pages 433–440, Berlin, Heidelberg, 2009. Springer-Verlag.

[70] C. M. Grin, A. W. Kopf, B. Welkovich, R. S. Bart, and M. J. Levenstein. Accuracy in the Clinical Diagnosis of Malignant Melanoma. *Archives of Dermatology*, 126(6):763–766, 1990.

[71] P. D. Grünwald. *The Minimum Description Length Principle (Adaptive Computation and Machine Learning)*. The MIT Press, 2007.

[72] K. Gurney. *An Introduction to Neural Networks*. Taylor & Francis, Inc., Bristol, PA, USA, 1997.

[73] I. Guyon and A. Elisseeff. An Introduction to Variable and Feature Selection. *Journal of Machine Learning Research*, 3:1157–1182, March 2003.

[74] T. Hammerling. Speicherexplosion für E-Mail und Fileserver – Platzproblem oder Strategiefrage? Presentation available at `http://voi.contextmatters.de/wp-content/uploads/2011/04/thorsten_hammerling.pdf` (visited on 2011-08-24).

[75] J. A. Hartigan and M. A. Wong. A K-Means Clustering Algorithm. *Applied Statistics*, 28:100–108, 1979.

[76] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 3rd edition, 2007.

[77] J. Hipp, U. Güntzer, and G. Nakhaeizadeh. Algorithms for Association Rule Mining — A general Survey and Comparison. *ACM Special Interest Group on Knowledge Discovery and Data Mining (SIGKDD) Explorations*, 2:58–64, June 2000.

[78] G. Holmes, M. Hall, and E. Frank. Generating Rule Sets from Model Trees. In *Proceedings of the 12th Australian Joint Conference on Artificial Intelligence (AI-99)*, pages 1–12. Springer, 1999.

[79] R. Holte, L. Acker, and B. Porter. Concept Learning and the Problem of Small Disjuncts. In *Proceedings of the 13th International Joint Conference in Artificial Intelligence (IJCAI-89)*, pages 813–818, IJCAI-89-address, 1989. Morgan Kaufmann.

[80] R. C. Holte. Very Simple Classification Rules Perform Well on Most Commonly Used Datasets. *Machine Learning*, 11(1):63–91, 1993.

[81] J. Hühn. *Induction and Fuzzification of Classification Rules*. Cuvillier, 2009.

[82] R. L. Iman and J. M. Davenport. Approximations of the critical region of the Friedman statistic. *Communications in Statistics - Theory and Methods*, A9:571–595, 1980.

[83] G. M. James. Variance and Bias for General Loss Functions. *Machine Learning*, 51(2):115–135, 2003.

[84] F. Janssen. Eine Untersuchung des Trade-Offs zwischen Precision and Coverage bei Regel-Lern-Heuristiken. Diploma thesis, TU Darmstadt, 2006. In German (English title: *On Trading Off Consistency and Coverage in Inductive Rule Learning*).

[85] F. Janssen and J. Fürnkranz. On Meta-Learning Rule Learning Heuristics. In *Proceedings of the 7th IEEE Conference on Data Mining (ICDM-07)*, pages 529–534, Omaha, NE, 2007.

[86] F. Janssen and J. Fürnkranz. An Empirical Investigation of the Trade-Off Between Consistency and Coverage in Rule Learning Heuristics. In T. Horvath, J.-F. Boulicaut, and M. Berthold, editors, *Proceedings of the 11th International Conference on Discovery Science (DS-08)*, pages 40–51, Budapest, Hungary, 2008. Springer Verlag.

[87] F. Janssen and J. Fürnkranz. A Re-evaluation of the Over-Searching Phenomenon in Inductive Rule Learning. In *Proceedings of the SIAM International Conference on Data Mining (SDM-09)*, pages 329–340, Sparks, Nevada, 2009.

[88] F. Janssen and J. Fürnkranz. On the quest for optimal rule learning heuristics. *Machine Learning*, 78(3):343–379, March 2010. DOI 10.1007/s10994-009-5162-2.

