Rule Learning: From Local Patterns to Global Models

TECHNISCHE UNIVERSITÄT DARMSTADT

Am Fachbereich Informatik der
Technischen Universität Darmstadt
zur Erlangung des akademischen Grades eines
Doktors der Naturwissenschaften (Dr. rer. nat.)
eingereichte

Dissertation

von

Dipl.-Inform. Jan-Nikolas Sulzmann
(geboren in Langen)

Referent:	Prof. Dr. Johannes Fürnkranz
Koreferent:	Prof. Dr. Stefan Kramer (Johannes Gutenberg Universität Mainz)
Tag der Einreichung:	09.03.2018
Tag der mündlichen Prüfung:	20.04.2018

D17
Darmstadt 2018

Sulzmann, Jan-Nikolas
Rule Learning: From Local Patterns to Global Models
Darmstadt, Technische Universität Darmstadt,
Jahr der Veröffentlichung der Dissertation auf TUprints: 2019
Tag der mündlichen Prüfung: 20.04.2018

Kurzfassung

Heutzutage werden große Datenmengen in vielen Bereichen des täglichen Lebens (z.B. im E-Commerce oder in sozialen Netzwerken) gesammelt und in Datenbanken (für eine zukünftige Verwendung) abgelegt. Obwohl die spezifischen Informationen in den gesammelten Daten bereits von Interesse sein können, sind allgemeinere Erkenntnisse über die Dateninhalte weitaus nützlicher. Eine Datenanalyse sollte aus diesem Grund darauf abzielen, derartiges (Teil-)Wissen zu erlangen. Eine Inspektion der Daten durch Menschen wird jedoch immer weniger praktikabel, da die vorliegenden Datenmengen immer größer und unhandlicher werden. Dieses Problem wird durch den KDD-Prozess (kurz für “Knowledge Discovery in Databases”) gelöst, der die notwendigen Werkzeuge für eine halbautomatische Datenanalyse zur Verfügung stellt. Dessen Hauptkomponente Data-Mining durchsucht die expliziten Fakten nach Regelmäßigkeiten. Üblicherweise werden derartige Regelmäßigkeiten als lokale Muster, die lokale Charakteristika der Daten beschreiben, oder als globale Modelle, die die Daten in ihrer Gesamtheit erklären, formuliert. In unserer Arbeit werden wir uns mit lokalen Mustern und globalen Modellen beschäftigen, die zukünftige und unbekannte Daten bezüglich eines Merkmals oder Klassenattributs klassifizieren. Interessanterweise können vorhersagende lokale Muster auf zwei Weisen eingesetzt werden, um eine einzige globale Vorhersage zu erhalten. Der integrative Ansatz behandelt die lokalen Muster als Bausteine und generiert aus ihnen ein globales Modell. Der dekodierende Ansatz verwendet die Vorhersagen mehrerer lokaler Muster und dekodiert diese in eine gemeinsame Vorhersage. Obwohl beide Ansätze vielversprechend sind, wurde bisher die Frage, wie lokale Muster zur Modellierung globaler Modelle eingesetzt werden können, noch nicht zufriedenstellend beantwortet. Daher betrachten wir drei wichtige Teilaspekte dieser Frage, die jeweils in einem Teil unserer Arbeit behandelt werden.

Der erste Teil unserer Arbeit befasst sich mit der Frage, wie eine Menge lokaler Muster dazu eingesetzt werden kann, optimale globale Vorhersagen zu erhalten. Das LeGo-Framework (ein Akronym für “from local patterns to global models”) bietet einen Ansatz, um diese Frage zu behandeln. Es unterteilt den Data-Mining-Prozess in drei aufeinanderfolgende Teilschritte: “Local Pattern Discovery” generiert eine Menge lokaler Muster, “Pattern Set Discovery” selektiert von dieser eine kleinere Teilmenge und “Global Modelling” verwendet diese Teilmenge zur Erstellung eines globalen Modells. Für jeden dieser drei

Teilschritte stehen diverse anwendbare Methoden zur Verfügung. Aus diesem Grund selektieren wir für jeden Teilschritt eine Auswahl von Methoden und evaluieren diese mittels eines empirischen Vergleichs.

Der zweite Teil unserer Arbeit behandelt die Frage, wie eine Menge lokaler Muster dazu verwendet werden kann, optimale Klassenwahrscheinlichkeiten vorherzusagen. Häufig sind Klassenwahrscheinlichkeiten nützlicher als eine einfache Vorhersage, da man sie als Konfidenzmaß für die Vorhersage verwenden kann (z.B. bei Abstimmungsschemata). Wir teilen diese Fragestellung in zwei Teilaufgaben auf: die Wahrscheinlichkeitsabschätzung und die Wahrscheinlichkeitsaggregation. Die Wahrscheinlichkeitsabschätzung berechnet für ein gegebenes lokales Muster die Klassenwahrscheinlichkeiten. Für diese Aufgabe betrachten wir einfache Methoden zur Wahrscheinlichkeitsabschätzung und die Technik Shrinkage, die die einfachen Wahrscheinlichkeitsabschätzungen glättet. Des Weiteren untersuchen wir auch den Einfluss der Suche nach lokalen Mustern auf die Performanz der Wahrscheinlichkeitsabschätzung. Die Wahrscheinlichkeitsaggregation dekodiert die Wahrscheinlichkeitsabschätzungen mehrerer lokaler Muster in eine einzige Wahrscheinlichkeitsabschätzung. Hierfür betrachten wir die Performanz ausgewählter Aggregationsmethoden.

Der dritte Teil untersucht die Frage, wie eine Menge lokaler Muster in ein kompaktes und verständliches Modell transformiert werden kann. Üblicherweise sind Mengen lokaler Muster schwierig zu interpretieren und ihre Verwendung zur Vorhersage erfordert zusätzliche Maßnahmen (z.B. Abstimmungsschemata). Diese beiden Probleme werden gleichzeitig gelöst, wenn man die lokalen Muster zur Erstellung eines globalen Modells verwendet. Zu diesem Zweck stellen wir den neuen Ansatz Rule Stacking zur Erstellung globaler Modelle vor. Rule Stacking entwickelt den allgemeinen Stackingansatz in zwei Aspekten weiter: eine alternative Generierung der Metadaten und eine zusätzliche Rücktransformation des Metamodells. Auf diese Weise erhält man ein komprimiertes und interpretierbares globales Modell, das direkt auf zukünftige Daten angewandt werden kann.

Abstract

In many areas of daily life (e.g. in e-commerce or social networks), massive amounts of data are collected and stored in databases (for future use). Even though the specific information contained in the collected data may already be interesting, more general insights into the data would be more useful. Clearly, a data analysis should aim for a discovery of such pieces of knowledge, but a human inspection becomes less and less feasible to do as the databases become more and more unmanageable. To this end, the KDD process (short for "Knowledge Discovery in Databases") provides the tools for a semi-automatic data analysis. Data mining, which is the main component of the KDD process, searches the explicit facts for regularities which represent pieces of knowledge. Usually, these regularities are formulated as local patterns which describe only local characteristics of the data or as global models which explain the whole data. In our work, we will concentrate on local patterns and global models that may be used to predict a feature of interest or class attribute for future and unknown data. Interestingly, predictive local patterns may be used to obtain global predictions in two ways. The integrative approach treats the local patterns as building blocks and builds with their help a global model. The decoding approach aggregates the predictions of the local patterns into a single global prediction. While both approaches are promising, the question, how local patterns may be employed for global modelling, has not been answered satisfactorily yet. To this end, we consider three important aspects of this question in this work.

The first aspect is, how may a set of local patterns be employed to obtain optimal global predictions. The LeGo framework (an acronym for "from **l**ocal patt**e**rns to **g**lobal m**o**dels") provides an approach to answer this question. It divides the data mining process into three subsequent steps: the local pattern discovery generates a set of local patterns, the pattern set discovery step selects a smaller subset from the set of local patterns, and the global modelling employs the reduced pattern set to build a global model. There are many methods available for each step. So, we employ a selection of methods for each step and evaluate their performances with respect to the first considered aspect empirically.

The second aspect is, how may a set of local patterns be utilised to obtain optimal class probabilities. Often class probabilities may be more useful than a simple prediction as they may be used as a confidence measure in the prediction (e.g. in voting schemes). We divide this aspect into two sub tasks:

the probability estimation and the probability aggregation. The probability estimation calculates class probabilities given a single local pattern. For this task, we consider basic probability estimation methods and shrinkage which is a technique to smooth the basic probability estimations. Furthermore, we examine the effect of the local pattern discovery on the quality of the probability estimation. The probability aggregation decodes the probability estimations of multiple patterns into a single probability estimation. For this purpose, we evaluate the performances of a selection of aggregation methods.

The third aspect is, how may a set of local patterns be transformed into a compact and understandable model. Usually, local pattern sets are hard to interpret and their utilisation for prediction necessitates additional efforts (e.g. voting schemes). These issues may be solved at once if the local patterns are employed to obtain a global model. To this end, we introduce rule stacking, which is a novel approach for global modelling. Rule stacking advances the standard stacking approach in two aspects: the meta data generation and the additional retransformation of the meta model. In this way, we obtain a compressed and interpretable global model that is directly applicable to future data.

Contents

List of Figures

List of Algorithms

List of Tables

1 Introduction

1.1 Motivation

In almost every area of social and commercial life, an abundance of information is generated for highly diverse purposes. Commonly, this information is collected in ever-growing databases for later use. The omnipresent internet and its various services are well-known examples for the collection of information. Commercial web sites (e.g. electronic commerce, gambling, or gaming sites) gather eagerly information about their customers such as their personal data (e.g. name, age, location), their shopping behaviour (e.g. preferred product categories or common shopping baskets) or web surfing behaviour (e.g. the frequency and succession of visited sites). Members of social networks (e.g. Facebook, Twitter, or LinkedIn) mutually share personal information about their private lives such as their personal activities, interests, and preferences, and/or about their professional life such as their curricula vitae and business interests.

Even though all this explicit, very specific information may already be very interesting, more general and hence more useful information could be derived from it. Commercial sites could customise their offerings, promotions, or advertisements to be tailor-made for each customer. Social networks could offer or upgrade services that alleviate the search for members that share common interests. Both examples have in common that the specific information has to be transformed into more generalised information or pieces of knowledge. For example, the very specific information "‘John Smith who likes skiing often visits Austria in winter"’ is less informative than the more general information "‘many persons which are interested in winter sports spent their winter holidays in alpine regions"’, since it provides a more abstract insight into the domain or, in other words, it represents a piece of knowledge about it.

Obviously, the main goal of the analysis of a data collection should be the discovery of knowledge. In some areas, a manual data analysis by human domain experts may be feasible, especially in small databases experts may interpret the data and recognise pieces of knowledge. However, databases grow steadily in size due to the availability of cheap storage space. Consequently, the interpretation of data by human experts becomes less and less feasible, and automatic methods for knowledge discovery are urgently needed. The interdisciplinary field of "‘Knowledge Discovery in Databases"’ (abbreviated KDD) addresses this issue by providing a semi-automatic process (referred to as the KDD process) that assists humans in the discovery of knowledge [PSF91, FPS96b]. Data mining is the main component of the KDD process as it provides the tools to search in the explicit facts of the data for regularities that each represent a single piece of implicit knowledge. The regularities found in this way are formulated explicitly as patterns. Depending on the intended use of these patterns, data mining may be divided into two main categories. On the one hand, descriptive data mining offers a wide range of algorithms for the discovery of local patterns (e.g. association rules or subgroups). Each of these patterns describes local characteristics of the data. Most of the information content of the data set may be captured by the total of local patterns, however, a complete picture or explaining global model is not obtained in this way. Obviously, the totality of the local patterns may be hard to interpret. On the other hand, predictive data mining provides algorithms for the generation of global models (e.g. decision trees, or decision lists) that aim to make precise predictions on future, unseen data. Such models may be seen as global patterns that explain the data as a whole. Usually, global models predict the values of a feature of interest which is commonly referred to as the class attribute of the data set. Classification, as the data mining task of predicting the class attribute is called, will be the focus of this work.

Even though descriptive data mining has been developed for a different purpose, the global modelling of predictive data mining may clearly profit from the local pattern discovery. To this end, two approaches to employ local patterns for global modelling may be considered:

Integrative Approach

Since global models usually may be composed of local patterns (for instance, decision lists consist of individual rules), an obvious way to combine these two worlds is to integrate the local pattern discovery into the global modelling process. This integrative approach is exemplified by the CBA algorithm (an acronym for **c**lassification **b**y **a**ssociation) [LHM98]. To this end, CBA employs conventional local pattern discovery algorithms to generate association rules (for example, the Apriori algorithm) for classification. Afterwards, a simple covering algorithm turns the generated association rules into a useful global model for prediction.

Decoding Approach

Alternatively to this integrative approach, each local pattern may be regarded as a model for the prediction of local properties. Analogous to ensembles of global models, the (local) predictions may be decoded into a single global prediction, obtaining an implicit global model in this way. Commonly, voting methods are employed to decode several predictions into a single one, predicting the class value that received the most votes. Since each local pattern covers only a (small) part of the data space, only the votes of the covering local patterns have to be considered for this purpose.

Both the integrative and the decoding approach have in common that they employ a set of local patterns to obtain predictions for unseen data. Even though this simple idea yields promising results, the general question, how local patterns may be employed for global modelling, has not been answered satisfactorily yet. We elaborate this question in this thesis, investigating three important aspects of it:

- How may a set of local patterns be employed to obtain optimal predictions?
- How may a set of local patterns be utilised to obtain optimal class probabilities?
- How may a set of local patterns be transformed into a compact and understandable model?

In the following chapters, we will study these questions, searching for satisfactory answers. In our studies, we utilise (propositional) rules as local patterns since rules are in our opinion the most natural way to formulate patterns. Thus, we will not distinguish between local patterns and rules, using both terms interchangeably in this work.

1.2 Contributions

As mentioned before, in this work we investigate, how a set of local patterns may be used to obtain accurate predictions on unseen data. To this end, we study three questions (see above), each of which is related to a specific aspect of the problem. According to these questions, the contribution of our work is divided into three parts:

Theory Formation

In the first part of our work, we investigate the question, how a set of local patterns may be employed to obtain optimal predictions [SF08]. The LeGo framework [KCFS08] provides a generic solution to this problem which is divided into three sub steps for this purpose. Between the previously described local pattern discovery and global modelling steps, an intermediate step, the pattern set discovery, is introduced that reduces the local pattern set to a smaller one. Ideally, the reduced pattern set consists only or at least mostly of those patterns that are needed in the subsequent global modelling step to obtain an optimal model. For each of the three steps several methods exist that may be employed for the associated task, allowing a myriad of configurations of the LeGo framework. Since the LeGo framework and the options for each of it steps have not been evaluated systematically so far, we are going to approach this task with an extensive empirical study. To this end, we consider a selection of different options for each step and identify the best amongst the selected options respectively.

Probability Estimation

In the second part of our work, we tackle the problem, how a set of local patterns may be utilised to obtain optimal (class) probabilities [SF09]. In other words, we want to estimate the probability that a data instance belongs to a class, using the local pattern set. Class probabilities may be useful in situations, when a mere prediction is not sufficient and a measure of confidence in the prediction (in form of a probability distribution) may be more useful (e.g. for optimising the weights for voting methods). The problem to obtain such class probabilities may be divided into two sub tasks. The first task is to determine the class probabilities of a given data instance from a single applicable local pattern. For this purpose, we employ basic probability estimation methods which estimate probabilities based on statistical properties of the pattern, and a technique to smooth the probabilities obtained in this way. The second task is to decode the class probability distributions of all applicable local patterns into a single one. To this end, we utilise well-known decoding methods to combine the basic probability estimations employed in the first task.

Theory Compression

In the third part of our work, we study the question, how a set of local patterns may be transformed into a compact and understandable model [SF11]. As mentioned before, a set of (independent) local patterns may be hard to interpret and additional efforts (e.g. the application of voting methods) are needed to obtain predictions with the help of these patterns. Both issues may be solved at once if we generate a global model on the basis of the local pattern set. For this purpose, we suggest a novel approach for the global modelling of rule-based local patterns. Rule stacking, as we named this approach, modifies the general meta learning method stacking and adapts it to the aforementioned global modelling task. The main differences between rule stacking and the standard stacking approach are the generation of the meta data and the retransformation of the obtained meta model into a compressed global model that is directly applicable to future data (and may be easily interpreted). Our experiments show that the global models generated by rule stacking are of lower complexity than the original local pattern set and are easier to interpret.

1.3 Outline

This work is organised as follows. First, we explain the fundamentals of data mining in Chapter 2. To this end, we give an introduction into the employed data, local patterns, global models, and the LeGo framework. In Chapter 3, we investigate the first question, how a set of local patterns may be employed to obtain optimal predictions. Our solution approach involves the evaluation of different configurations of the three-step generic LeGo framework, identifying the best methods for each step. Next, we concern ourselves in Chapter 4 with the second question, how a set of local patterns may be utilised to obtain optimal (class) probabilities. For this purpose, we investigate the performances of basic probability estimation methods, a single shrinkage method and probability aggregation methods. In Chapter 5, we consider the third question, how a set of local patterns may be transformed into a compact and understandable model. For its solution, we propose a novel meta learning approach called rule stacking. At last, we summarise the results of our work in Chapter 6.

2 Foundations

In our environment, we encounter a multitude of patterns which are either of natural or artificial origin. Usually, natural rocks exhibit textures that are witness to their formation (e.g. igneous rocks like speckled granite or metamorphic rocks like banded slate). Through evolution, Earth's flora, fauna and fungi have developed many different patterns for varied purposes, e.g. as camouflage (for example, the points and stripes of big cats like tigers, cheetahs, and jaguars), or aposematism of colouration (for example, the red and white patterns of fly agarics, or the yellow and black patterns of wasps respectively are an evidence of their inedibility and toxicity). Due to its fascination of natural patterns, humanity has created an abundance of artificial patterns in various fields. Repetitive ornaments of geometrical or floral design are used for decoration in arts, architecture, and fashion. Interestingly, those patterns have not to be exact to be recognisable by the human eye (see Figure 2.1).

Common ground of all these patterns is that they indicate an (almost) invariant structure that is repeated with a certain regularity. To apply this observation to data, a pattern represents a set of feature values that regularly co-occur under certain conditions. Typically, this regularity is based on the occurrence of other feature values that influence the appearance of the relevant feature set with a sufficient certainty. From our point of view, the most natural, intuitive way to model such data patterns is to use deterministic rules of the form

$$X \implies Y, \tag{2.1}$$

where both X and Y are sets of feature values. Hence, rules are the patterns we prefer for data mining, and we will focus on rule-based data mining algorithms [FGL12] in this work and its associated experiments.