[89] F. Janssen and J. Fürnkranz. Separate-and-conquer Regression. In *Proceedings of the German Workshop on Lernen, Wissen, Adaptivität - LWA2010*, pages 81–89, 2010.

[90] F. Janssen and J. Fürnkranz. Heuristic Rule-Based Regression via Dynamic Reduction to Classification. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI-11)*, pages 1330–1335, Barcelona, Spain, 2011.

[91] N. Japkowicz and M. Shah. *Evaluating Learning Algorithms: A Classification Perspective*. Cambridge University Press, New York, NY, USA, 2011.

[92] T. M. Jørgensen, A. Tycho, M. Mogensen, P. Bjerring, and G. B. E. Jemec. Machine-learning Classification of Non-melanoma Skin Cancers from Image Features obtained by Optical Coherence Tomography. *Skin Research and Technology*, 14(3):364–369, 2008.

[93] J. Kalbfleisch. *Probability and Statistical Inference*. Universitext. Springer, 1979.

[94] A. Karalič and I. Bratko. First Order Regression. *Machine Learning*, 26(2-3):147–176, 1997.

[95] A.-C. Karpf. Bidirectional Rule Learning. Bachelor's thesis, TU Darmstadt, 2010.

[96] W. Klösgen. Problems for Knowledge Discovery in Databases and their Treatment in the Statistics Interpreter Explora. *International Journal of Intelligent Systems*, 7:649–673, 1992.

[97] W. Klösgen. Explora: A Multipattern and Multistrategy Discovery Assistant. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 249–271. American Association for Artificial Intelligence, Menlo Park, CA, USA, 1996.

[98] A. Knobbe, B. Crémilleux, J. Fürnkranz, and M. Scholz. From Local Patterns to Global Models: The LeGo Approach to Data Mining. In *From Local Patterns to Global Models (LEGO-08), ECML/PKDD Workshop*, 2008.

[99] R. Kohavi. A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 1137–1145, 1995.

[100] Y. Kono, M. Ikeda, and R. Mizoguchi. THEMIS: A Nonmonotonic Inductive Student Modeling System. *International Journal of Artificial Intelligence in Education*, 5:371–413, March 1994.

[101] I. Kononenko. Machine Learning for Medical Diagnosis: History, State of the Art and Perspective. *Artificial Intelligence in Medicine*, 23(1):89–109, 2001.

[102] S. Kotsiantis and D. Kanellopoulos. Association Rules Mining: A Recent Overview. *GESTS International Transactions on Computer Science and Engineering*, 32(1):71–82, 2006.

[103] J. Langford, R. Oliveira, and B. Zadrozny. Predicting Conditional Quantiles via Reduction to Classification. In *Proceedings of the 22nd Annual Conference on Uncertainty in Artificial Intelligence (UAI-06)*, pages 257–264, Arlington, Virginia, 2006. AUAI Press.

[104] N. Lavrač, P. A. Flach, and B. Zupan. Rule Evaluation Measures: A Unifying View. In S. Džeroski and P. Flach, editors, *Proceedings of the 9th International Workshop on Inductive Logic Programming (ILP-99)*, pages 174–185. Springer-Verlag, 1999.

[105] N. Lavrač, B. Kavšek, P. Flach, and L. Todorovski. Subgroup Discovery with CN2-SD. *Journal of Machine Learning Research*, 5:153–188, 2004.

[106] N. Lavrač, B. Cestnik, and S. Džeroski. Search Heuristics in Empirical Inductive Logic Programming. In *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI92-ILP)*, Vienna, Austria, 1992.

[107] N. Lavrač, B. Cestnik, and S. Džeroski. Use of Heuristics in Empirical Inductive Logic Programming. In S. H. Muggleton and K. Furukawa, editors, *Proceedings of the 2nd International Workshop on Inductive Logic Programming (ILP-92)*, number TM-1182 in ICOT Technical Memorandum, ILP-92-address, 1992. Institute for New Generation Computer Technology.