In the remaining chapter, we give attention to the foundations of two data mining tasks: the search for local patterns and the generation of global models. At first, we introduce in Section 2.1 the relevant properties of the data that we consider in this work. Then, we illustrate the multi-step process of "Knowledge Discovery in Databases" and its main component, the data mining step in Section 2.2. Afterwards, we describe the properties of rule-based local patterns, the general process of learning single local patterns and two categories of local pattern discovery algorithms, subgroup discovery and association rule mining, that will be relevant in our work, in Section 2.3. For these categories, we explain practical pattern discovery algorithms that find

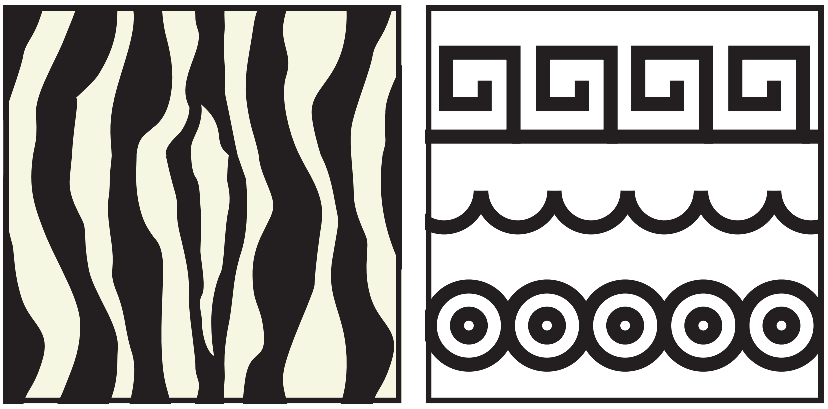

(a) Zebra pattern (b) Ornamental patterns

Figure 2.1.: Patterns in nature (a) and arts (b).

application in our experiments. In the subsequent Section 2.4, we deal with the details of predictive global models. For this purpose, we describe classifiers, the general separate-and-conquer-algorithm for combining local patterns into global patterns or models, the applications of classifier ensembles, and the evaluation of classifiers. We present the rule learning algorithm Ripper which is a well-known and effective implementation of the separate-and-conquer approach. At last, we describe the LeGo framework [KCFS08] which provides a generic solution to the problem, how a set of local patterns may be employed to obtain a global model in Section 2.5.

2.1 Data

Before we are going into the details of data mining, we define data and its relevant properties. Data may be divided into two categories: structured and unstructured data. The first category, structured data, consists of relational data tables, while unstructured data comprises texts and web documents. In this work, we will only consider structured data. A structured data set is defined by a specified set of attributes and consists of a set of instances. Each attribute is determined by its type (e.g. numerical, nominal) and its domain. The domain of an attribute is the set of permissible attribute values. For instance, the domain of "cardinal directions" could be "north", "east", "south", and "west". Each instance of a structured data set is a tuple of attribute values that are drawn from their respective domain. Since the number of attributes of a structured data set is determined, the length of these tuples is fixed, too.

An *attribute* A is described basically by its domain $dom(A)$ which is a set of permissible values

$$dom(A) = \{a_1, \cdots, a_{|A|}\}, \tag{2.2}$$

where

$$|A| = |dom(A)| \tag{2.3}$$

is the number of possible attribute values a_i of attribute A.

Attributes may be divided into multiple *categories* or attribute types which define the set of permissible attribute values of a respective attribute. In our work, we consider two basic attribute types: numerical and nominal attributes. If the set of attribute values is a set of numbers, the attribute is also called *numerical* ($dom(A) \subseteq \mathbb{R}$). *Nominal* attributes permit only a limited list of symbolic attribute values. For example, the legal values of an attribute “primary colours” could be “red”, “blue”, and “yellow”.

A *data set* D is defined by a fixed number of attributes

$$D \subseteq dom(A_1) \times \cdots \times dom\left(A_{|S|}\right) = S, \tag{2.4}$$

where $dom(A_1) \times \cdots \times dom\left(A_{|S|}\right)$ is the data space S and A_i is one of its $|S|$ attributes. $|S|$ is both the dimension of the data space and the number of attributes.

An *instance* d of the data set D is defined by a tuple of attribute values

$$d = \left(a_{d,1}, \cdots, a_{d,|S|}\right) \in D, \tag{2.5}$$

where

$$a_{d,k} \in dom(A_k) \tag{2.6}$$

is one of the legal values of the attribute A_k. For a given data set, $|D|$ is the number of instances in the data set.

In summary, a data set may be described by a matrix of attribute values

$$D = \begin{pmatrix} a_{d_1,1} & \cdots & a_{d_1,|S|} \\ \vdots & \ddots & \vdots \\ a_{d_{|D|},1} & \cdots & a_{d_{|D|},|S|} \end{pmatrix}, \tag{2.7}$$

where the rows and columns represent instances and attributes respectively.

In many real-world databases, the values of some attributes may be known only for parts of the available data, as these attribute values are not measurable or hard to determine. Unfortunately, these attributes are often of special interest, as they may be used to categorise the instances of a data set into separate categories or classes. Consequently, these attribute values have to be determined by other means for the unclassified available and future data. For example, human experts may derive these values from the interpretation of other attributes, that are more easily available, based on their experience. In astronomy, an abundance of (physical) measuring data of astronomical objects is collected automatically, but their category (e.g. star or planet class) still has to be determined by human experts. Obviously, a precise prediction model could be used for the assistance of the human experts in their categorisation efforts (e.g. by providing a preliminary selection of interesting objects) or even for the complete automation of the process.

Both approaches may be handled by classification which is as mentioned before the focus of this work. The goal of classification learning is the prediction of attribute values of a specific attribute whose values are only known for a part of the available data. For this purpose, classification learning algorithms process the observed data to generate models that predict these values for previously unseen instances. According to this, we consider only rule-based patterns that predict values of the class attribute in our work. Consequently, we have to introduce the class attribute and extend the previous definitions of the data set and its instances to classification.

As mentioned above, the main goal of classification learning is the prediction of values of a specific nominal attribute (in contrast to regression where the predicted attribute is numerical) for previously unseen data. In classification learning, this attribute is called the class attribute or label of the data set. According to this we define a classification data set as follows:

$$D \subseteq dom(A_1) \times \cdots \times dom\left(A_{|S|-1}\right) \times dom(L) = S, \tag{2.8}$$

where L is the *class attribute* or *label*. $|L|$ denotes the number of possible class values or labels. If $|L| = 2$, the data set and the associated learning problem is called *binary*. Analogously, if $|L| > 2$, we have to deal with a *multi-class* data set and learning problem.

Additionally, the definition of data instances has to be extended:

$$d = \left(a_{d,1}, \cdots, a_{d,|S|-1}, l_d\right) \in D, \tag{2.9}$$

where $l_d \in dom(L)$ is the class label of d. A data instance whose class label is known is referred to as a *labelled instance* or an *example* (of its class). Accordingly, the class label of *unlabelled instances* is unknown. Likewise, a data set that consists only of labelled or unlabelled instances is called labelled or unlabelled respectively.

Furthermore, a labelled data set may be divided into subsets D_l that each consist of the data instances belonging to a specific class l:

$$D_l = \{d \in D | l_d = l\} \tag{2.10}$$

The size n_l of the data set D_l is defined as

$$n_l = |D_l| \tag{2.11}$$

In real-world data sets, some attribute values of one or more data instances may not reflect their real values, as the generation of data sets is for several reasons prone to errors. Physical values which have to be determined by an appropriate measuring instrument (e.g. length, weight, temperature) may be quantified wrongly as these instruments have only limited precision. Additionally, real-world data sets are often generated using form data (e.g. patients filling in medical blanks in hospitals). The data collected may be faulty, as on the one hand the data is only correct if the form is completed correctly, and on the other hand, the forms have to be digitised without errors. In machine learning (as in physics), this phenomenon is known as *noise*. Data instances and data sets, in which it occurs, are called noisy. As we will see later, successful data mining algorithms have to deal with noise in the data.

2.2 Knowledge Discovery in Databases & Data Mining

In this section, we concern ourselves with the interdisciplinary field of "‘Knowledge Discovery in Databases"’ (abbreviated KDD) and its main component data mining. KDD provides a semi-automatic process (referred to as the KDD process) that assists humans in the discovery of knowledge [PSF91, FPS96a, FPS96b]. The associated KDD process has been defined as follows:

> *Knowledge discovery in databases is the non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable pattern in the data [FPS96a].*

This definition introduces several important terms and concepts that have to be clarified first.

Data often exhibits regularities, such as the frequent co-occurrence of certain feature values. Regularities may be formulated as expressions which are referred to as patterns in some kind of representation language (e.g. rules, or decision trees). Thus, each pattern describes a single characteristic of (a subset of) the data and represents an explicit formulation of knowledge in this way. A collection of patterns may be used to induce a model that describes the complete data (set). The KDD definition makes demands on the extracted patterns: validity, novelty, usefulness, and understandability. To check if patterns meet these demands, quantitative measures, referred to as quality heuristics, are employed to evaluate them. Normally, objective measures for validity or certainty (e.g. the estimation of the prediction accuracy on data), or utility

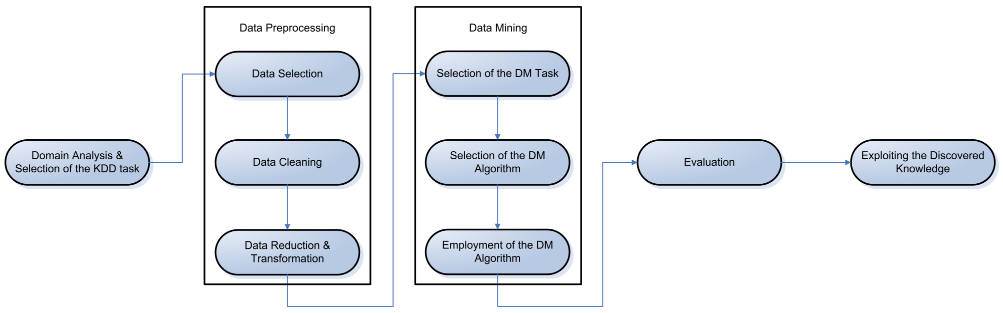

Data Preprocessing
Data Mining
Domain Analysis & Selection of the KDD task
Data Selection
Data Cleaning
Data Reduction & Transformation
Selection of the DM Task
Selection of the DM Algorithm
Employment of the DM Algorithm
Evaluation
Exploiting the Discovered Knowledge

Figure 2.2.: KDD Process: illustration of the life cycle of the process of "Knowledge Discovery in Databases".

(e.g. the achieved gain in cost or time savings) may be easily defined. Measures for novelty and understandability are a bit problematic because they will always be a bit subjective, depending on the user's experience and preferences. Nevertheless, we will concentrate on patterns that exhibit a high certainty and are understandable, as we search for accurate local patterns in the form of conjunctive rules, which may easily be interpreted.

The complete KDD process consists of multiple, consecutive steps, in which one or more automated, parametrised methods are applied to the data (see Figure 2.2). The interaction of these steps is not strictly linear, as unpromising results of one step can necessitate the restart at former steps using other methods or parameters (please note that such restarts have been omitted in the figure for a better overview). The key part of the KDD process is the data mining step. The goal of this step is to discover and extract interesting patterns from the database. Generally, the patterns gathered in this way are used to generate models that may be used either to obtain a better understanding of the current data (e.g. relations between feature values) or to analyse future data (e.g. making predictions on unknown examples). In our work, we will concentrate on the data mining step of the KDD process and especially on the machine learning methods that we use in our experiments. Nevertheless, we will describe the objectives and the employed techniques of each step of the KDD process briefly in the next paragraphs.

2.2.1 Step 1: Domain Analysis & Selection of the Knowledge Discovery Task

The initial step still has to be done manually by human experts. First, these experts have to analyse the domain of the given data set, identify the special properties of its domain and formulate possible goals of the knowledge discovery. Next, they have to select one of the formulated goals as the main goal or task of the KDD process. Obviously, this decision has a major impact on the following steps. On the one hand, it influences the manual preselection of the data which is used for the knowledge discovery task, as only an appropriate subset of the available data is normally needed. On the other hand, it constrains the number of meaningful choices in each of the following steps (e.g. the selection of a prediction task appoints predictive data mining methods in the later data mining step).

2.2.2 Step 2: Data Selection

In the *data selection* step, a target data set has to be prepared, on which the knowledge discovery task will be performed. The tricky part of this step is to select only the data that is relevant for the previously defined KDD goal. Therefore, we have to determine a subset of the available data by selecting only a relevant, representative portion of the available instances and features. Obviously, this step has a major impact on the following steps. On the one hand, if too few features or instances are selected, important information may be missing and the entire process may fail. On the other hand, if too many features or instances are chosen, irrelevant information may be considered and processed. Hence, the entire process may be slowed down or may be even impossible as some data mining methods are prone to a big number of features and/or instances (e.g. nearest neighbour learning [Agg14]). That being the case, both aspects, the relevance and size of the considered data, have to be traded off carefully.

Normally, the data that has been selected for the data analysis is distributed over several relational tables which may be part of one or more data bases. Therefore, we have to extract the previously determined parts of each relational table. Afterwards, the selected parts have to be adjusted (e.g. the normalisation of different denominations, values or formats of associated features [Pyl99]) and integrated in to a single data set. Sometimes, the available data may be not sufficient as the data misses important features or possesses only an inadequate number of instances. In both cases, it may be worthwhile to collect or generate additional data in an appropriate way. In a few cases, target data sets for a given data analysis task may be already available (e.g. created for a former, similar KDD task). Obviously, the direct use of available data sets saves time and effort. Nevertheless, the creation of a new data set may be rewarding in many cases, since different aspects of the domain may be emphasised in this way.

Before the selected target data is used for data mining, the application of one or more *pre-processing* methods could be worthwhile. The most popular goals of pre-processing methods [Pyl99] are the *data cleaning* (e.g. removal of inconsistencies, noise, or outliers) and the *data reduction* (e.g. by feature subset selection or sampling), and the *data transformation* (e.g. discretisation) . The utilisation of these pre-processing methods takes place in the two subsequent steps.

2.2.3 Step 3: Data Cleaning

The pre-processing step *data cleaning* (a.k.a. data cleansing) focusses on the treatment of faulty or misleading data to obtain a clean data set that is consistent with its domain [Pyl99]. These faults comprises inconsistent, missing, and noisy attribute values that may originate from technical problems (e.g. values may have been corrupted by faulty measures, transmission, or storage), human interaction (e.g. users may have (in-)voluntarily entered wrong entries), or different semantics or metrics (e.g. the utilised data sources use different dictionaries, systems or units of measure). Obviously, these faults may influence the data mining process unfavourably and therefore should be treated appropriately. Dependent on these reasons, the detection and treatment of these faults may be either trivial (e.g. adjusting date formats or conversion of units of measure) or somewhat expensive as additional means have to be employed (e.g. filling missing values with a default or statistical value, for example, the mean or median, or with a prediction of a data mining model).

Additionally, data sets may contain instances that deviate significantly from other instances. *Outliers*, as those instances are called, are either caused by noise or they represent exceptions to the distribution that generated the data set. Independent of the reason, the removal of outliers may be reasonable, as they may influence the induction process of data mining methods in a unfavourable way (e.g. trying to fit every instance correctly, including the outliers, may lead to overfitting). The detection of outliers is normally based on either statistical methods (e.g. the identification of unlikely observations given an assumed probability distribution) or methods that take the spatial proximity into account (e.g. nearest neighbour learning [Agg14]). Please note, that the detection and handling of noise may already solve the problem of outliers if their occurrence may be traced back to noise.

2.2.4 Step 4: Data Reduction & Transformation

The data reduction and transformation step concentrates on two issues: reduction of dimensionality and feature engineering [Pyl99]. *Data reduction* deals with the problem that machine learning methods may not scale very well with big databases as these learning algorithms often employ counting operations (e.g. calculating the frequency of specific feature values under given constraints) and/or multi-dimensional access to the database (e.g. selecting a subset of feature values and samples). This problem may be solved if the dimensionality of the data set is reduced to a size that may be stored in memory, or if the design of the data mining algorithm minimises the number of database accesses (e.g. Apriori, see Section 2.3.3.4). Obviously, there are two approaches for this reduction: selecting either a subset of instances or a subset of features. The first one is known as sampling [Pyl99], and the latter one as feature subset selection [Pyl99] which is essentially a part of feature engineering and will be discussed in the following paragraph. *Sampling* selects a subset or sample of instances of the data set by drawing randomly (with or without replacement) a (stratified) subset of instances from the data set. In any case, the drawing should avoid any systematic errors (e.g. preferring one sample to another) to prevent a biased subset of the data.

Feature engineering comprises three categories of techniques that address the selection and processing of features: the feature subset selection, feature transformation, and feature construction. *Feature subset selection* aims to remove redundant, uninteresting, or irrelevant features from the selected data set, possibly increasing the efficiency of the data mining step (e.g. the generation time may be polynomial or even exponential in the number of features) and the accuracy of the resulting model (e.g. by preventing overfitting). *Feature transformation* solves two issues. Occasionally, data mining methods are designed inherently to handle only specific feature categories (e.g. numerical or nominal ones) and are not able to process other categories for this reason. This problem may be solved by discretisation and numerisation of unsuitable features. *Discretisation* techniques transform numerical features into nominal ones, mapping number intervals to nominal values (e.g. the explicit age into age categories). *Numerisation* methods have the opposite effect, as they transform nominal features into numerical ones. Two-valued nominal features are transferred into binary numerical features representing the two feature values by 0 and 1. Multivalued nominal features are either transformed into multiple binary numerical features or into one multivalued numerical feature. Another issue of feature transformation is the *feature scaling* of numerical values. If a data set contains multiple numerical features or, in other words, has multiple dimensions, it may be reasonable (e.g. for distance-based data mining learning algorithms, such as nearest-neighbour learning [Agg14]) either to rescale and map a numerical feature to a pre-defined interval (e.g. [0, 1] or [-1,1]) or to standardise a numerical feature (e.g. translating it to have zero mean and unit variance). *Feature construction* derives new features by combining existing features (e.g. the body mass index is a function of a person's height and weight). In this way, data mining learning methods may exploit mutual information that could be too complex in its original representation to be described by a pattern or model.

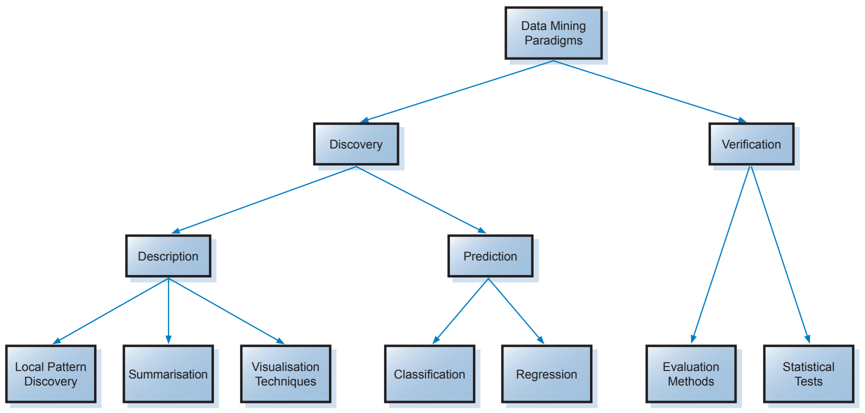


Figure 2.3.: Data mining paradigms: a simplified overview.

2.2.5 Step 5: Data Mining

In the previous steps (steps 1 to 4), we defined a knowledge discovery task for this KDD process and generated a data set step-by-step for it. Thus, we have everything at hand that we need for the next step: the application of data mining. *Data mining* searches in the explicit facts of the data set for interesting, previously unknown regularities, that represent bits of information or knowledge about the data and its domain, and formulates them as patterns. To this end, data mining provides a multitude of algorithms for the pattern extraction which originated, amongst others, in the fields of statistics, machine learning, and artificial intelligence. The data mining tasks that these algorithms solve are as diverse as their origin.

Since there are several data mining tasks designed for different purposes and even more data mining algorithms associated to each of these tasks, the data mining task and an appropriate algorithm have to be selected and fitted to the previously defined knowledge discovery task. Hence, the data mining step may be divided into three phases. First, an appropriate data mining task has to be selected. According to this selection, a data mining algorithm associated with the selected task has to be chosen and employed in the second and third step respectively. In the following paragraphs, we will briefly discuss these steps without going into the details of data mining and its algorithms.

Phase 1: Selection of the Data Mining Task

In the first step of the KDD process, the discovery task selected for this KDD process has been defined informally (in natural language). Now, the next step is to match this informal task to an existing data mining task. To this end, one or more data mining tasks may represent appropriate approaches for its solution. If more than one option is available, human experts have to decide which data mining task will be utilised in this step. This decision depends mainly on two factors: the performance of the data mining task (e.g. in terms of the quality of the extracted patterns or model, or the time consumption), and the personal preferences and experiences of the human experts.

Data mining (see Figure 2.3) and its tasks may be divided into two main categories: description and prediction (respectively referred to as descriptive and predictive data mining), that differ in their extraction process and the representation and purpose of the extracted knowledge or patterns. *Descriptive data mining* is orientated towards the interpretation and understanding of the data at hand. The extracted patterns describe certain properties of the data set and of its domain. *Predictive data mining* pursues a different goal as it searches for a global model that aims to predict the values of a specific feature of interest, referred to as class attribute, for new and previously unseen instances of this domain. In contrast to the patterns obtained by descriptive data mining, the global models of predictive data mining explain the data as a whole. Furthermore, global models range from black-box models (e.g. support vector machines [SC08]) to interpretable models (e.g. decision trees [RM14] and rule sets [FGL12]). Obviously, well interpretable models may provide a description of the data, too. Analogously, descriptive data mining algorithms may be used to obtain predictive patterns that may be combined to predictive models as we will see later in this chapter. Hence, the border between predictive and descriptive data mining algorithms is a bit fuzzy in this regard.

Furthermore, the general tasks of descriptive and predictive data mining may be divided into categories according to more specifically defined descriptive or predictive data mining tasks, respectively. Descriptive data mining encompasses amongst others *local pattern discovery* (e.g. the search for descriptive local patterns such as association rules or subgroups), *summarisation* methods (e.g. the calculation of mean or average values of a feature) and *visualisation techniques* (e.g. ROC space [Faw06]). Similarly, predictive data mining may be further divided into two categories, depending on the type of the predicted class attribute. *Classification* searches for models that predict nominal feature values. On the contrary, *regression* discovers models that return numerical values for prediction. In many cases different approaches to solve a data mining task have been developed. Algorithms that pursue a similar approach (e.g. utilising the same base methodology, patterns, or models) may be summarised in an algorithm family. However, the members of such a family differ in their concrete implementation of the approach. For example, the local pattern discovery algorithm families association rule mining and subgroup discovery both search for local patterns in form of rules, namely association rules and subgroups respectively, but their discovery processes, patterns and purpose differ. Analogously, classification may be solved amongst others by decision tree and rule learning algorithms that both generate global models - either decision trees or rule sets - for classification. As mentioned before, we will concentrate on two data mining categories, local pattern discovery and classification, and their approaches of association rule mining, subgroup discovery, and rule learning respectively.

Phase 2: Selection of the Data Mining Algorithm

In the previous phase, a data mining task has been selected to solve the knowledge discovery task. The next step will be the selection of a data mining algorithm associated with the chosen task. As mentioned before, there exist several categories of algorithms (e.g. classification problems may be processed by decision tree or rule learning) for each data mining task that solve it. Analogously, each algorithm category summarises several algorithms that share a similar model representation (e.g. association rules, classification rules, or decision trees) but work more or less differently.

The decision to employ one of these data mining algorithms or categories depends primarily on two factors: the quality of the induced patterns or model, and the interpretability of both. In respect to the interpretability, the data mining algorithm categories can be coarsely divided into black-box and white-box algorithms. Black-box algorithms (e.g. SVMs [SC08]) are tuned up for performance, though the internal patterns or models they generate are not represented in a form that is comprehensible for humans. On the contrary, white-box algorithms have also been designed for performance, but their model representations display the generated patterns and, if applicable, how they are used in the induced model. As a rule of thumb, up-to-date black-box algorithms tend to outperform their white-box counterparts at least slightly although this advantage is paid dearly by the loss of interpretability. Both aspects, the quality and the interpretability of the generated patterns and models, have to be traded off by human experts in respect to the given knowledge discovery task.

Data mining algorithms basically consist of three primary (algorithmic) components: the representation of the patterns and models, the evaluation of these objects, and the search for them. Additionally, data mining algorithms may include a overfitting avoidance component which may either be integrated in the search or may be applied separately. In the following, these components will be discussed briefly.

Representation: For each data mining algorithm, its representation is defined by a representation language for the desired patterns and, if applicable, for the induced model. Representation languages describe the valid values for the discovered patterns (e.g. conjunctive or disjunctive rules) or models (e.g. decision trees or decision lists) respectively. Obviously, patterns and models are usually drawn from different representation languages. Nevertheless, the pattern representation may influence the model representation, as a model may consist of a composition of extracted patterns (e.g. a decision list is made up of one or more rules).

The choice of the representation language for patterns and models has a major impact on the success of the data mining process. On the one hand, if the expressiveness of the chosen representation language is too limited, the resulting patterns and models may be too coarse or too general to cover all intricacies of the considered data set. On the other hand, a highly expressive representation language may increase the danger of overfitting the training data. Both may lead to a suboptimal accuracy on the present or unseen data. However, data mining algorithm families determine usually a specific base pattern or model that may be differently instantiated by its members. For example, classification rule learning demands the employment of rules as base patterns, though implementing algorithms may choose the specific kind of rules (e.g. conjunctive or disjunctive rules).

Evaluation: During the search for patterns and models, data mining algorithms have to be able to determine or at least estimate the quality of a considered pattern or model. For this purpose, data mining algorithms employ heuristic evaluation methods or functions that calculate a quantitative estimate for the quality of patterns or models. Based on this quality estimate, data mining algorithms may determine if a specific pattern or model is interesting (e.g. if the estimate exceeds a pre-defined quality threshold) for the current data mining task. Furthermore, data mining algorithms may compare two or more patterns (or models) by their estimated quality and select the most qualitative ones as these may be the most promising ones for the final results or further data mining steps.

Usually, data mining algorithms deploy several evaluation methods in their life cycle as at least one evaluation method is necessary for the evaluation of patterns and models respectively. Some data mining algorithms even utilise variable evaluation methods in different stages of their process. For instance, the classification rule learner Ripper which will be discussed later in this chapter employs differing evaluation methods depending on its current phase (e.g. training or pruning phase). Obviously, the diverse data mining tasks make differing demands on the extracted patterns and models and influence the reasonable approaches to evaluate these (e.g. predictive accuracy versus descriptive interestingness). Accordingly, evaluation methods may be designed for a given data mining task and for either patterns or models only.

Search: After choosing the representation language and the evaluation methods for patterns (and if applicable models), the pattern extraction process of a data mining algorithm may be reduced basically to an optimisation task: search in the domain of the chosen representation language for the patterns or models that optimise the selected evaluation methods (and criteria, e.g. a minimal interestingness threshold). Unfortunately, this is often not just a simple optimisation task that may be solved by a deterministic computation. Hence, data mining algorithms employ a heuristic search for patterns and/or models that approximate the optimal solution step-wise. Commonly, such a search starts with one or more candidate patterns (or models), refines them iteratively and chooses the best one(s) as its solution.

Consequently, this refinement process is defined by three components: the initialisation, the refinement, and the quantity of the considered candidate patterns (or models). Basically, candidate patterns are initialised either as the most specific or as the most general pattern in the chosen representation language. Depending on the chosen initialisation, a pattern is refined bottom-up by generalisation (removing one or more conditions) or top-down by specialisation (adding one or more conditions respectively). In each refinement step, the candidate patterns of the previous steps are refined and only one or more of the best patterns obtained in this way are stored for the next step. The exact number of stored candidates depends on the third component: the *search strategy*. In data mining, three search strategies or algorithms are commonly used: hill climbing, beam search, and exhaustive search. *Hill climbing* stores in each step only the best refined pattern. Similarly, *beam search* stores the k best patterns in each step and therefore is a generalisation of hill climbing. *Exhaustive search* enumerates all patterns that are representable in the chosen representation language.

Overfitting avoidance: As noisy data may lead to overfitted models that tend to be overly complicated and to have a low accuracy on unseen data, many learning algorithms employ methods to avoid it. The basic idea of these methods is to keep the induced rules and consequently the model simple, as a simple model may be more predictive than more complex ones. For this purpose, rule learning algorithms utilise pruning methods that stop the refinement process prematurely (known as pre-pruning), remove redundant conditions or unnecessary rules from the induced model (known as post-pruning), or integrate both pruning approaches (for example, incremental reduced error pruning [FW94]).

Phase 3: Employment of the Data Mining Algorithm

So far, a data mining task and an associated data mining algorithm have been selected in the previous phases. Next, the chosen algorithm will be employed to process the prepared data set. Most data mining algorithms and their respective implementations feature several parameters. Amongst others, those parameters modify the pattern extraction in different aspects: the search strategy used for finding the patterns, the evaluation heuristic(s) employed to approximate the quality of the patterns or model, or the pattern format utilised to represent the patterns. Since there are multiple options for each parameter, choosing a good or in the best case optimal parameter setup is not an easy task. Hence, it is common that several setups are considered. This approach involves multiple repetitions of this step, the evaluation and comparison of the results in the evaluation step, and the selection of the most promising setup afterwards.

2.2.6 Step 6: Evaluation

Depending on the selected data mining task, patterns have been extracted from the data set and if applicable a model has been induced using them. Before these patterns or the model may be installed in a commercial or scientific operational system or used for gaining insights into the domain, their quality (e.g. their accuracy, or memory and time consumption) should be evaluated first. Since an exact calculation of the quality is often not feasible in practice, approximations of the considered quality are required. For this purpose, evaluation methods, as described before in the data mining step, have been developed that approximate the quality of patterns and models or manage the evaluation (e.g. cross-validation) and comparison (e.g. statistical tests, or visualisation techniques) of models. Relevant evaluation methods are illustrated in detail in Section 2.3 for local patterns and in Section 2.4 for global models, respectively. Nevertheless, we will give a short overview over the evaluation here.

The evaluation of models, especially if the model is used for predictions, needs further considerations, as the simple employment of quality heuristics to rate models may be insufficient. Such ratings may be too overoptimistic, as the patterns may be fitted too much to the present data set but generalise badly on other data sets of the domain. As mentioned above, this phenomenon is commonly referred to as overfitting. To avoid or recognise overfitted models, evaluation techniques divide the available data set into two or more parts. Then, some of these parts (referred to as the training data set) are used to generate or train the model and the other parts (referred to as the testing or evaluation data set) are employed only to evaluate this model. In practice, there exist several modifications of this approach, the most common one is the *cross-validation* technique which will be discussed later in this chapter (see Section 2.4.4.1).

As mentioned in the previous step, it may be worthwhile to try out multiple data mining algorithms and choose the best performing one. In this case, all candidate data mining algorithms are evaluated, using the same quality heuristic and evaluation technique, and are compared mutually with either statistical tests (e.g. the sign test [Dem06]) or visualisation techniques (e.g. the ROC space [Faw06]) using their derived heuristic qualities (see Section 2.4.4.2). *Statistical tests* may be used to decide if the quality of two or more compared models either is significantly different, given a pre-defined significance level, or not. *Visualisation techniques* may be used to depict the quality of one or more patterns or models and allow to choose visually the best performing one, given specific constraints on their performance (e.g. cost-sensitive learning). The results of some statistical tests may be visualised by similar techniques, too.

If the patterns or model of the (best performing) data mining algorithm yield a sufficient high quality, the discovered knowledge is ready for the incorporation in an operational system.

2.2.7 Step 7: Utilisation of the Discovered Knowledge

In the previous steps, patterns and/or models have been extracted and evaluated. The next step is to incorporate the results of the optimal data mining algorithm in an operational system to solve the initially defined knowledge discovery task. For this purpose, either an operational system is designed from scratch or an existing one is modified, taking the newly discovered knowledge into account. In the integration process, the pre-processing techniques that have been employed in the KDD process have to be considered and applied to the operational data if necessary. Additionally, further challenges may arise as the KDD process has extracted the patterns or the model under laboratory conditions. For example, the data set reflects only a snapshot of the domain, but the domain and the data that describes it may undergo dynamic changes (e.g. features and their respective values may change or even become unavailable). In this case, the integration may have to be adjusted or in the worst case, the KDD process has to be restarted.

2.3 Local Pattern Discovery

In the previous section, we briefly described the KDD process and emphasised its main step data mining. In this section, we concern ourselves with the data mining task of local pattern discovery. To this end, we will first (informally) define local patterns and all concepts that are necessary in this context. Next, we introduce the local pattern discovery task and show how local patterns may be induced by a general discovery approach. At last, we describe two families of local pattern discovery algorithms, subgroup discovery and association rule mining, and two correspondent algorithms that we will later consider in our experiments.

Let us start with the definition of local patterns. As previously mentioned, we utilise decision rules of the following form $X \implies Y$ for the representation of local patterns. In general, the *body* or *premise* X and the *head* or *consequent*

Y of a rule may consist of one or more attribute-value pairs which we refer to as *conditions* or *features*

$$A_i \circ a_{j,i}, \tag{2.12}$$

where A_i is one of the attributes of the dataspace S, $a_{j,i} \in dom(A_i)$ is one of its permissible attribute values, and

$$\circ \in \{<, \leq, =, >, \geq, \neq\} \tag{2.13}$$

is a comparison operator whose actual selection is determined by the type of attribute A_i. For nominal attributes, the comparison $\circ$ is a test of (in-)equality ($=$ or $\neq$), whereas in the case of numerical attributes, the test may also be less than (or equal) or greater than (or equal) a constant numerical value. In this work, we restrict the head of the considered rules we induce in our experiments to a single attribute-value pair, as the intention of these induced rules is the prediction of values of the class attribute. Additionally, only the predicted class label is denoted in the rule head since the class attribute is already determined by the learning problem. As mentioned above, the body of an induced rule consists of a set of features which we employ as a conjunction of conditions. In summary, a conjunctive classification rule r has the following form:

$$\underbrace{A_{i_1} \circ a_{j_1,i_1}}_{condition_1} \wedge \cdots \wedge \underbrace{A_{i_{|r|}} \circ a_{j_{|r|},i_{|r|}}}_{condition_{|r|}} \Longrightarrow l_r \tag{2.14}$$

where $i_k \in \{1, \cdots, |S|-1\}$, $j_K \in \{1, \cdots, \left|A_{i_k}\right|\}$ and the predicted class value $l_r \in L$. Accordingly, the *length or size of a rule* $|r|$ is equal to the number of its conditions.

A set of local patterns which we refer to either as a *pattern set* or *rule set* is denoted as follows

$$R = \left\{r_1, \cdots, r_{|R|}\right\}, \tag{2.15}$$

where $|R|$ is the number of rules.

An instance d is said to be *covered* by the rule r (denoted by $r \supseteq d$), if each $condition_k$ of the rule is met by d ($a_{d,k} \circ a_{j_k,i_k}$). In this case, the class value of the rule is predicted for the covered instance. Similarly, the rule r is called a *covering rule* for this instance. For a given set of instances D and a rule r, the instances that are covered by this rule are denoted by

$$D_r = \{d \in D | r \supseteq d\} \tag{2.16}$$

Hence, the number of covered instances is defined as

$$n_r = |D_r| \tag{2.17}$$

For each class l_i, the number of instances n_{r,l_i} that are covered by a given rule r is defined as follows:

$$n_{r,l_i} = \left|D_{l_i} \cap D_r\right| \tag{2.18}$$

For the predicted class l, we define the total number of positive instances

$$P = |D_l| \tag{2.19}$$

and the number of covered positive instances

$$p = n_{r,l} \tag{2.20}$$

that are classified correctly by the rule. Analogously, we denote the total number of negative instances

$$N = |D| - P \tag{2.21}$$

and the number of covered negative instances

$$n = n_r - p \tag{2.22}$$

that are classified incorrectly.

The subset of patterns of the pattern set R that cover a given instance d is denoted by

$$R(d) = \{r \in R | r \supseteq d\} \tag{2.23}$$

As mentioned before, patterns or models are rated by heuristic quality functions for diverse purposes. The calculation of these heuristics is usually based on one or more statistical values, and the four numbers p, n, P and N are usually sufficient for the calculation of many of these quality functions. Essentially, such a quality function h calculates a numerical value for patterns or models. Usually, the higher this numerical value is, the better is the quality of the considered object. Let us illustrate this rather abstract description by the exemplification of a few well-known quality heuristics that are used in this work.

The first quality function is *precision* or *confidence* which essentially calculates the percentage of positive instances among covered instances:

$$h_{prec}(r, D) = \frac{p}{p+n} \tag{2.24}$$

A drawback of precision is that it is prone to overfitting. For example, covering one positive instance and no negative ones is as good as covering a thousand positive instances and no negative ones.