[108] L. Li and H.-T. Lin. Ordinal Regression by Extended Binary Classification. In B. Schölkopf, J. C. Platt, and T. Hofmann, editors, *Advances in Neural Information Processing Systems 19*, pages 865–872. MIT Press, 2007.

[109] H. Lin. *From Ordinal Ranking to Binary Classification*. CIT theses. California Institute of Technology, 2008.

[110] B. Liu, W. Hsu, and Y. Ma. Integrating Classification and Association Rule Mining. In R. Agrawal, P. Stolorz, and G. Piatetsky-Shapiro, editors, *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining (KDD-98)*, pages 80–86, 1998.

[111] H. Liu and H. Motoda. *Feature Selection for Knowledge Discovery and Data Mining*. Kluwer Academic Publishers, Norwell, MA, USA, 1998.

[112] P. S. Maclin, J. Dempsey, J. Brooks, and J. Rand. Using Neural Networks to Diagnose Cancer. *Journal of Medical Systems*, 15:11–19, March 1991.

[113] R. S. Michalski. On the Quasi-Minimal Solution of the Covering Problem. In *Proceedings of the 5th International Symposium on Information Processing (FCIP-69)*, volume A3 (Switching Circuits), pages 125–128, Bled, Yugoslavia, 1969.

[114] I. Mierswa, M. Wurst, R. Klinkenberg, M. Scholz, and T. Euler. YALE: Rapid Prototyping for Complex Data Mining Tasks. In L. Ungar, M. Craven, D. Gunopulos, and T. Eliassi-Rad, editors, *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-06)*, pages 935–940, New York, NY, USA, August 2006. ACM.

[115] J. Mingers. An Empirical Comparison of Selection Measures for Decision-Tree Induction. *Machine Learning*, 3(4):319–342, 1989.

[116] T. M. Mitchell. Generalization as Search. *Artificial Intelligence*, 18(2):203–226, 1982.

[117] T. M. Mitchell. *Machine Learning*. McGraw-Hill Series in Computer Science. WCB/McGraw-Hill, Boston, MA, 1997.

[118] D. Mladenić. Combinatorial Optimization in Inductive Concept Learning. In *Proceedings of the 10th International Conference on Machine Learning (ICML-93)*, pages 205–211, 1993.

[119] M. Mozina, J. Demšar, J. Zabkar, and I. Bratko. Why Is Rule Learning Optimistic and How to Correct It. In *Proceedings of the 17th European Conference on Machine Learning (ECML-06)*, pages 330–340, 2006.

[120] S. Muggleton. Inductive Logic Programming. *New Generation Computing*, 8:295–318, 1991.

[121] S. Muggleton and L. de Raedt. Inductive Logic Programming: Theory and Methods. *Journal of Logic Programming*, 19:629–679, 1994.

[122] S. H. Muggleton. Inverse Entailment and Progol. *New Generation Computing*, 13(3,4):245–286, 1995. Special Issue on Inductive Logic Programming.

[123] S. K. Murthy and S. Salzberg. Lookahead and Pathology in Decision Tree Induction. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 1025–1031, Montreal, Canada, 1995. Morgan Kaufmann.

[124] A. Naseri. Vergleich zwischen geordneten und ungeordneten Regelmengen. Master's thesis, TU Darmstadt, 2011. In German (English title: *A comparison of ordered and unordered rule sets*).

[125] T. Niblett. Constructing Decision Trees in Noisy Domains. In *Proceedings of the 2nd European Working Session on Learning (EWSL-87)*, pages 67–78, 1987.

[126] G. Pagallo and D. Haussler. Boolean Feature Discovery in Empirical Learning. *Machine Learning*, 5(1):71–99, May 1990.