The second quality function, the *Laplace-corrected precision* (or short the Laplace function), remedies this effect by raising the number of positive and negative covered instances by one. In this way, it calculates the quotient of the number of correctly covered instances plus one and the total number of covered instances plus two:

$$h_{laplace}(r, D) = \frac{p+1}{p+n+2} \tag{2.25}$$

A third quality function is *accuracy* which determines the percentage of correctly classified instances (the total of covered positive and uncovered negative instances):

$$h_{acc}(r, D) = \frac{p+(N-n)}{P+N} \tag{2.26}$$

Hence, covering one positive instance is as good as not covering one negative instance. For comparisons, the difference between positive and negative covered instances may be used $(p-n)$.

The fourth and last quality function is *weighted relative accuracy* which normalises accuracy with the class distribution:

$$h_{wra}(r, D) = \frac{p+n}{P+N}\left(\frac{p}{p+n} - \frac{P}{P+N}\right) \tag{2.27}$$

Thus, covering one percent of positive instances is as good as not covering one percent of negative instances

At last, we introduce two special patterns that are often used as the starting point of the local pattern discovery: the most specific pattern

$$\mathit{false} \implies l \tag{2.28}$$

that covers no instances at all and the most general pattern

$$\mathit{true} \implies l \tag{2.29}$$

that covers all instances of the domain. The latter one is usually referred to as the empty pattern or rule.

2.3.1 Learning a Single Pattern

Prior to this, we introduced the relevant definitions and properties of data and local patterns. Now, we will show how local patterns may be derived from a given data set. For this purpose, we illustrate the associated problem by a basic pattern discovery algorithm that provides a simple solution [Für99]. We show that this simple approach possesses all the main components of a data mining algorithm and that it is an instantiation of a more general pattern discovery algorithm that we describe afterwards.

Essentially, the problem of local pattern discovery may be defined as follows:

Algorithm 1 SimpleFindBestPattern [Für99]

```
 1: procedure SimpleFindBestPattern($Instances$)
 2:     $BestRule$ = EmptyRule()
 3:     $Rule = BestRule$
 4:     repeat
 5:         $NoImprovement$ = true
 6:         for $Condition \in Conditions$ do
 7:             $Refinement = Rule \cup Condition$
 8:             if Precision($Refinement$) > Precision($BestRule$) then
 9:                 $BestRule = Refinement$
10:                 $NoImprovement$ = false
11:         $Rule = BestRule$
12:     until NoImprovement
13:     return $BestRule$
```

Find the best pattern(s) according to a selected quality measure.

While this problem statement seems to be simple, the search for a solution may turn out to be difficult. Finding an optimal solution may only be guaranteed, if the pattern space is searched exhaustively for it. However, an exhaustive search may not be feasible as e.g. the pattern space may be too large. Additionally, the solution on the training data does not have to be necessarily optimal on future, unseen data. For this reason, various local pattern discovery algorithms were developed to find heuristically a solid solution that also performs well on unknown data. The majority of these algorithms may be reduced essentially to a common discovery approach and represent specific instantiations of the associated generic algorithm. Before we deal with this generic algorithm, we introduce a simplified algorithm that is also an instantiation of the generic one. Both algorithms extract local patterns only for a single chosen class since they were designed to solve binary learning problems. For this purpose, they generate rules for one class, the positive class, and no rules for the other, the negative class. Their instances are treated as positive and negative instances respectively for quality evaluations. Consequently, a rule is completely defined by the set of its conditions. Nevertheless, these algorithms may be used to generate patterns for all classes of a binary or multi-class learning problem as we will see later in Section 2.4.3.

The base idea of the simplified local pattern discovery algorithm SimpleFindBestPattern (see Algorithm 1) is to refine the most general rule step-by-step until its precision may not be further improved. For this purpose, this algorithm starts with the initialisation of two rules, the current considered rule *Rule* and the best rule *BestRule* found so far, as the most general rule. Afterwards, *Rule* is refined by adding one of the available conditions. In this way,

Algorithm 2 FindBestPattern [Für99]

```
 1: procedure FINDBESTPATTERN($Instances$)
 2:     $BestRule = \textsc{InitialiseRule}(Instances)$
 3:     $BestQuality = \textsc{EvaluateRule}(BestRule)$
 4:     $Rules = \{BestRule\}$
 5:     while $Rules \neq \emptyset$ do
 6:         $Candidates = \textsc{SelectCandidates}(Rules, Instances)$
 7:         $Rules = Rules \setminus Candidates$
 8:         for $Candidate \in Candidates$ do
 9:             $Refinements = \textsc{RefineRule}(Candidate, Instances)$
10:             for $Refinement \in Refinements$ do
11:                 $Quality = \textsc{EvaluateRule}(Refinement, Instances)$
12:                 if not $\textsc{StoppingCriteria}(Refinement, Quality, Instances)$ then
13:                     $\textsc{InsertSort}(Rules, Refinement)$
14:                     if $Quality > BestQuality$ then
15:                         $BestRule = Refinement$
16:                         $BestQuality = Quality$
17:         $Rules = \textsc{FilterRules}(Rules, Instances)$
18:     return BestRule
```

we obtain several more specific refinements of *Rule*. If one of these refinements has a higher precision than the current best rule, *BestRule* is replaced by this refinement. At last, *Rule* is set to *BestRule*. This refinement loop is repeated until no refinement of the current *Rule* has a higher precision than *BestRule*. When the algorithms stops, the best rule *BestRule* is returned as the result of the algorithm.

Despite its simplicity, this simple algorithm features all the components of a data mining algorithm as defined in Section 2.2.5. Each of its steps and employed methods are related to one or more of these components. The initialisation of the most general pattern and the refining of patterns by adding conditions depend on the chosen representation language, conjunctive rules, and the chosen refinement strategy, the top-down refining principle. The evaluation of the pattern quality is done by the quality method precision. The number of candidate rules, their replacement, and the stopping criterion are due to the selected search strategy, hill climbing. Clearly, the components influence the specific instantiations of the other components (e.g. the refining of patterns depends both on the representation and search). In summary, SIMPLEFINDBESTPATTERN is a top-down hill-climbing rule learner that generates local patterns in the form of conjunctive rules.

Since there are many more choices for the components of a data mining algorithm, a more generic algorithm is needed that allows to model (almost) every configuration of components. For this purpose, we present the local pattern discovery algorithm FINDBESTPATTERN (see Algorithm 2) which is an adjusted version of FindBestRule in [Für99]. FINDBESTPATTERN searches the pattern space for a single rule that optimises a given heuristic quality function (computed by the procedure EVALUATERULE). For this purpose, it considers one or more so-called candidate rules for refining and stores one or more of the resulting refinements for the next iteration. The procedures INITIALISERULE and EVALUATERULE are used to initialise and rate the initially best rule *BestRule* which is used to set-up the initial candidate rule list *Rules*. Using the candidate rule list *Rules*, the inner loop refines the candidate rules as long as *Rules* is not empty. At the beginning of each cycle, the procedure SELECTCANDIDATES selects a subset of candidate rules from *Rules*. For each selected candidate rule *Candidate*, the procedure REFINERULE determines all possible refinements which are evaluated by EVALUATERULE. If a refinement *NewRule* is not discarded by the procedure STOPPINGCRITERIA, it is inserted by the procedure INSERTSORT. If the heuristic quality of *Candidate* is better than that of *BestRule*, *NewRule* is stored as the new *BestRule*. Similarly, its quality replaces the current *BestQuality*. When all candidates have been processed in this way, the procedure FILTERRULES selects a subset of candidates for the next iteration. When no candidate rules remain, the repetition ends and the current *BestRule* is returned.

Similar to SIMPLEFINDBESTPATTERN the procedures of FINDBESTPATTERN may be related to the main components of a data mining algorithm as follows. The procedures INITIALISERULE and REFINERULE determine the refinement or search strategy as they define how the candidate rules are initialised and refined. Additionally, they constitute the employed pattern representation of the algorithm. The search strategy depends on the procedures SELECTCANDIDATES and FILTERRULES since they determine which candidates are considered for the refinement process. The search heuristic is provided by the procedure EVALUATERULE which computes the heuristic quality of a rule. The overfitting avoidance is accomplished by the pre-pruning procedure STOPPINGCRITERIA.

2.3.2 Subgroup Discovery

In this section, we will deal with the first considered category of local pattern discovery: subgroup discovery [NLW09]. To this end, we will describe briefly the motivation, relevant terms and properties, and the associated learning problem of subgroup discovery. Afterwards, we show exemplarily how this problem may be solved by BSD [LRA10], a bit-set based algorithm, which we later use in our experiments.

2.3.2.1 Motivation

In the medical domain, the identification of high-risk groups for a given disease may be highly valuable as new insights in the causes of disease (e.g. a patient's predisposition, diet, lifestyle, or other factors) may be gained by the analysis of the identified subgroups of patients. In this way, future afflictions of people/patients may be predicted and at the best prevented if possible. Clearly, such subgroups should consist only of patients that differ significantly in certain discriminatory characteristics from the healthy people. Interestingly, some patients may belong to multiple relevant subgroups as they may exhibit multiple risk factors.

The underlying data mining goal of this example may be summarised as follows:

> *Find descriptions for subgroups of the data, that are most unusual with respect to a specified concept of interest.*

The concept of subgroup discovery which is a well-known data mining method for the discovery of local patterns was introduced in [Klö96, Wro97] to handle this data mining task. The goal of subgroup discovery is to describe the most unusual characteristics of a subset of the data, referred to as population in this context, that is defined by an attribute-value pair (referred to as the concept of interest). Such characteristics are unusual in the sense that they respectively represent a differing behaviour of a subgroup of the considered population with respect to the residual data. To this end, subgroup discovery searches for interesting subgroup patterns that describe relations between the concept of interest and a set of explaining conditions. Hence, subgroups are usually described by conjunctive rules.

2.3.2.2 Properties of Subgroup Discovery

As mentioned before, a *population* is defined as a subset of the data that is determined by a single attribute-value pair or feature (referred to as the *concept of interest* in the context of subgroup discovery). In other words, a population consists of all instances that are covered by a given feature. Subgroup discovery searches for unusual *subgroups* of this population that exhibit different properties in comparison to the residual data. Such a subgroup is determined by a single *subgroup description* which consists of a conjunction of explaining conditions. The subgroup description and the concept of interest compose together a subgroup pattern that may be modelled by a conjunctive rule. The explaining conditions form the body of the rule, and the concept of interest may be regarded as the rule head. Subgroup discovery may be easily adjusted to the discovery of local classification patterns. To this end, a class-label pair $L = l$ is chosen as the concept of interest. Under these circumstances, a subgroup discovery algorithm will generate classification rules that predict the chosen class label. In addition to these properties, all the terms, properties, statistical values, and functions that were described for general local patterns are directly applicable to subgroups (see beginning of Section 2.3).

Since subgroup discovery usually searches for the (k) most interesting subgroups in respect to the concept of interest, the quality of subgroups has to be determined. Although diverse heuristic quality functions for local patterns may be employed to measure the interestingness of subgroups, subgroup discovery algorithms usually employ quality functions that adhere to the monotonicity axioms introduced in [Klö96]. These axioms imply that a quality function should be monotone in the size of a subgroup and the frequency of the concept of interest in the subgroup. Additionally, such quality functions may not only be used to rate the current considered subgroup but also to compute optimistic estimates for it [GRW08, AL09]. Such optimistic estimates are upper boundaries for the maximally achievable quality of the possible refinements of a subgroup. With the help of the optimistic estimates, large parts of the considered pattern space may be pruned as unpromising areas may be ignored without loss of performance.

Subgroup discovery algorithms usually search for subgroup patterns only based on the measured quality of the individual subgroups and do not take into account the redundancy and overlap between these patterns. A typical k-best subgroup discovery algorithm does not employ covering approaches, the discovered pattern set may consist of very similar, overlapping subgroups. Additionally, further potentially more interesting and diverse subgroups are hidden from the user, as only the k "best" but overlapping and irrelevant subgroups are returned. Clearly, additional techniques (e.g. filtering of overlapping subgroups) have to be employed to prevent this loss of information.

For this purpose, the *concept of irrelevancy* has been introduced. Given two subgroup patterns s_i and s_j, s_j is irrelevant with respect to s_i, if and only if the positive instances covered by s_j are a subset of the positive instances covered by s_i and the negative instances covered by s_i are a subset of the negative instances covered by s_j:

$$D_l \cap D_{s_j} \subseteq D_l \cap D_{s_i} \wedge (D \setminus D_l) \cap D_{s_i} \subseteq (D \setminus D_l) \cap D_{s_j}. \tag{2.30}$$

For any quality function that satisfies the Kloesgen axioms in [Klö96] holds that the quality of s_j is lower than that of s_i, if s_j is irrelevant to s_i.

Usually, subgroup discovery includes a relevancy check and a consequential removal of irrelevant subgroups by filtering. There exist two standard approaches for such a relevancy check: filtering irrelevant subgroups as a post-processing step and filtering during the search phase. The post-processing approach allows the employment of standard subgroup discovery algorithms that do not have to be modified to integrate a relevancy check. After the extraction of subgroups, a separate post-processing step removes the irrelevant subgroups. Clearly, the advantage of this approach is that the discovery process is not slowed down by an integrated relevancy check, but the size of the resulting set of subgroups is more or less unpredictable as the relevancy of induced subgroups depends on the data set. Consequently, larger sets of subgroups have to be extracted to assure that at least k relevant subgroups remain.

To remedy this problem, the second filtering approach incorporates a relevancy check in the search process by either including it in the design of a new algorithm or by modifying existing ones. After the generation of a subgroup, two relevancy checks are applied. The first one checks if the new subgroup is irrelevant to any previously generated subgroups. If this is not the case, the new subgroup is added to the set of found subgroups and the second relevancy check takes place. The second one checks the relevancy of the previously found subgroups in respect to the newly found subgroup, removing the irrelevant ones. In this way, multiple subgroups could be removed by adding one more relevant subgroup, but in practice the reduction of the result size is normally very limited. Clearly, the bottleneck of this approach are the relevancy checks after the generation of each subgroup. For instance, a naïve implementation of the relevancy check requires one complete pass over the database [LRA10]. Hence, the relevancy check may be more time-intensive than the actual search component of the employed subgroup discovery algorithm.

2.3.2.3 Bitset-based Subgroup Discovery Algorithm (BSD)

In our work, we employed the bitset-based subgroup discovery algorithm (abbreviated BSD) that was introduced in [LRA10] for the generation of subgroup patterns. BSD is a novel branch-and-bound algorithm that efficiently extracts the k-best relevant subgroups (of a pre-defined maximal length). For this purpose, it incorporates an efficient data structure, an integrated relevancy check, and a branch-and-bound strategy.

BSD employs bitset vectors as a vertical data structure that encodes bitwise, for every feature value, which instances exhibit this value. The employment of bitset vectors is motivated by two advantages over traditional horizontal data structures (e.g. relational table). First, all necessary bitset vectors may be generated by a single database pass, making additional database lookups unnecessary. Second, all bitwise operations on bitset vectors are very time and memory efficient. Consequently, the time consumption of all necessary calculations, including the integrated relevancy check, is significantly reduced.

BSD employs a branch-and-bound strategy that explores the pattern space by a depth-first search. This strategy utilises the previously mentioned optimistic

Algorithm 3 BSD [LRA10]

```
1: procedure INITIALISE($Instances$, $k$, $length_{max}$)
2:     $Conditions = \text{INITIALISEBITSETVECTORS}(Instances)$
3:     $Rules = \emptyset$
4:     $\text{BSD}(\text{EMPTYRULE}(), Conditions, 1, Rules, length_{max})$
5:     return $Rules$
1: procedure BSD($Rule$, $Conditions$, $Length$, $Rules$, $length_{max}$)
2:     $Conditions_{new} = \emptyset$
3:     for all $Condition \in Conditions$ do
4:         if $\text{OPTESTIMATE}_{Bit}(Condition, Rule) > \text{MINQUALITY}(Rules)$ then
5:             $Conditions_{new} = Conditions_{new} \cup \{Condition\}$
6:             if $Quality(Condition, Rule) > \text{MINQUALITY}(Rules)$ then
7:                 $Refinement = \text{REFINESUBGROUP}_{Bit}(Condition, Rule)$
8:                 if $\text{REFINEMENTISRELEVANT}_{Bit}(Refinement, Rules)$ then
9:                     $Rules = Rules \cup \{Refinement\}$
10:                    $\text{REMOVEIRRELEVANTSGS}_{Bit}(Rules, Refinement)$
11:                    if $\text{SIZE}(Rules) > k$ then $Rules = \text{KBEST}(Rules)$
12:    if $Length < length_{max}$ then
13:        $\text{SORT}(Conditions_{new})$
14:        for all $Condition \in Conditions_{new}$ do
15:            $Conditions_{temp} = Conditions_{new} \setminus Condition$
16:            if $\text{OPTESTIMATE}_{Bit}(Condition, Rule) > \text{MINQUALITY}(Rules)$ then
17:                $Refinement = \text{REFINESUBGROUP}_{Bit}(Condition, Rule)$
18:                $\text{BSD}(Refinement, Conditions_{new}, Length + 1, Rules, length_{max})$
```

estimate for future refinements of a considered subgroup to prune whole areas of the pattern space that in no case may improve the quality of the search result. In this way, the resulting set of subgroups will not lose any relevant subgroups, but the search process may be performed significantly faster.

In the following, we will describe the steps of BSD sketchily (its simplified pseudocode is shown in Algorithm 3), for a more comprehensive description we refer to [LRA10]. BSD searches for the k-best subgroups that are confined by a pre-defined maximal length which is determined by the parameter $length_{max}$. For an easier representation of the process, we assume that these parameters, k and $length_{max}$, are globally available.

BSD starts with initialising the necessary bitset vectors. For each condition, two bitset vectors are generated that encode which instances are covered by the condition (one bit per condition). The first one encodes the positive instances, the second one encodes the negative instances respectively. Using these bitset vectors, the statistical values (e.g. the number of positive or negative covered instances) may be easily computed. Functions whose operations benefit from the bitset vectors are marked with the index *Bit*. After the initialisation, the main procedure BSD is executed, using an empty rule, the calculated bitset vectors, a length of one, and an initially empty result set as its parameter.

The procedure BSD consists of two parts. In the first one, each of the available conditions is considered for refining the current rule. If the optimistic estimate of the resulting refinement (without being generated) is lower than or equal to the lowest quality in the current rule set, the condition is discarded and the next condition may be considered. Otherwise, the condition is stored in the set of promising conditions and its processing continues. If the quality of the associated refinement is higher than the lowest quality in the current rule set, the refinement of the current rule and the condition is computed. If the refinement is relevant with respect to the current result set, the refinement is added to it and all rules in result set that are irrelevant with respect to the refinement are removed. At last, the result set is reduced to the k best rules, if its size is larger than k. When these steps have been executed for all conditions, the second part takes place.

The steps of the second part are performed if the current length is below the pre-defined maximum rule length $length_{max}$. If so, the set of promising conditions is processed. For this purpose, it is sorted in descending order according to the individual quality of its members. Afterwards, the optimistic quality of each condition is compared to the current lowest quality in the result set (which may have increased in the meantime). If the quality of a condition is still higher, the procedure BSD is called recursively, using the following adjusted parameters. To this end, the refinement of the current rule and the considered condition is calculated and used as the starting point for new refinements. The condition is then removed temporarily from the conditions that are available for the next recursion. The length is increased by one.

2.3.3 Association Rule Mining

In this section, we will focus on the second considered category of local pattern discovery: association rule mining. To this end, we will describe briefly the motivation, relevant terms and properties, and the associated learning problem of association rule mining. We explain how association rule mining algorithms may be used to generate local patterns for classification. Afterwards, we show exemplarily how this problem may be solved by the Apriori algorithm and summarise the approach of the CHARM algorithm which we later employ in our experiments.

2.3.3.1 Motivation

Retail organisations and commercial websites are nowadays able to collect and store large amounts of associated shopping data, referred to as market basket data. Records in such data typically consist of shopping items, products or services, that were bought or paid in a respective transaction. In this context, a transaction does not have to occur at a single point in time but may consist of items that a customer has bought over a prolonged time period. For example, the transaction of services that are paid at the end of an accounting period (e.g. video-on-demand or book/music online stores).

Many organisations consider this shopping data as an opportunity to obtain a better understanding of the purchase behaviour of their customers. Using the obtained knowledge, the marketing departments may improve their marketing concepts (e.g. cross-selling or up-selling of complementary products) and campaigns (e.g. sales promotions of or discounts on a product may increase the sales of other related products) by tailoring them to the needs and wishes of their customers. Additionally, the store or catalogue design of retail stores or online shops may be improved based on the knowledge about buying patterns.

In the light of their value and usefulness, the problem of mining these buying patterns, in the form of association rules, over basket data was introduced in [AIS93]. An informal example of such an association rule might be that 95 percent of the customers that bought the first two volumes of the Lord of the Rings also bought the third volume. Obviously, such rules are, as mentioned before, very easy to interpret and thus may be easily utilised for the information-driven development of customised marketing.

2.3.3.2 Foundations of Association Rule Mining

In the following, we will formulate the properties of market basket or transaction data and association rules and introduce the associated problem of mining the later. As mentioned before, a market basket consists of a subset of the products and/or services that a vendor offers. These products and services are uniformly referred to as *items*. Let

$$I = \left\{i_1, i_2, \cdots, i_{|I|}\right\} \tag{2.31}$$

be the finite set of all items. A set of items $X \subseteq I$ is called an *itemset*. Furthermore, the size of an itemset is equal to the number of items it contains ($|X|$). An itemset of size k is referred to as a *k-itemset*. Since itemsets are mathematical sets, the standard set operations (e.g. union or intersection) may be applied to them. The items that were bought in one accounting period are summarised in a single transaction. The market basket data is made up of such transactions. Let

$$T = \left\{t_1, t_2, \cdots, t_{|T|}\right\} \tag{2.32}$$

be a set of transactions. Each *transaction* t_i is an itemset so that $t_i \subset I$. An association rule

$$X \implies Y \tag{2.33}$$

consists of two itemsets $X \subset I$ and $Y \subset I$. Similarly to classification rules, X forms the body or antecedent of the rule and Y is the head or consequent of the association rule.

To determine the quality of itemsets or association rules, we introduce two heuristic quality measures: support and confidence. The *support* of an itemset is equal to its absolute or relative frequency in the considered market basket data (set of transactions) T:

$$Sup_{abs}(X) = |\{t \in T | X \subseteq t\}| \qquad Sup_{rel}(X) = \frac{Sup_{abs}(X)}{|T|} \tag{2.34}$$

Please note that the absolute and relative support may be used interchangeably as long as only one of these methods is used in the same context (e.g. for comparison purposes). Hence, we will not distinguish between these two methods in the following and omit the identifying indices.

Itemsets and rules that have a support greater than a pre-defined minimum support threshold sup_{min} are called *frequent* (or *large*). The set of all frequent itemsets is defined as

$$F = \left\{X \subseteq I | Sup(X) \geq sup_{min}\right\} \tag{2.35}$$

Correspondingly, F_k denotes the set of all frequent k-itemsets.

The support of an itemset possesses a helpful property with respect to its frequency, it is anti-monotone according to the number of items it contains. For any itemsets X and Y holds:

$$\begin{aligned} Sup(X \cup Y) &= |\{t \in T | (X \cup Y) \subseteq t\}| \\ &= |\{t \in T | X \subseteq t\} \cap \{t \in T | Y \subseteq t\}| \\ &\leq |\{t \in T | X \subseteq t\}| \\ &= Sup(X) \end{aligned} \tag{2.36}$$

This means that all subsets of a frequent itemset must be frequent, too. As we will see later, this property turns out to be very useful to reduce the computational effort for the frequent itemset generation.

The *support* of a rule $X \implies Y$ is defined as the support of the union of its antecedent and consequent:

$$Sup(X \implies Y) = Sup(X \cup Y) \tag{2.37}$$

The *confidence* of a rule $X \implies Y$ measures the validity of the rule on the data by the ratio of the transactions that contain antecedent X of the rule to the transactions that contain both the antecedent X and the head Y:

$$Conf(X \implies Y) = \frac{Sup(X \cup Y)}{Sup(X)} \tag{2.38}$$

Rules that have a confidence greater than a pre-defined minimum confidence threshold $conf_{min}$ are called *confident*.

Similarly to the support of an itemset, the confidence of an association rule is anti-monotone in the size of its consequent. For any itemsets X, Y, and Z holds:

$$\begin{aligned} Conf(X \cup Y \implies Z) &= \frac{Sup(X \cup Y \cup Z)}{Sup(X \cup Y)} \\ &\geq \frac{Sup(X \cup Y \cup Z)}{Sup(X)} \\ &= Conf(X \implies Y \cup Z) \end{aligned} \tag{2.39}$$

Analogously to the anti-monotonicity of the support, this property may be exploited to make the generation of confident rules more efficient as we will see later.

According to the introduced notions, the problem of mining association rules may be formulated as follows:

> *Given a set of transactions T, generate all association rules whose support and confidence are greater than the (user-specified) minimum support and confidence thresholds.*

Before the generation of the association rules may start, these thresholds have to be specified. In other words, the problem of association rule mining is to search for frequent, confident rules.

The discovery of such association rules can be decomposed into two sub-problems or phases:

1. Find all frequent itemsets: in the first phase the given transaction data is (efficiently) searched for frequent itemsets.

2. Generate all confident rules: the frequent itemsets found are used to generate all confident association rules.

This decomposition into two sub-problems that has been introduced in [AIS93] is a widely used approach for association rule mining. The best known representative for an association rule mining algorithm that employs this approach is the Apriori algorithm which we will describe later (see Section 2.3.3.4).

2.3.3.3 Class Association Rules

So far, we assumed a transaction database for the discovery of general association rules. However, any structured (classification) data set may be used for the extraction of these local patterns, and association rule mining algorithms may be employed to obtain association rules for classification which are commonly referred to as class association rules [LHM98]. In this section, we explain briefly how to deal with these two issues.

Structured data sets may be easily transformed into transaction data sets that consist of transactions and items. To this end, each instance of a structured data set is treated as a transaction of the transformed data set. If necessary, the transactions obtained in this way are extended by a unique transaction identifier. The number of transactions is equal to the number of instances. Additionally, each attribute-value pair (including the class attribute and its labels) of the original data set is treated as an item of the transformed one. An item is contained by a transaction, if the associated attribute-value pair covers the associated instance.

After the transformation of a structured classification data set, the employed association rule mining algorithm or its starting point respectively have to be slightly adjusted, too. There are several approaches for this adjustment, but we will concentrate on the one that we employ in our experiments. For this purpose, we change the starting point of our algorithms. Instead of starting with 1-itemsets, our algorithm runs receive a set of modified 2-itemsets. Such an itemset is initialised with one item related to a class-label pair and the other one related to an attribute-value pair of a non-class attribute. In this way, it is guaranteed that each obtained frequent itemset contains exactly one class-label pair and may be used to build a classification rule for this reason.

2.3.3.4 Apriori

The Apriori algorithm has been introduced to mine association rules efficiently on large databases [AS94]. By the time it was introduced, it was the state-of-the-art association rule miner being considerably faster than contemporary algorithms [AS94], and even nowadays it may be considered as a benchmark for other association rule mining algorithms. Apriori refined the two phases approach that was introduced for the AIS algorithm [AIS93] by notably reducing the number of considered candidate itemsets and the number of database lookups for the support calculation consequently.

The first phase of the Apriori algorithm Apriori-FreqSet (see Algorithm 4) generates the set of all frequent itemsets F for a pre-defined minimum support threshold sup_{min}. For this purpose, it generates first all 1-itemsets, calculates

Algorithm 4 Apriori - Frequent Itemset Generation [AS94]

```
1:  procedure APRIORI-FREQSET($I$, $sup_{min}$)
2:      $C_1 = \{X \in I\}$
3:      $F_1 = \text{FREQUENTITEMSETS}_{DB}(C_1, T, sup_{min})$
4:      $F = F_1$
5:      $k = 1$
6:      while $F_k \neq \emptyset$ do
7:          $k = k + 1$
8:          $C_k = \text{GENERATECANDIDATES}(F_{k-1})$
9:          $C_k = \text{FREQUENTITEMSETS}_{Subset}(C_k, F_{k-1})$
10:         $F_k = \text{FREQUENTITEMSETS}_{DB}(C_k, T, sup_{min})$
11:         $F = F \cup F_k$
12:     return $F$
```

their support on the transaction data and keeps only the frequent ones. Afterwards, it determines recursively the set of all frequent k-itemsets F_k (where $k \geq 2$) as long as frequent (k-1)-itemsets have been found in the previous iteration ($F_{k-1} \neq \emptyset$). This iterative generation of frequent itemsets consists basically of three steps.

First, the frequent (k-1)-itemsets F_{k-1} found in the previous pass are used to generate the candidate itemsets C_k. To this end, Apriori combines all (k-1)-itemsets

$$a = \{a_1, a_2, \cdots, a_{k-1}\} \tag{2.40}$$

and

$$b = \{b_1, b_2, \cdots, b_{k-1}\}, \tag{2.41}$$

that share the same first k-2 items ($a_i = b_i, i \leq k-2$) but differ in their (k-1)-th item ($a_{k-1} < b_{k-1}$), into candidate itemsets

$$c = \{a_1, \cdots, a_{k-1}, b_{k-1}\}. \tag{2.42}$$

This approach guarantees that all possibly frequent candidates are found exactly once as it makes use of two properties of the considered itemsets. On the one hand, we do not miss any frequent k-itemset as each has at least two frequent subsets of size k-1 due to the anti-monotonicity of support. On the other hand, we do not generate duplicates since the items of each itemset are ordered by their lexicographical order and so each frequent k-itemset may be generated only by the combination of a certain pair of itemsets.

In the second step of the itemset generation, the previously extracted candidates C_k are reduced by removing those itemsets that cannot be frequent due

Algorithm 5 Apriori - Generate Confident Association Rules [AS94]

```
1: procedure APRIORI-CONFRULES($F$, $conf_{min}$)
2:     $Rules = \emptyset$
3:     for all $f \in F \setminus F_1$ do
4:         $Rules = Rules \cup$ APRIORI-CONFRULES($f, f, conf_{min}$)
5:     return $Rules$
1: procedure APRIORI-FREQSET(f, a, $conf_{min}$)
2:     $Rules = \emptyset$
3:     $RuleBodies = \{b \subset a | \; |b| = |a| - 1\}$
4:     for all $RuleBody \in RuleBodies$ do
5:         $RuleHead = f \setminus RuleBody$
6:         $Rule = (RuleBody \implies RuleHead)$
7:         if $Conf(Rule) \geq conf_{min}$ then
8:             $Rules = Rules \cup \{Rule\}$
9:             if $|b| > 1$ then
10:                $Rules = Rules \cup$ APRIORI-CONFRULES($f, b, conf_{min}$)
11:    return $Rules$
```

to the monotonicity of support. For each candidate, we compute its k subsets of size k-1 by removing one of its items respectively and check if they have been frequent in the previous iteration by checking if they are members of the set of frequent (k-1)-itemsets F_{k-1}. If one of the subsets of an itemset is infrequent, this itemset cannot be frequent due to the anti-monotonicity of support and may be removed from the candidate set C_k.

In the third and last step, the support of the remaining candidates C_k has to be calculated on the data by counting the number of transactions that contain a respective candidate. To this end, a fast method to determine if a candidate is a subset of a transaction is needed. Without going into details, Apriori employs an hash-tree based structure that returns the candidates that are contained in each transaction in a single pass over the data. The resulting information is aggregated to the support of each candidate. Afterwards, we obtain the frequent itemsets F_k by removing the infrequent ones from the candidate set C_k. The frequent itemsets and their support are stored in a efficient lookup data structure (for example a hash table) for two purposes. On the one hand, it speeds up the membership test for each subset in the previous step. On the other hand, it allows a fast confidence calculation for candidate rules that are considered in the second phase of the Apriori algorithm as we will see in the following paragraphs. At last, we obtain all frequent itemsets by unifying the results of this iteration and of previous ones ($F = F_1 \cup F_2 \cdots$).

The second phase of the Apriori algorithm APRIORI-CONFRULES (see Algorithm 4) searches for all confident association rules. To this end, Apriori selects consecutively one of the frequent itemsets that were discovered in the previous phase and uses it to construct association rules. In contrast to the AIS algorithm, Apriori allows rules with one or more items in its consequent.

The basic idea is to partition the items of a selected frequent itemset F into two itemsets and use them respectively as the antecedent and the consequent of an association rule. In this way, we obtain rules of the form

$$A \implies F \setminus A, \tag{2.43}$$

where $A \subset F$. Since there are $2^k - 2$ possible subsets (omitting F and $\emptyset$) of an k-itemset F, we obtain an equally large number of candidate rules whose confidence scores have to be calculated. Obviously, many of these rules will turn out to be inconfident and the calculation effort would be wasted. To avoid generating inconfident rules, Apriori improves this naïve approach by exploiting the anti-monotonicity of confidence: if a rule

$$X \implies Y \tag{2.44}$$

is inconfident, all rules

$$X \setminus Z \implies Y \cap Z, \tag{2.45}$$

where $Z \subset X$ cannot be confident either.

According to this observation, Apriori-ConfRules generates the subsets A of the frequent itemset F in a recursive depth-first search. This search starts with a set of candidate rules that have a single item in its consequent. In each iteration, the confidence of the candidate rules has to be calculated. Since the frequent itemsets and their support were stored in Apriori's first phase, the support counts that are needed to calculate the confidence may be retrieved efficiently. If a candidate rule obtained in this way is found to be confident, Apriori stores it, generates new rules by moving a single item from its body to its head and adds these rules to the set of candidate rules. Otherwise, the inconfident rule is discarded and may not be used as the seed of further candidate rules due to the anti-monotonicity of confident rules.

2.3.3.5 CHARM

In the previous section, we described how the Apriori algorithm solves the problem of the market basket analysis by an exhaustive search for frequent itemsets and by the generation of all confident association rules based on these itemsets. Depending (mainly) on the density of the transaction data and the chosen minimum support and confidence thresholds, the set of frequent and confident association rules may become very large and may obviously contain many redundant and uninteresting rules. Clearly, this bulk of association rules may be hard to handle. To remedy this problem, the set of association rules should only consist of non-redundant rules. For this purpose, an alternative approach for association rule mining was proposed in [ZH02]. The basic idea of this approach is to mine only a specific kind of frequent itemsets: closed frequent itemsets (which we will define later). The set of closed frequent itemsets is much smaller and easier to handle than the set of all frequent itemsets, it may be used to generate all non-redundant association rules [ZH02].

Since the determination of the set of closed frequent itemsets is the most computationally intensive step in this process, the bottom-up search employed by the Apriori algorithm or similar algorithms is not feasible in this context. Hence, CHARM was introduced in [ZH02]. It is an effective algorithm for the enumeration of closed frequent itemsets that features both a good time and space performance. For this purpose, CHARM employs several innovative ideas which include a novel tree-like search space that enables the simultaneous exploration of the itemset and transaction space, a hybrid search that may skip levels and whole branches in the tree structure, and a hash-based closedness checking. In this section, we will sketchily explain the CHARM algorithm, concentrating on the ideas behind its hybrid search and employed data structure. For further details, we refer to [ZH02] which provides a survey of this specific type of frequent itemset discovery and describes the theoretical background and implementation details of the algorithm.

As mentioned above, CHARM explores the itemset space and the transaction space simultaneously. Thus, we define the properties of these two spaces and of closed frequent itemsets first. Each transaction

$$t_{tid} \in T = \left\{t_1, \cdots, t_{|T|}\right\} \tag{2.46}$$

may be identified by its unique transaction identifier

$$tid \in TID = \{1, \cdots, |T|\}. \tag{2.47}$$

Comparable to itemsets, a set of transaction identifiers $Y \subseteq TID$ is called a *tidset*. Between the itemset space and the tidset space exist two mappings. For an itemset X, the mapping $t(X)$ is the set of all transactions (or transaction identifiers) which contain the itemset X as a subset:

$$t : I \rightarrow TID, t(X) = \{tid \in TID | X \subseteq t_{tid}\}. \tag{2.48}$$

Please note that the size of a tidset $|t(X)|$ is equal to the absolute support of the itemset X.