[127] S.-H. Park and J. Fürnkranz. Efficient Pairwise Classification. In *Proceedings of 18th European Conference on Machine Learning (ECML-07)*, pages 658–665, Warsaw, Poland, 2007. Springer-Verlag.

[128] S.-H. Park and J. Fürnkranz. Efficient Decoding of Ternary Error-Correcting Output Codes for Multiclass Classification. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD-09)*, volume Part II, pages 189–204, Bled, Slovenia, 2009. Springer-Verlag.

[129] G. Piatetsky-Shapiro. Discovery, Analysis, and Presentation of Strong Rules. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 229–248. MIT Press, 1991.

[130] J. Quinlan. Learning First-Order Definitions of Functions. *Journal of Artificial Intelligence Research*, 5:139–161, 1996.

[131] J. R. Quinlan. Learning Efficient Classification Procedures and their Application to Chess End Games. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning. An Artificial Intelligence Approach*, pages 463–482. Tioga, Palo Alto, CA, 1983.

[132] J. R. Quinlan. Learning Logical Definitions from Relations. *Machine Learning*, 5(3):239–266, 1990.

[133] J. R. Quinlan. MDL and Categorial Theories (Continued). In A. Prieditis and S. J. Russell, editors, *Proceedings of the 12th International Conference on Machine Learning (ICML-95)*, pages 464–470. Morgan Kaufmann, 1995.

[134] J. R. Quinlan and R. M. Cameron-Jones. Oversearching and Layered Search in Empirical Learning. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 1019–1024, 1995.

[135] R. J. Quinlan. Learning with Continuous Classes. In *Proceedings of the 5th Australian Joint Conference on Artificial Intelligence (AI-92)*, pages 343–348, Singapore, 1992. World Scientific.

[136] L. D. Raedt, P. Frasconi, K. Kersting, and S. Muggleton, editors. *Probabilistic Inductive Logic Programming - Theory and Applications*, volume 4911 of *Lecture Notes in Computer Science*. Springer, 2008.

[137] P. Refaeilzadeh, L. Tang, and H. Liu. Cross-validation. In *Encyclopedia of Database Systems*, pages 532–538. Springer, 2009.

[138] U. Reinhold and E. Breitbart. *Hautkrebsprävention: Früherkennung und Vorbeugung*. Schlütersche Verlag, Hans-Böckler-Allee 7, 30173 Hannover, 2007.

[139] P. R. Rijnbeek and J. A. Kors. Finding a short and accurate decision rule in disjunctive normal form by exhaustive search. *Machine Learning*, 80(1):33–62, 2010.

[140] L. J. Rips. *The Psychology of Proof: Deductive Reasoning in Human Thinking*. MIT Press, Cambridge, MA, USA, 1994.

[141] J. Rissanen. Modeling By Shortest Data Description. *Automatica*, 14:465–471, 1978.

[142] R. Rivest. Learning Decision Lists. *Machine learning*, 2(3):229–246, 1987.

[143] L. Rokach and O. Maimon. Top-down Induction of Decision Trees Classifiers - A Survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 35(4):476–487, 2005.

[144] U. Rückert and S. Kramer. A Statistical Approach to Rule Learning. In *Proceedings of the 23rd International Conference on Machine Learning (ICML-06)*, pages 785–792, New York, NY, USA, 2006. ACM.

[145] S. R. Safavian and D. Landgrebe. A Survey of Decision Tree Classifier Methodology. *IEEE Transactions on Systems, Man and Cybernetics*, 21(3):660–674, 1991.

[146] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA, 1986.

[147] S. Salzberg. On Comparing Classifiers: Pitfalls to Avoid and a Recommended Approach. *Data Mining and Knowledge Discovery*, 1:317–327, 1997.

[148] R. E. Schapire and Y. Singer. Improved Boosting Algorithms Using Confidence-rated Predictions. *Machine Learning*, 37(3):297–336, December 1999.

[149] T. Scheffer. Finding Association Rules that Trade Support Optimally against Confidence. *Intelligent Data Analysis*, 9(3):381–395, 2005.