For a tidset $i(Y)$, the mapping $i(Y)$ is the itemset that is contained in all the transactions whose identifiers are in the tidset Y.

$$i : TID \rightarrow I, i(Y) = \{X \in I | \forall tid \in Y.X \subseteq t_{tid}\}. \tag{2.49}$$

Algorithm 6 CHARM - Closed Frequent Itemset Generation [ZH02]

```
 1: procedure CHARM($I$, $sup_{min}$)
 2:     $FrequentITPairs = \text{FREQUENTITPAIRS}_{\text{DB}}(I, T, sup_{min})$
 3:     return CHARM-EXTEND($FrequentITPairs$, $sup_{min}$)

 1: procedure CHARM-EXTEND($Candidates$, $sup_{min}$)
 2:     $ClosedITPairs = \emptyset$
 3:     for all $X_i \times t(X_i) \in Candidates$ do
 4:         $Candidates_{new} = \emptyset$
 5:         for all $X_j \times t(X_j) \in Candidates, X_i < X_j$ do
 6:             $X = X_i \cup X_j$
 7:             CHARM-PROPERTY($X_i, X_j, X, Candidates_{new}, Candidates, sup_{min}$)
 8:         if $Candidates_{new} \neq \emptyset$ then CHARM-EXTEND($Candidates_{new}, sup_{min}$)
 9:         if ISNOTSUBSUMED($X, L$) then $ClosedITPairs = ClosedITPairs \cup \{X\}$
10:     return $ClosedITPairs$

 1: procedure CHARM-PROPERTY($X_i, X_j, X, Candidates_{new}, Candidates, sup_{min}$)
 2:     if $Sup(X) \geq sup_{min}$ then
 3:         if $t(X_i) = t(X_j)$ then
 4:             REMOVE($X_j \times t(X_j), Candidates$)
 5:             REPLACEALLOCCURRENCES($X_i, X, Candidates_{new}$)
 6:         else if $t(X_i) \subset t(X_j)$ then
 7:             REPLACEALLOCCURRENCES($X_i, X, Candidates_{new}$)
 8:         else if $t(X_i) \supset t(X_j)$ then
 9:             REMOVE($X_j \times t(X_j), Candidates$)
10:             $Candidates_{new} = Candidates_{new} \cup \{X\}$
11:         else if $t(X_i) \neq t(X_j)$ then
12:             $Candidates_{new} = Candidates_{new} \cup \{X \times t(X)\}$
```

Using $t(X)$ and $i(Y)$, we define the *closure* of an itemset X as

$$c_{it}(X) = i \circ t(X) = i(t(X)), \tag{2.50}$$

the composition of these two mappings. An itemset X is *closed* if it is equal to its closure, i.e., $X = c_{it}(X)$. Once we map an itemset to the tidset that contains it, and then map that tidset back to the set of items common to all tids in the tidset, we obtain a closed itemset. After the application of such a round-trip $(i \circ t)$, we cannot extend the obtained closed itemset, no matter how many additional round-trips we make. Since the support of an itemset X is equal to the support of its closure ($sup(X) = sup(c_{it}(X))$), it is sufficient to mine only the set closed frequent itemsets [ZH02]. Afterwards, all frequent itemsets may be re-constructed, if necessary, by calculating all subsets of each closed frequent itemset.

Together the itemset and tidset space form the search space of the CHARM algorithm. Its elements

$$X \times t(X), i(Y) \times Y \in I \times TID, \tag{2.51}$$

which are referred to as itemset-tidset pairs (abbreviated IT-pair), are the base data structure of the CHARM algorithm. Their advantage becomes obvious, when we regard the fundamental operation on IT-pairs that CHARM employs in its bottom-up search process: the union of two IT-pairs X_i and X_j. For this purpose, we calculate the union of their itemsets and the intersection of their tidsets:

$$X_i \times t(X_i) \cup X_j \times t(X_j) = \left(X_i \cup X_j\right) \times \left(t(X_i) \cap t(X_j)\right) \tag{2.52}$$

In this way, we obtain a new IT-pair whose support has been calculated without a database lookup, as the intersection of tidsets may be computed directly.

Now, we have all tools at hand to describe the CHARM algorithm. As mentioned above, CHARM explores both the itemset and tidset space and enumerates only the closed frequent itemsets avoiding all subsets of a closed itemset. For this purpose, CHARM employs a novel search method that intelligently skips several levels and branches in the tree-like search space. Additionally, CHARM prunes this search space by removing itemsets and their outgoing branches that are either infrequent or non-closed. CHARM explores the search space implicitly by a single fundamental operation on two IT-pairs: the union of their two itemsets and the intersection of their tidsets. Dependent on the result of this operation, CHARM decides how to update the search space (and the current results) and how to continue its depth-first exploration. As the algorithm employs a depth-first search for closed itemsets, the closure of a non-closed itemset will be added to the result set before the itemset in question. Hence, it suffices to check if a candidate IT-pair is subsumed by another one in the result set by comparing their tidsets. If an identical tidset is found, the candidate IT-pair cannot be closed as its closure is already in the results. To reduce the number of comparison costs, CHARM stores the tidset in a hash table according to the sum of their transaction identifiers.

The pseudocode of the CHARM algorithm is outlined in Algorithm 6. Its search starts with an initial set of all 1-frequent IT-pairs that is successively extended or pruned. Afterwards, the algorithm calls two procedures recursively: Charm-Extend and Charm-Property. Charm-Extend organises the traversal of the search space. For this purpose, it picks one IT-pair $X_i \times t(X_i)$ and combines it with any IT-pair $X_j \times t(X_j)$, if $X_i < X_j$ is valid according to a total ordering of the itemsets, to obtain a new candidate IT-pair $X = (X_i \cup X_j) \times (t(X_i) \cap t(X_j))$. For each obtained IT-pair, Charm-Property is called to check its properties. If the obtained candidate IT-pair is frequent ($|t(X_i) \cap t(X_j)| \geq sup_{min}$), the tidsets of the two parent IT-pairs are compared. There are four possible constellations which are dealt with as follows:

1. $t(X_i) = t(X_j)$: as the itemsets X_i, X_j and X have the same closure, all occurrences of X_i are replaced by X, and X_j does not have to be considered further.

2. $t(X_i) \subset t(X_j)$: all occurrences of X_i are replaced by X as they have the same closure. However, X_j must be retained as it generates a different closure.

3. $t(X_i) \supset t(X_j)$: X_j is removed from further considerations as it has the same closure as X. Consequently, X is added to the itemset candidates.

4. $t(X_i) \neq t(X_j)$: since both itemsets generate different closures, none may be removed or replaced. X is added to the itemset candidates.

In summary, three operations are involved.

- $X_j \times t(X_j)$ is removed from the current set of candidate IT-pairs *Candidates*.

- X replaces X_i and all occurrences of X_i in any IT-pair in the set of new candidate IT-pairs $\mathit{Candidates}_{new}$

- $X \times t(X)$ is added to the set of new candidate IT-pairs $\mathit{Candidates}_{new}$

After each combination of $X_i \times t(X_i)$ with another IT-pair has been evaluated, CHARM-EXTEND explores the new obtained candidate IT-pairs $\mathit{Candidates}_{new}$ in a depth-first manner, if any new IT-pairs have been found by CHARM-PROPERTY. At last, the current IT-pair $X_i \times t(X_i)$ is added to the result set if it is not subsumed by any IT-pair in the result set. Then, the whole procedure is repeated with the next IT-pair in the current set of candidate IT-pairs *Candidates*, until no further itemsets may be combined. In the end, the set of closed IT-pairs *ClosedITPairs* is returned.

2.4 Global Models

In the previous section, we dealt with the discovery of local patterns for classification. As these allow only local predictions, we will concern ourselves in this section with global models for classification which are referred to as classifiers. To this end, we explain the properties of classifiers first. Afterwards, we show how a global model may be learned with the aid of the separate-and-conquer strategy (abbreviated SeCo strategy) which generates local patterns

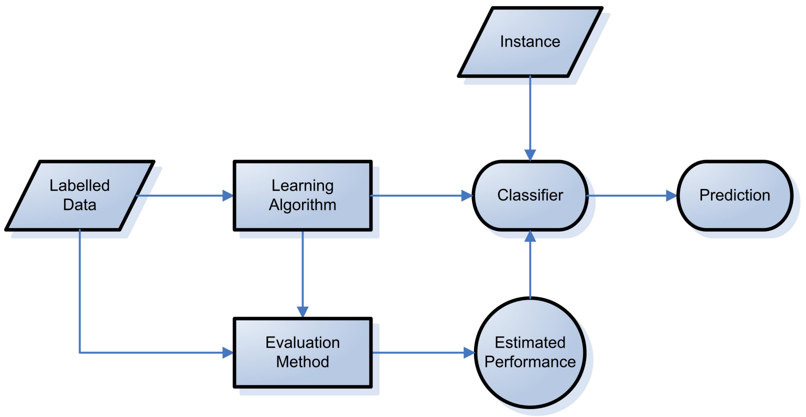


Figure 2.4.: Classification learning: illustration of the life cycle of a classification learning algorithm.

and employs them to obtain a global model [Für99]. Next, we deal with ensemble learning and class binarisations that both employ several global models for prediction, treating the global models as a single one. Subsequently, we explain how the performance of a global model may be evaluated. At last, we present Ripper, a very efficient separate-and-conquer rule learning algorithm that we employ in our experiments.

2.4.1 Classifiers

The learning algorithm generates a global model on the information contained in the labelled classification data. The classifier, as such a global model is called, may be used to predict the class labels of unlabelled instances. To obtain an approximation of the quality of this prediction, an evaluation method rates the performance or quality of the classifier heuristically. First, we will concentrate on classifiers and their properties, while the evaluation of this type of global models will be discussed later in this section.

As mentioned before, classification learning utilises the observed labelled data, commonly referred to as training data, to generate a predictive global model which classifies unlabelled instances. A *classifier*, as such a model is called, tries to predict the class value of an (unlabelled) instance using only the values of the attributes A_1 to $A_{|S|-1}$. Thus, a classifier can be seen as a function of the following format:

$$c : dom\left(A_1\right) \times \cdots \times dom\left(A_{|S|-1}\right) \rightarrow L,\ d \mapsto c(d) \tag{2.53}$$

The generation of a classifier is called its *training or learning phase*. Accordingly, the application of a classifier is referred to as the *prediction phase* or, in the case of evaluation, as the *test phase*. The complete classification learning scheme is depicted in Figure 2.4.

Decision lists are one category of classifiers that we will encounter several times in this work. They consist of a list of rules that are sorted meaningfully (e.g. by their creation order or individual quality). In the prediction phase, these rules are evaluated according to this ordering. To this end, an instance is classified by the first rule that covers the instance. Usually, the last rule in the list is a default rule that covers all (yet uncovered) instances and predicts a determined default class value. Such a decision list which consists of $|R|-1$ rules and one default rule has basically the following form:

$$\begin{aligned} & r_1 \\ & r_2 \\ & \cdots \\ & r_{|R|-1} \\ & true \implies l_{default} \end{aligned} \tag{2.54}$$

2.4.2 Separate-and-Conquer: Learning a Global Model

In this section, we describe the learning of a global model exemplified by the separate-and-conquer strategy [Für99]. Many rule learning algorithms are based on this generic separate-and-conquer strategy (abbreviated SeCo) that essentially employs local pattern discovery to obtain a global model. The basic idea of this approach is to iteratively generate local patterns and to utilise them to construct a global model in form of a decision list [Für04]. For this purpose, SeCo employs a local pattern discovery algorithm to extract a single pattern from the training data. As it explains or covers only a part of the training instances, additional patterns are needed for a complete global model. To obtain a different pattern, the instances that are covered by this rule are removed or separated from the training instances. These two steps - learning a single pattern and removing the instances that it covers - are repeated until all remaining instances are conquered by learning additional covering rules. For a deeper insight into the SeCo strategy and the different options for each of its data mining components, we refer to [Für99].

The generic SeCo algorithm SeparateAndConquer which is an implementation of the SeCo strategy is shown in Algorithm 7. At the beginning of this algorithm, the model *DecisionList* is initialised as an empty decision list. Then, the SeCo or covering loop is repeated as long as there are still positive instances that are not covered by one or more rules. In the first part of this loop, Algorithm 2 FindBestPattern that we described previously in Section 2.3.1

Algorithm 7 SeparateAndConquer [Für99]

```
1:  procedure SeparateAndConquer(Instances)
2:      DecisionList = EmptyList()
3:      while Positives(Instances) do
4:          Rule = FindBestPattern
5:          if RuleStoppingCriterion(DecisionList, Rule, Instances) then
6:              exit while
7:          RemoveCoveredInstances(Instances, Rule)
8:          DecisionList = Append(DecisionList, Rule)
9:      PostProcess(DecisionList)
10:     return DecisionList
```

extracts from the yet uncovered instances a single learned rule *Rule*. Next, RuleStoppingCriterion checks if the found rule is not beneficial for the resulting decision list. If so, the covering loop is exited. Otherwise, the processing of the covering loop is continued. In the second part, the extracted rule *Rule* is appended at the end of *DecisionList*, and the instances that are covered by this rule are removed from the training data *Instances*. After the termination of the covering loop, the obtained *DecisionList* may be modified by a post-processing step provided by the procedure PostProcess which usually includes data mining methods for the overfitting avoidance (e.g. post-pruning) and the addition of a default rule.

Similarly to the algorithm FindBestPattern, SeparateAndConquer is inherently designed to handle binary learning problems, but, as we will see later in this section, there exist methods to apply it to multi-class learning problems (see Section 2.4.3). Its step may be related to the main components of data mining algorithm. In addition to the components of FindBestPattern, SeparateAndConquer features two procedures for the overfitting avoidance: the pre-pruning procedure RuleStoppingCriterion and the post-pruning procedure PostProcess.

2.4.3 Ensembles

In the previous sections, we concerned ourselves with local patterns and their utilisation to obtain global models. In this section, we deal with two data mining categories, ensemble learning [Die00] and class binarisations [Für02], that employ a group of global models for classification which we refer to as an ensemble. Apparently, the predictions of the global models of an ensemble have to be decoded into a single prediction using either a meta model (e.g. the model induced by stacking) or a decoding method (e.g. voting methods) as we will see later in this section.

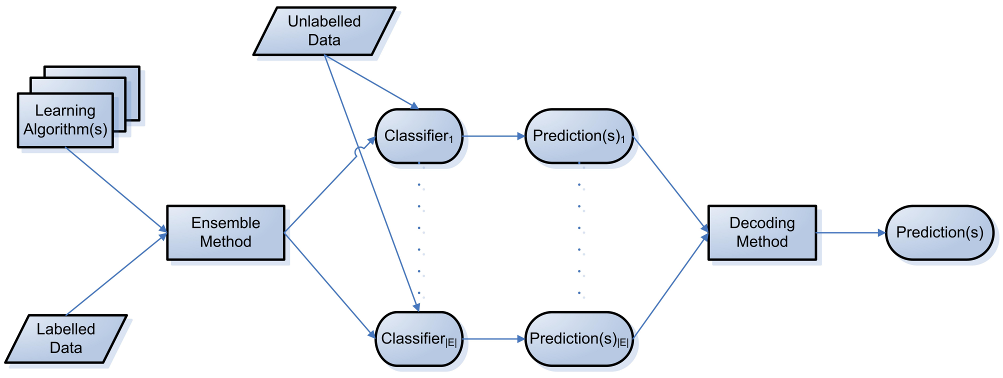


Figure 2.5.: Ensemble learning: illustration of the life cycle of an ensemble learning algorithm.

The remaining section is organised as follows. First, we introduce ensemble learning, describing its main ideas, functionalities and properties, and the ensemble methods bagging and stacking. Afterwards, we consider the concept of class binarisation, introducing its main idea, its field of application, and three representatives that were involved in our experiments. At last, we show how the predictions of a group of classifiers may be decoded by voting methods.

2.4.3.1 Ensemble Learning

Ensemble learning [Die00], which is illustrated in Figure 2.5, employs several individual base classifiers (referred to as an ensemble of classifiers) and combines them in order to obtain a superior meta classifier that outperforms the original ones. An important part of this approach is to guarantee the diversity of the *ensemble*

$$E = \left\{c_1, \cdots c_{|E|}\right\}. \tag{2.55}$$

Different classifiers may be obtained either by the modification of the data (e.g. by sampling of the instances or selecting randomly a subset of the features) or by the exploitation of the learning algorithm characteristics (e.g. by using algorithms with random components or by the application of multiple algorithms with different properties). Evidently, the combination of multiple classifiers may lead to a more expressive classification model than a single one. Analogously, the multiple reuse or variation of the training data may reduce the effect of noise or inconsistencies. Next, we will introduce two ensemble learning methods, bagging and stacking, which are important representatives in our experiments.

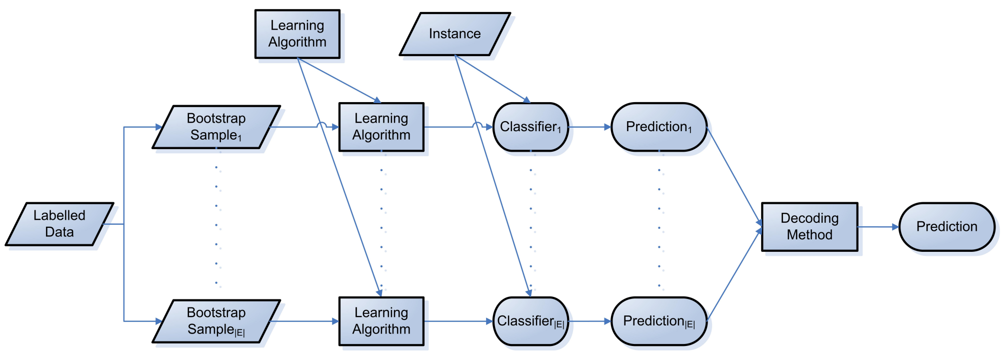


Figure 2.6.: Bagging: illustration of the life cycle of a bagging ensemble.

Bagging

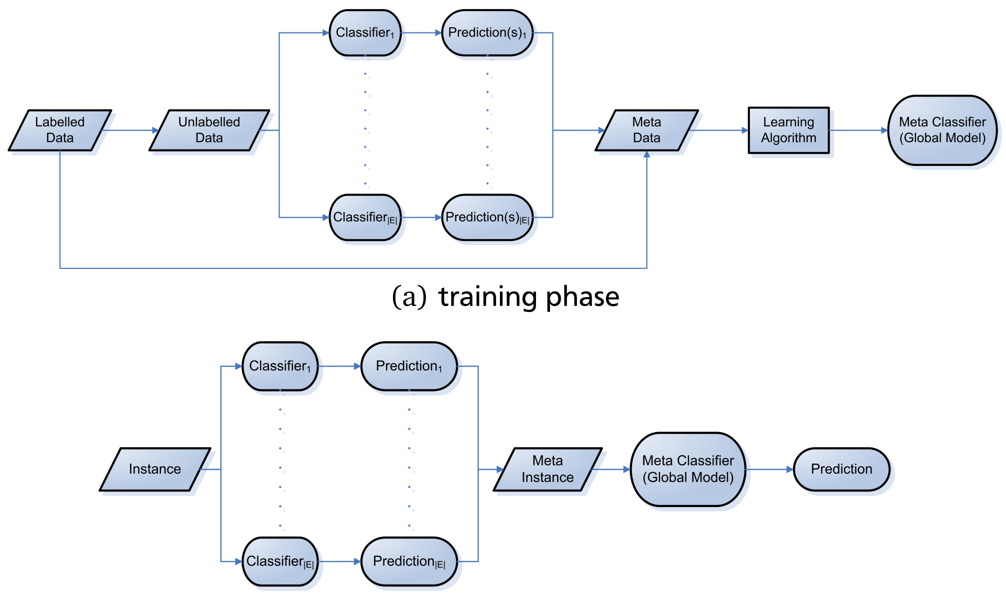
The bagging (the acronym of **b**ootstrap **agg**regat**ing**) algorihm [Bre96] builds an ensemble of different classifiers by training each of these classifier on a bootstrap sample of the original data set. A bootstrap sample is generated by drawing randomly with replacement t instances of the training set. Consequently, drawn instances are not removed from training set and may be redrawn more than once. Repeating this technique $|E|$-times, we get the bootstrap samples $B_1, B_2, \cdots, B_{|E|}$. For each bootstrap sample B_i, we generate a classifier c_i. The predictions of these classifiers are aggregated by voting methods which will be introduced later in this section.

Whether the bagging approach yields an improved accuracy or not, depends on the stability of the employed base classifiers [Bre96]. On the one hand, an unstable classifier which is (very) sensitive to small changes in the training data generates an ensemble of distinctly different classifiers, leading to an improved accuracy. On the other hand, a stable classifier generates an ensemble of slightly differing or even equivalent classifiers, hence the overall performance may be degraded slightly. Please note that in the case of interpretable data mining methods (e.g. rule learning or decision tree learning) the improved performance is bought by loosing the interpretability of the model.

Nevertheless, the bagging approach may be implemented easily. Only an additional outer loop in the training phase that selects the bootstrap sample and assigns it to the learning algorithm, and a voting method for the prediction phase have to be added. The complete approach is depicted in Figure 2.6.

Stacking

Stacking is an ensemble learning method [Wol92], which generates a global meta classifier based on the predictions of an ensemble of $|E|$ base level classifiers. The key idea of stacking is to build a meta data set based on the predictions of these base level classifier. The meta data set uses the base level



(a) training phase

(b) prediction phase

Figure 2.7.: Stacking: illustration of the training (a) and prediction phase (b) of stacking, which uses the predictions of the base classifiers on the original data sets as features of the meta level data set.

classifiers as meta attributes (c_1 to $c_{|E|}$) and their predictions (e.g. $c_i(d_j)$) respectively as their (attribute) values. In this way, the original $|D|$ instances are transformed into $|D|$ meta instances. Each meta instance consists of the predictions it receives from each classifier. For training the meta classifiers, the meta instances are labelled with the class labels of the corresponding base level instances (e.g. l_{d_i}), as shown in Figure 2.8. Thus, the resulting meta level model is based only on the predictions of the base level classifiers. In the testing phase, the test instance is transformed into a meta instance by determining the prediction of each base classifier. Afterwards, the original instance is classified by the prediction of the meta level classifier, using the meta instance as its input. Thus, the meta level model acts as a global model and no decoding methods (e.g. voting methods) are needed. The complete approach is illustrated in Figure 2.7

Prior work has shown that the simple version of stacking described above does not perform as well as other ensemble techniques, and several improvements have been proposed. Most notably, it has been shown that instead of using the class label as an attribute at the meta level, it is beneficial to augment the meta data set with the confidences of the base level classifiers into

Attributes			Class
$a_{d_1,1}$	$\cdots$	$a_{d_1,\|S\|-1}$	l_{d_1}
$a_{d_2,1}$	$\cdots$	$a_{d_2,\|S\|-1}$	l_{d_2}
$\cdots$	$\cdots$	$\cdots$	$\cdots$
$a_{d_{\|D\|},1}$	$\cdots$	$a_{d_{\|D\|},\|S\|-1}$	$l_{d_{\|D\|}}$

(a) original data set

c_1	c_2	$\cdots$	$c_{\|E\|}$	Class
$c_1(d_1)$	$c_2(d_1)$	$\cdots$	$c_{\|E\|}(d_1)$	l_{d_1}
$c_1(d_2)$	$c_2(d_2)$	$\cdots$	$c_{\|E\|}(d_2)$	l_{d_2}
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
$c_1(d_{\|D\|})$	$c_2(d_{\|D\|})$	$\cdots$	$c_{\|E\|}(d_{\|D\|})$	$l_{d_{\|D\|}}$

(b) training set for stacking

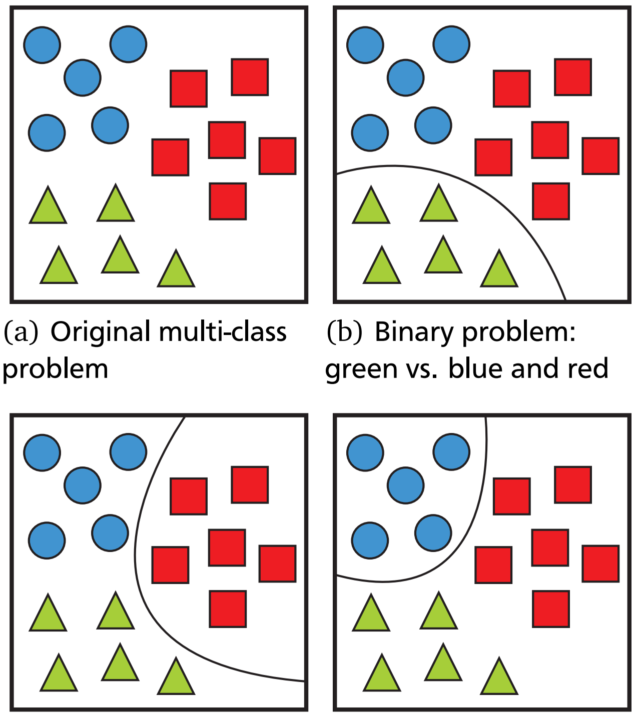
Figure 2.8.: Stacking: illustration of the meta data generation of stacking, which uses the predictions of the base classifiers on the original data sets (a) as features of the meta level data set (b).

their predictions [TW99]. Subsequently, it was shown that it may be even better to use the entire predicted class probability distribution [See02].

2.4.3.2 Class Binarisations

As mentioned before, real-world learning problems are often multi-class problems. However, many classification algorithms are inherently binary, as they are designed to discriminate only between two class labels. A well known example of such a binary classifier are separate-and-conquer rule learning algorithms which we encountered already in Section 2.4.2. Class binarisation techniques [Für02] solve this problem by transforming the multi-class learning problem into several binary learning problems that can be handled by binary classifiers. This approach is widely used as binary classifiers may be directly applied and no further modifications to the learning algorithm are necessary (e.g. generalising the learning algorithm to multi-class case).

Class binarisations solve multi-class learning problems by decomposing the original multi-class problem into several binary problems. For each of these binary problems, a base classifier is trained. On the basis of the predictions of these base classifiers, a prediction for the original multi-class problem is decoded. For the decoding of the predictions, several methods may be used as we will see later in this section. In the next sections, we introduce three class binarisation methods: the unordered, ordered and pairwise class binarisation.



(a) Original multi-class problem

(b) Binary problem: green vs. blue and red

(c) Binary problem: red vs. blue and green

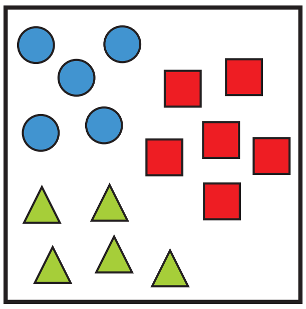
(d) Binary problem: blue vs. green and red

Figure 2.9.: Unordered class binarisation: the original multi-class problem (a) that consists of the blue, green and red classes is decomposed into the three one-against-all binary problems (b),(c), and (d).

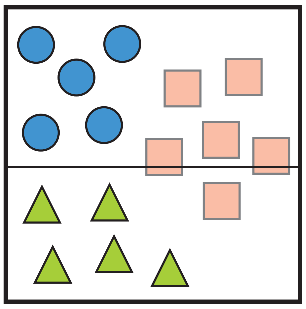
Unordered and Ordered Class Binarisation

The unordered class binarisation [CB91] is a popular class binarisation method (see Figure 2.9). Its basic idea is to generate a binary learning problem for each of the class label where one class is discriminated from all remaining class labels. Hence, this class binarisation is also known as a one-against-all class binarisation. Nevertheless, we will refer this approach as an unordered class binarisation as this term is more common in the context of rule learning.

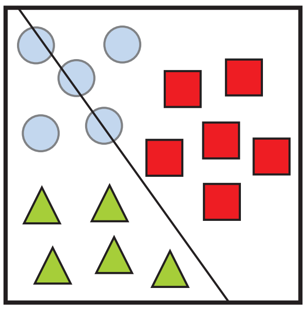
As already mentioned, the unordered class binarisation transforms a $|L|$-class learning problem into $|L|$ binary learning problems (one for each class in the data set). Each of these binary learning problems, which are constructed by modifying (a copy of) the original multi-class data set, aims to discriminate one class from all other classes. For the class l_i, all instances of this class are used as positive instances, whereas the instances of all other classes $l_j \in L \setminus l_i$ are used as the negative ones.



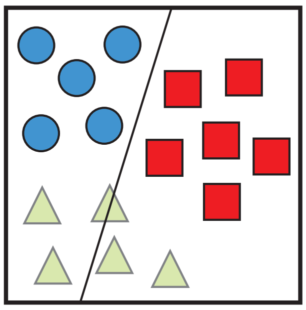
(a) Original multi-class problem



(b) Binary problem: blue vs. green



(c) Binary problem: green vs. red



(d) Binary problem: blue vs. red

Figure 2.10.: Pairwise class binarisation: the original multi-class problem (a) that consists of the blue, green and red classes is decomposed into the three pairwise binary problems (b),(c), and (d).

If we propose an ordering of the classes (e.g. in ascending order according to the frequency of the classes in the data), this approach may be extended to an ordered class binarisation. Here we also try to discriminate one class from other classes, but this time only the instances of the classes that are ranked higher are used as negative ones. Without loss of generality, we assume that class labels were distributed according to this ordering. For the class l_i, the instances of the class l_i are used as the positive instances and the instances of the classes l_{i+1} to $l_{|L|}$ as the negative ones. In this way, we obtain $|L|-1$ binary learning problems. Usually, the highest ranking class $l_{|L|}$ experiences a special treatment (e.g. by a default rule).

Pairwise Class Binarisation

The pairwise or round-robin class binarisation [Für02] is named after the round-robin tournament mode which is commonly used in tournaments of sports and games (e.g. in sports leagues or chess). In this tournament mode, each participant has to compete with each other participant. Obviously, this basic idea may easily be translated to class binarisations. To this end, pairwise class binarisation transforms a $|L|$-class learning problem into $|L|\cdot(|L|+1)/2$ pairwise binary learning problems and trains one classifier for each of these learning problems. Each pairwise classifier is trained only with the instances of the classes l_i and l_j, ignoring the examples of all other classes l_k $(k \neq i, j)$ completely in this training process.

The pairwise learning problems are usually much simpler than the original multi-class learning problems and the (un-)ordered learning problems as two classes may be more easily separated than multiple ones (see Figure 2.10). Consequently, simpler and significantly more accurate classifiers may be found in this way. Even though this approach has to train a quadratic number of classifiers, the total training time of a pairwise class binarisation is comparable to (or even less than) the total training time of an (un-)ordered class binarisation due to the reduced training set size of each pairwise problem [Für02]. However, the quadratic number of classifiers may be a bottleneck for the classification time, as a naïve decoding employs all classifiers for prediction.

2.4.3.3 Voting Methods

Usually, the predictions of an ensemble have to be decoded into a single prediction. For this purpose, the previously introduced ensemble learning and class binarisations methods (with the exception of stacking) employ voting methods. Common ground of all voting methods is, as the name suggests, the interpretation of the individual predictions as votes for the predicted class. The various voting methods differ in the weights they assign to the vote of a classifier of the ensemble.

For a classifier ensemble, the classification by voting works essentially as follows:

$$c_E(d) = \arg\max_{l \in L} \sum_{c \in E} vote(c, l, d) \tag{2.56}$$

where the weight of the vote $vote(c, l, d)$ depends on the chosen voting method. Commonly, two voting methods are used: (*simple*) *voting*

$$vote_v(c, l, d) = \begin{cases} 1, & \textit{if } c(d) = l, \\ 0, & \textit{otherwise}. \end{cases} \tag{2.57}$$

that simply counts the votes for each predicted class (weight=1) and *weighted voting*

$$vote_{wv}(c, l, d) = \begin{cases} h(c, D), & \textit{if } c(d) = l, \\ 0, & \textit{otherwise}. \end{cases} \tag{2.58}$$

that uses a measure of confidence $h(c, D)$ of the voting classifiers as the weights of these votes. Usually, the quality of a classifier or its confidence in its prediction are used for this purpose. While the former may be estimated for every classifier, the latter may only be used if it is provided by the classifier. In the next section, we will show how the quality of a classifier may be estimated.

2.4.4 Evaluation

After the generation of a classifier, we usually want to evaluate its quality or performance using appropriate performance measures. In our work, we consider two categories of performance measures: the accuracy of the classifier (on previously unseen data instances) and the complexity of the classifier. While the complexity of a classifier normally may be measured exactly (for example, by counting its components), its accuracy, as we will see later, may only be estimated by evaluation methods (e.g. cross-validation). Based on the evaluated performances of classifiers, statistical tests may be used to compare the learning algorithms that generated those classifiers.

Below, we describe the statistical values that are needed to evaluate the accuracy and complexity of a single global model and how the evaluation method cross-validation may be employed to obtain an estimation of the true accuracy. Afterwards, we show how multiple learning algorithms may be compared by statistical tests.

2.4.4.1 Evaluation of Global Models

Before a classifier is used for prediction, the accuracy of the classifier on previously unseen data has to be evaluated. Of course, we cannot determine this value exactly, since we do not know the class values of these unseen data instances in general. For this reason, we calculate the accuracy of the classifier on data instances whose class values are known previously and estimate the true accuracy of the classifier using this accuracy value.

The *accuracy* of a classifier is a measure for the quality of its prediction. A classifier c predicts correctly, if $c(d) = l_d$ is valid. Otherwise ($c(d) \neq l_d$), it makes an error. The accuracy of a classifier is equal to the fraction of instances of a D that are correctly classified by the classifier c:

$$h_{acc}(c, D) = \frac{|\{d \in D | c(d) = l_d\}|}{|D|} \tag{2.59}$$

The *error rate* of a classifier c on a data set D is complementary to its accuracy:

$$h_{err}(c, D) = 1 - h_{acc}(c, D). \tag{2.60}$$

Please note that the accuracy of a rule and the accuracy of a classifier differ.

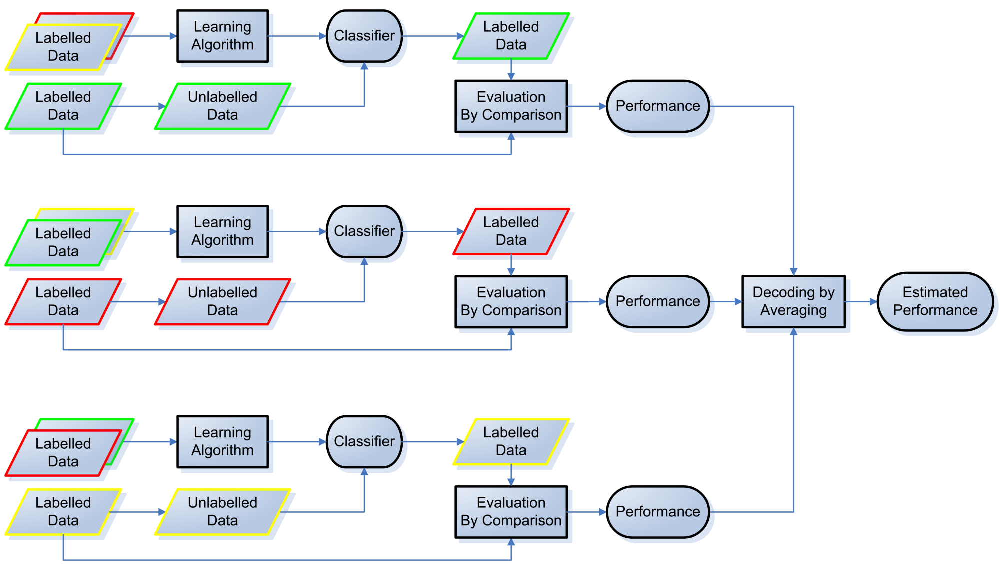


Figure 2.11.: Cross-validation: exemplary illustration of the scheme of a 3-fold cross-validation. The data set is split into three separate folds (coloured green, red or yellow, respectively). Each pair of folds is used one time for training a classifier while the remaining third fold is used to evaluate the resulting classifier. Afterwards the obtained performance scores are averaged.

Cross-Validation

If we use the same data for the training and the evaluation of a classifier, the estimated accuracy tends to be overly optimistic. Especially if the classifier overfits the data. Several techniques solve this problem by partitioning the data into two or more sets of data instances. One data set is used to generate or train the classifier. On the other data set(s), we apply the classifier and calculate its accuracy. One representative of this approach is the cross-validation which we use in our work to estimate the performance of the considered classifiers. Since labelled data instances are usually scarce, all instances should be used for the training. The obtained classifier may be then evaluated using one of the above-mentioned techniques.

Cross-validation partitions the data into f subsets or folds. For each fold, we train a classifier on the union of all other folds, and estimate the performance of the classifier on the selected fold. Afterwards, the estimated performance is calculated as the average performance of these classifiers.

For a f-fold cross-validation, we partition our data set into f subsets (folds) of equal size:

$$D = D_1 \cup \cdots \cup D_f \tag{2.61}$$

Usually the data set cannot be split into f subsets of exactly the same size, hence the data is split as uniformly as possible, e.g. splitting a data set consisting of 32 data instances into 3 subsets would result in two subsets of 11 data instances and one subset of 10 data instances. If the original distribution of class labels is preserved approximately in all subsets, the cross-validation is called stratified.

As we use only one of these subsets for testing and the rest of the data set for the training, we introduce the complement of each subset:

$$\overline{D_i} = D \setminus D_i, \tag{2.62}$$

where $i \in \{1, \cdots, f\}$. So, the complementary subsets $\overline{D_i}$ are used as the training sets and the subsets D_i as the test sets, respectively.