[150] C. D. Scott and M. A. Davenport. Regression Level Set Estimation via Cost-Sensitive Classification. *IEEE Transactions on Signal Processing*, 55(6):2752–2757, June 2007.

[151] R. B. Segal. *Machine Learning as Massive Search*. PhD thesis, University of Washington, 1997.

[152] R. J. Simes. Treatment Selection for Cancer Patients: Application of Statistical Decision Theory to the Treatment of Advanced Ovarian Cancer. *Journal of Chronic Diseases*, 38(2):171–186, 1985.

[153] R. Sison and M. Shimura. Student Modeling and Machine Learning. *International Journal of Artificial Intelligence in Education*, 9(1-2):128–158, 1998.

[154] A. J. Smola and B. Schölkopf. A Tutorial on Support Vector Regression. *Statistics and Computing*, 14:199–222, August 2004.

[155] S. Steger. Implementation und Evaluation eines Regressionsregellerners. Diploma thesis, TU Darmstadt, 2011. In German (English title: *Implementation and Evaluation of a Regression Rule Learner*).

[156] W. Stolz, O. Braun-Falco, P. Bilek, W. H. C. Burgdorf, and M. Landthaler. *Farbatlas der Dermatoskopie*. Georg Thieme Verlag, Rüdigerstraße 14, 70469 Stuttgart, 2001.

[157] J.-N. Sulzmann and J. Fürnkranz. An Empirical Comparison of Probability Estimation Techniques for Probabilistic Rules. In J. Gama, V. Santos Costa, A. Jorge, and P. B. Brazdil, editors, *Proceedings of the 12th International Conference on Discovery Science (DS-09)*, pages 317–331. Springer-Verlag, 2009. Winner of Best Student Paper Award.

[158] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning. MIT Press, 1998.

[159] P.-N. Tan, V. Kumar, and J. Srivastava. Selecting the Right Interestingness Measure for Association Patterns. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-02)*, pages 32–41, Edmonton, Alberta, 2002.

[160] P.-N. Tan, V. Kumar, and J. Srivastava. Selecting the Right Objective Measure for Association Analysis. *Information Systems*, 29:293–313, June 2004.

[161] M. Thiel. Separate and Conquer Framework und disjunktive Regeln. Diploma thesis, TU Darmstadt, 2005. In German (English title: *Separate and Conquer Framework and Disjunctive Rules*).

[162] L. Todorovski, P. A. Flach, and N. Lavrac. Predictive Performance of Weighted Relative Accuracy. In D. A. Zighed, J. Komorowski, and J. Zytkow, editors, *Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD-00)*, pages 255–264. Springer-Verlag, September 2000.

[163] L. Torgo. Data Fitting with Rule-Based Regression. In *Proceedings of the 2nd International Workshop on Artificial Intelligence Techniques (AIT-95)*, 1995.

[164] L. Torgo and J. Gama. Regression by Classification. In *Proceedings of 13th Brazilian Symposium on Artificial Intelligence (SBIA-96)*, pages 51–60. Springer-Verlag, 1996.

[165] L. Torgo and J. Gama. Search-based Class Discretization. In *Proceedings of the 9th European Conference on Machine Learning (ECML-97)*, pages 266–273. Springer Verlag, 1997.

[166] L. G. Valiant. A Theory of the Learnable. *Communications of the ACM*, 27:1134–1142, November 1984.

[167] V. Vapnik, E. Levin, and Y. L. Cun. Measuring the VC-Dimension of a Learning Machine. *Neural Computation*, 6(5):851–876, 1994.

[168] B. Ženko, S. Džeroski, and J. Struyf. Learning Predictive Clustering Rules. In *Proceedings of the 4th International Workshop on Knowledge Discovery in Inductive Databases: Revised Selected and Invited Papers, volume 3933 of LNCS*, pages 234–250. Springer, 2005.

[169] Y. Wang and I. H. Witten. Induction of Model Trees for Predicting Continuous Classes. In *Poster papers of the 9th European Conference on Machine Learning (ECML-97)*. Springer, 1997.

[170] G. I. Webb. OPUS: An Efficient Admissible Algorithm for Unordered Search. *Journal of Artificial Intelligence Research*, 5:431–465, 1995.

[171] T. Weise. *Global Optimization Algorithms - Theory and Application*. Self-Published (available under `http://www.it-weise.de/projects/book.pdf`, visited on 2012-05-03), 2nd edition, 2009.

[172] S. M. Weiss and N. Indurkhya. Reduced Complexity Rule Induction. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI'91)*, pages 678–684, San Francisco, CA, USA, 1991. Morgan Kaufmann Publishers Inc.

[173] S. M. Weiss and N. Indurkhya. Rule-Based Regression. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 1072–1078, 1993.

[174] S. M. Weiss and N. Indurkhya. Rule-based Machine Learning Methods for Functional Prediction. *Journal of Artificial Intelligence Research*, 3:383–403, 1995.

[175] S. M. Weiss and N. Indurkhya. Lightweight Rule Induction. In *Proceedings of the 17th International Conference on Machine Learning (ICML-00)*, pages 1135–1142, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[176] M. Wiese. A Bidirectional ILP Algorithm. In *Proceedings of the MLnet Familiarization Workshop on Data Mining with Inductive Logic Programming (ILP for KDD)*, pages 61–72, 1996.

[177] I. H. Witten and E. Frank. *Data Mining — Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann Publishers, 2nd edition, 2005.

[178] L. Wohlrab and J. Fürnkranz. A Review and Comparison of Strategies for Handling Missing Values in Separate-and-Conquer Rule Learning. *Journal of Intelligent Information Systems*, 36:73–98, February 2011.

[179] D. H. Wolpert and W. G. Macready. No Free Lunch Theorems for Optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

[180] S. Wrobel. An Algorithm for Multi-relational discovery of Subgroups. In J. Komorowski and J. Zytkow, editors, *Proceedings of the 1st European Symposion on Principles of Data Mining and Knowledge Discovery (PKDD-97)*, pages 78–87, Berlin, 1997. Springer Verlag.

[181] T. Wu, Y. Chen, and J. Han. Association Mining in Large Databases: A Re-examination of Its Measures. In *Proceedings of the 11th European Symposium on Principles of Data Mining and Knowledge Discovery (PKDD-07)*, pages 621–628, Warsawa, Poland, 2007. Springer-Verlag.

[182] H. Xiong, S. Shekhar, P.-N. Tan, and V. Kumar. Exploiting a Support-based Upper Bound of Pearson's Correlation Coefficient for Efficiently Identifying Strongly Correlated Pairs. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-04)*, pages 334–343, Seattle, USA, 2004.

[183] M. Zorman, M. M. Štiglic, P. Kokol, and I. Malčić. The Limitations of Decision Trees and Automatic Learning in Real World Medical Decision Making. *Journal of Medical Systems*, 21:403–415, December 1997.

Own Publications

Journal Papers

Frederik Janssen and Johannes Fürnkranz. On the quest for optimal rule learning heuristics. *Machine Learning*, 78(3):343–379, March 2010. DOI 10.1007/s10994-009-5162-2.

Conference Papers

Frederik Janssen and Johannes Fürnkranz. On Meta-Learning Rule Learning Heuristics. In *Proceedings of the 7th IEEE Conference on Data Mining (ICDM-07)*, pages 529–534, Omaha, NE, 2007.

Frederik Janssen and Johannes Fürnkranz. An Empirical Investigation of the Trade-Off Between Consistency and Coverage in Rule Learning Heuristics. In T. Horvath, J.-F. Boulicaut, and M. Berthold, editors, *Proceedings of the 11th International Conference on Discovery Science (DS-08)*, pages 40–51, Budapest, Hungary, 2008. Springer Verlag.

Frederik Janssen and Johannes Fürnkranz. A Re-evaluation of the Over-Searching Phenomenon in Inductive Rule Learning. In *Proceedings of the SIAM International Conference on Data Mining (SDM-09)*, pages 329–340, Sparks, Nevada, 2009.

Frederik Janssen and Johannes Fürnkranz. Heuristic Rule-Based Regression via Dynamic Reduction to Classification. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI-11)*, pages 1330–1335, Barcelona, Spain, 2011.

Proceedings

Frederik Janssen and Melanie Hartmann, editors. *Proceedings of the LWA 2009: Lernen - Wissen - Adaptivität*, number TUD-KE-2009-04 (Knowledge Engineering Group Technical Report), TUD-CS-2009-0157 (Telekooperation Group Technical Report), Darmstadt, Germany, 2009.

Workshop Papers

Frederik Janssen and Johannes Fürnkranz. Separate-and-conquer Regression. In *Proceedings of the German Workshop on Lernen, Wissen, Adaptivität - LWA2010*, pages 81–89, 2010.

Frederik Janssen and Johannes Fürnkranz. A Re-evaluation of the Over-Searching Phenomenon in Inductive Rule Learning. In Martin Atzmüller and Joachim Baumeister, editors, *Proceedings of the German Workshop on Lernen, Wissen, Adaptivität - LWA2008*, pages 42–50, Würzburg, Germany, 2008.

Frederik Janssen and Johannes Fürnkranz. Meta-Learning Rule Learning Heuristics. In Pavel B. Brazdil and A. Bernstein, editors, *Proceedings of ECML-PKDD-07 Workshop on Planning to Learn (PlanLearn-07)*, pages 9–21, Warsaw, Poland, 2007.

Frederik Janssen and Johannes Fürnkranz. Meta-Learning Rule Learning Heuristics. In *Proceedings of the German Workshop on Lernen, Wissen, Adaptivität - LWA2007*, pages 167–174, Halle, Germany, 2007.

Frederik Janssen and Johannes Fürnkranz. On Trading Off Consistency and Coverage in Inductive Rule Learning. In Klaus-Dieter Althoff and M. Schaaf, editors, *Proceedings of the German Workshop on Lernen, Wissen, Adaptivität - LWA2006*, pages 306–313, Hildesheim, Germany, 2006. Gesellschaft für Informatik e. V. (GI).

Frederik Janssen, Faraz Fallahi, Jan Noessner, and Heiko Paulheim. Towards Rule Learning Approaches to Instance-based Ontology Matching. In *1st International Workshop on Knowledge Discovery and Data Mining Meets Linked Open Data (Know@LOD)*, pages 13–18, Heraklion, Greece, 2012.

Diploma Thesis

Frederik Janssen. Eine Untersuchung des Trade-Offs zwischen Precision and Coverage bei Regel-Lern-Heuristiken. Diploma thesis, TU Darmstadt, 2006. In German (English title: *On Trading Off Consistency and Coverage in Inductive Rule Learning*).

Wissenschaftlicher Werdegang[1]

06/1998	Allgemeine Hochschulreife
09/1999 — 06/2006	Studium der Informatik an der Technischen Universität Darmstadt
09/2006	Diplom in Informatik
seit 09/2006	Doktorand am Fachbereich Informatik, Technische Universität Darmstadt
02/2002 — 04/2004	Wissenschaftliche Hilfskraft, GMD Darmstadt
seit 09/2006	Wissenschaftlicher Mitarbeiter, Fachbereich Informatik, Technische Universität Darmstadt

Erklärung[2]

Hiermit erkläre ich, dass ich die vorliegende Arbeit, mit Ausnahme der ausdrücklich genannten Hilfsmittel, selbständig verfasst habe.

[1] gemäß § 20 Abs. 3 der Promotionsordnung der TU Darmstadt
[2] gemäß § 9 Abs. 1 der Promotionsordnung der TU Darmstadt