For each complementary subset $\overline{D_i}$, we train a classifier using only the information contained in $\overline{D_i}$:

$$c_{\overline{D_i}} \tag{2.63}$$

Hereafter we evaluate this classifier on the appropriate subset D_i using the heuristic evaluation function h_{acc}:

$$h_{acc}(c_{\overline{D_i}}, D_i). \tag{2.64}$$

After the computation of the heuristic value of each classifier, we calculate their average value:

$$\bar{h}_{acc} = \frac{1}{f}\sum_{i=1}^{f} h_{acc}(c_{\overline{D_i}}, D_i) \tag{2.65}$$

2.4.4.2 Comparison of Data Mining Algorithms

As mentioned before, there is an abundance of data mining learning algorithms, and this number increases continuously. Thus, it is important to know which of the available learning algorithms is the superior one for the given data mining task. To compare several learning algorithms, statistical tests may be employed. The base idea of these tests is to decide if one or more compared algorithms are significantly better than the others. Usually, this approach is used in the development of new algorithms, too.

For the evaluation of the learning algorithms in our experiments, we employ statistical tests whose null hypothesis is that the performances of the compared algorithms do not differ significantly. Consequently, the compared algorithms perform significantly different if this null hypothesis is rejected. Otherwise, no significant differences in performance could be detected. Depending on the number of compared learning algorithms, we employ two different test configurations as proposed in [Dem06]: the Wilcoxon signed ranks test for the comparison of two algorithms and the Friedman test with a post hoc Nemenyi test for the comparison of multiple (three or more) algorithms. For both configurations, the classifiers considered for comparison are applied to multiple data sets. Subsequently, their performance may be estimated by an aforementioned evaluation method. We denote the evaluated performance score of c_i on D_j as $score_{i,j}$. Afterwards these scores are used to obtain performance rankings whose members are numbered serially. Obviously, a performance score may occur multiply in a ranking. In this case, an average score is calculated:

$$rank_{avg} = \frac{rank_x + rank_y}{2} \tag{2.66}$$

for all equally performing ranked members from $rank_x$ to $rank_y$. This average score is then used to substitute the ranks from $rank_x$ to $rank_y$.

Comparison of two Data Mining Algorithms

For the comparison of two classifiers, we employ the non-parametric Wilcoxon signed ranks test (Wilcoxon, 1945). It computes the differences in performance of two classifiers for each data set, assigns ranks to these differences according to their absolute values, and compares the sums of assigned ranks of the positive and the negative differences. If the minimum of these sums is below the critical value, the null-hypothesis, that the two classifiers are not significantly different, may be discarded.

The Wilcoxon signed ranks test works as follows. For each data set D_j, we calculate the difference between the performance scores of the two classifiers c_1 and c_2:

$$diff_j = score_{1,j} - score_{2,j} \tag{2.67}$$

Afterwards, we sort these differences in descending order and assign ranks to them according their absolute values (denoted by $rank(diff_j)$). The largest absolute difference receives the lowest rank of 1. In case of equal difference values, average ranks are determined (see above). Then, we compute the sum of positive differences where c_1 outperformed c_2 and the sum of the negative ones where c_2 performed better than c_1. The ranks of zero differences ($diff_i = 0$) are split evenly among these two sums. Accordingly, we obtain the following two sums:

$$ranks_+ = \sum_{diff_i > 0} rank(diff_i) + \frac{1}{2} \sum_{diff_i = 0} rank(diff_i) \tag{2.68}$$

$$ranks_- = \sum_{diff_i < 0} rank(diff_i) + \frac{1}{2} \sum_{diff_i = 0} rank(diff_i) \tag{2.69}$$

If the minimum

$$T = \min(ranks_+, ranks_-) \tag{2.70}$$

of these sums is below the critical value for the given number of data sets N, the null hypothesis may be rejected. If more than 25 data sets are involved, the statistic

$$z = \frac{\min(ranks_+, ranks_-) - \frac{1}{4}N(N+1)}{\sqrt{\frac{1}{24}N(N+2)(2N+1)}} \tag{2.71}$$

is distributed approximately normally.

Comparison of Multiple Data Mining Algorithms

For the evaluation of multiple learning algorithms in our experiments, we employ the Friedman [Fri37, Fri40] test with a post hoc Nemenyi-Test [Nem63]. The Friedman test is a non-parametric statistical test used for the comparison of multiple learning algorithms. The null-hypothesis of the Friedman test assumes that all learning algorithms are equivalent and so their average ranks should be equal. So, its main idea is to rank the learning algorithms for each data set separately and use the average ranks for comparisons.

In this ranking, the learning algorithms are sorted in a descending order, assigning the best performing learning algorithms the rank 1, the second best rank 2, and so on. We denote the rank of the j-th of k learning algorithms on the i-th of N data sets with r_i^j. If several learning algorithms perform equally well on a data set, an average rank is assigned to those learning algorithms as described above.

As already mentioned the Friedman tests uses the average rank (over all data sets)

$$\overline{rank} = \frac{1}{N} \sum_{i=1}^{N} r_i^j \tag{2.72}$$

of each learning algorithm for comparison. Using these average ranks, we calculate the Friedman statistics

$$\chi_F^2 = \frac{12N}{k(k+1)} \left[\sum_{j=1}^{k} \overline{rank}^2 - \frac{k(k+1)^2}{4} \right], \tag{2.73}$$

which is itself used to calculate an improved statistics

$$F_F = \frac{(N-1)\chi_F^2}{N(k-1) - \chi_F^2} \tag{2.74}$$

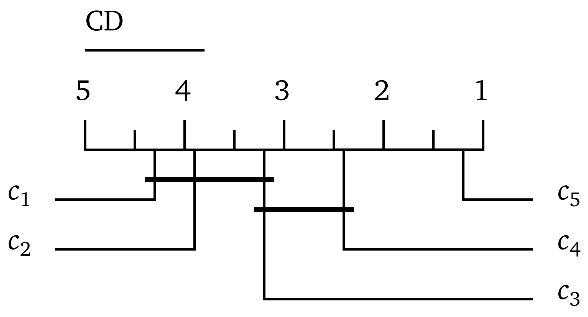


Figure 2.12.: Critical distance chart: an illustrative example.

proposed in [ID80]. Under the null-hypothesis F_F is distributed according to the F-Distribution with $k-1$ and $(k-1)(N-1)$ degrees of freedom.

If the null hypothesis is rejected, we apply a post hoc Nemenyi test. Its base idea is that two learning algorithms perform significantly different if their corresponding average ranks differ by at least the critical difference

$$CD = q_a \sqrt{\frac{k(k+1)}{6N}}. \tag{2.75}$$

The critical values q_a are computed by dividing the studentized range statistics by $\sqrt{2}$.

The results of the Nemenyi test can be depicted in a so called critical distance chart (abbreviated CD chart). In this visualisation one can easily recognise groups of learning algorithms that differ significantly or do not. The CD chart consists basically of three parts: a number line, charted learning algorithms and horizontal bars that connect learning algorithms that do not differ significantly. The number line is, quite unusually, in descending order, starting with the highest obtainable rank (the number of learning algorithms) and ending with rank 1. Sometimes it may be more appropriate to display only a section of the number line, e.g. cutting off the unnecessary outer parts (below or above the lowest or highest average rank, respectively). The learning algorithms are charted according to their average rank.

Let us illustrate such a CD chart with an example (see Figure 2.12). In this example, five classifiers c_1 to c_5 are charted according to their average performance. Three groups of learning algorithms whose performances do not differ significantly may be identified. The first group consists of the classifiers c_1, c_2 and c_3 which performed worst. The second group comprises classifiers c_3, which also belongs to the first group, and c_4. To the last and best performing group belongs only classifier c_5. It is clearly the best classifier in this example as it performed significantly better than the others.

2.4.5 Ripper

In our work, we used Ripper (an acronym of **r**epeated **i**ncremental **p**runing to **p**roduce **e**rror **r**eduction) as a base and benchmark learner for our experiments [Coh95]. Ripper is a very efficient and accurate rule learner which is designed to solve the problem that many rule learning systems scale very poorly on large and noisy data sets. Its base algorithm I-REP* is a modification of the incremental reduced error pruning approach (abbreviated I-REP) introduced in [FW94]. I-REP* is a separate-and-conquer rule learning algorithm that integrates pre-pruning and post-pruning into the learning process, combining the advantages of both worlds. On the one hand, I-REP* prunes each rule immediately after its generation, instead of pruning a completed rule set. On the other hand, I-REP* stops adding rules and completes its learning process when adding a new rule would increase the minimum description length of the current decision list above a certain threshold.

The I-REP* algorithm starts with a binary training data, trying to separate one class from the other. The first step of I-REP* splits the (uncovered) positive and negative instances of the training data into a growing and pruning set. In the second step, the growing data is used to grow a single rule. To this end, a modified version of FOIL [Qui90, QC93] is used to generate the rules in the growing phase. For this purpose, it considers conditions as follows: $A = a_i$ for nominal attributes and $A \leq a_i$ and $A \geq a_i$ for numerical attributes. For growing a rule, this modified FOIL selects from all available conditions the condition that maximises FOIL's information gain criterion

$$h_{FOIL}(r) = p\left(log_2\left(h_{prec}\left(r_{parent}\right)\right) - log_2\left(\frac{p}{p+n}\right)\right) \tag{2.76}$$

of the resulting rule (where r_{parent} is its parent rule) until the rule does not cover any negative instances or no further conditions may be added.

Afterwards, we prune this rule by deleting conditions until any further deletions would lead to a decreased accuracy on the pruning set. For pruning the resulting rule, I-REP* considers the removal of any final sequence of conditions and chooses the removal that maximises the rule-value metric

$$h_{RVM}(r) = \frac{p-n}{p+n}\left(\equiv \frac{p}{p+n} = h_{prec}(r)\right) \tag{2.77}$$

on the pruning data. Please note that this metric is equivalent to the precision metric for this purpose.

Next, the stopping condition of I-REP*, which is based on the minimum description length principle [Ris78], is checked. I-REP* employs the method described in [Qui95] to calculate the total description length of the current

Algorithm 8 I-REP* [Coh95]

```
 1: procedure I-REP*($Instances, SplitRatio, MDL_{Threshold}$)
 2:     $DecisionList$ = EmptyList()
 3:     while Positives($Instances$) $\neq \emptyset$ do
 4:         SplitInstances($SplitRatio, Instances, Instances_{Grow}, Instances_{Prune}$)
 5:         $Rule$ = GrowRule($Instances_{Grow}$)
 6:         $Rule$ = PruneRule($Rule, Instances_{Prune}$)
 7:         if $MDL$(Append($DecisionList, Rule$)) $-$ $MDL$($DecisionList$) $\geq$ $MDL_{Threshold}$ then
 8:             exit while
 9:         else
10:             $DecisionList$ = Append($DecisionList, Rule$)
11:             RemoveCoveredInstances($Instances, Rule$)
12:     Append($DecisionList, DefaultRule$)
13:     return $DecisionList$
```

rule set (including the last generated rule) and the uncovered examples and compares it to the smallest total description length obtained so far. If the new description length is more than d (default value is 64) bits larger than the smallest one, or when there are no more uncovered examples, I-REP* stops generating rules. Otherwise, we add the rule to the rule set, remove the covered instances from the training set (both the growing and pruning sets) and restart the process at the first step with the remaining instances which are again split in growing and pruning sets using a new division.

Ripper employs the I-REP* algorithm - at least twice - for the generation of rules (see Algorithm 9). First, I-REP* is used to generate an initial rule set. In the second step, this rule set is optimised using an optimisation approach which resembles the effects of conventional reduced error pruning. The functionality of this step will be explained in the subsequent paragraph. Next, the examples that are covered by the optimised rule set are removed from the training data. Afterwards I-REP* is employed again to cover any remaining positive examples, adding new rules to the optimised rule sets. The last three steps of Ripper may be iterated k-times using the result of the last step as the input of the first one. Hence, the resulting algorithm is referred actually as Ripper$_k$ and Ripper stands basically for the instantiation of this algorithm that uses a single iteration ($k = 1$).

In the optimisation phase, Ripper evaluates the optimisation potential of each of its rules, according to their order in the decision list (from the first one to the last one). For each rule r_i, two alternatives, a replacement and a revision of r_i, are generated. The replacement $Rep(r_i)$ is grown, starting with an empty rule, and then pruned to minimise the error of the decision list in which $Rep(r_i)$

Algorithm 9 Ripper$_k$ [Coh95]

```
1: procedure RIPPER($Instances, SplitRatio, MDL_{Threshold}, k$)
2:     $DecisionList$ = I-REP*($Instances, SplitRatio, MDL_{Threshold}$)
3:     for $i \leftarrow 1, k$ do
4:         $DecisionList$ = OPTIMISERULESET($DecisionList$)
5:         REMOVECOVEREDINSTANCES($Instances, DecisionList$)
6:         APPEND($DecisionList$, I-REP*($Instances, SplitRatio, MDL_{Threshold}$))
7:     return $DecisionList$
```

replaces the original rule r_i. The revision $Rev(r_i)$ is generated analogously, but the growing starts with the original rule r_i. The revised rule replaces the original rule in the decision list, too. Thus, we obtain three decision lists: the original one and two decision lists that contain the replacement or the revision of the rule. The minimum description length of each decision list is calculated deleting the rules beforehand that increase the description length. Afterwards, a decision is made if the original decision list, its replacement or revision is included in the original rule set choosing the rule whose rule set has the minimal description length.

Contrary to the original I-Rep, that has been designed inherently as a binary learner, I-Rep* and consequently Ripper may also solve multi-class learning problems. To this end, I-Rep* employs internally class binarisation methods - either an ordered or unordered class binarisation - to break down the multi-class problem into several binary ones. Depending on the class binarisation method employed in I-Rep*, Ripper is run either in its ordered or unordered mode. In the ordered mode, Ripper's default mode, I-Rep* utilises an ordered class binarisation that orders the classes according to their frequency in the training data, generates rules that distinguishes one class from the more frequent classes respectively, and arranges the resulting rules in a decision list according to this ordering. For the most frequent class, a default rule is added that predicts this class. In the unordered mode, I-Rep* learns one rule set for each class that distinguishes this class from all other classes. For prediction, the most precise covering rule in these rule sets is determined and used for prediction.

Please note that the pseudocodes of the previously introduced algorithms (Algorithm 8 and 9) describe consciously binary versions of the correspondent algorithms for ease of notation and understanding.

2.5 The LeGo framework: From Local Patterns to Global Models

In this section, we describe the generic LeGo framework (an acronym for "from **l**ocal patt**e**rns to **g**lobal m**o**dels") [KCFS08] that may be utilised for a variety of data mining task. In this framework, the process of solving a data mining task is broken down into three subsequent steps. In the first step, a (local) pattern discovery algorithm is employed to generate a (large) set of local patterns. Afterwards, this set is reduced to a smaller one that at best contains only highly informative and non-redundant patterns in the second step. In the third and last step, the reduced pattern set is used to build a global model. Since none of these steps is tied to a specific algorithm, several options are available for each step, making the framework very flexible and adaptive. Many existing data mining algorithms that are based on pattern discovery may be modelled in this framework. Furthermore, the LeGo framework may be fitted into the well-known KDD process. On the one hand, it provides a generic approach for the data mining phase of the KDD process. On the other hand, it shares certain similarities to other phases of the KDD process, as we will see later.

In the following, we illustrate this approach by an exemplary data mining algorithm, give a short summary of the three steps of the LeGo framework afterwards and discuss its advantages in the last section of this chapter.

2.5.1 Illustrative Example

The field of local pattern discovery provides a great range of techniques that produce extensive collections of (local) patterns [MBS04]. Since most of these techniques are exhaustive by nature, the discovered pattern collections comprehend more or less the complete information content of the database. However, the knowledge fragments represented by the local patterns need to be further processed into a global model. As the pattern collections obtained by local pattern discovery are usually rather large and show high redundancy, a reduction to a (small) subset of patterns may be worthwhile. To this end, the pattern set discovery step tries to select only patterns that are highly informative in the context of the global data mining problem. Using the reduced pattern set, the global modelling step employs data mining methods to generate a global model.

A well-known example for an algorithm that utilises this step-wise approach is CBA (an acronym for **c**lassification **by** **a**ssociation) [LHM98]. For the local pattern discovery, it employs conventional association rule learning algorithms (for example, the Apriori algorithm, see Section 2.3.3.4) to generate association rules. Afterwards, the pattern collection achieved in this way is reduced to a pattern set that contains only classification association rules. At last, CBA typically employs a simple covering algorithm. For this purpose, each candidate rule is rated by some heuristic and the best rated one is repeatedly added to the global model which most often is either a disjunction of rules or a decision list.

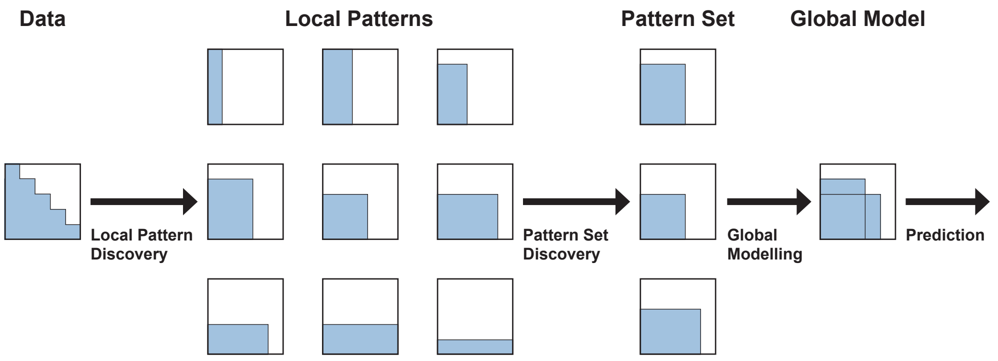


Figure 2.13.: The LeGo framework [KCFS08]

In many cases, other data mining algorithms may not separated into the LeGo phases as easily as the CBA algorithm could be separated, since the individual phases may be interleaved (for example, SeCo algorithms, see Section 2.4.2). Consequently, these algorithms may not easily be recognised as instantiation of the LeGo framework.

2.5.2 Local Pattern Discovery

Local pattern discovery explores (most often exhaustively) the pattern search space, which is defined by the employed local pattern discovery algorithm (e.g. frequent itemset mining algorithms search the itemset space), for candidate patterns that adhere to user-specified constraints (e.g. the minimum support for frequent itemsets). Usually, the quality of the found candidate patterns is rated either by individual properties (e.g. by their support in the data set) or in the context of some global data mining task (e.g. by their predictive power in a classification context). In a way, local pattern discovery may be seen as an automation of the feature construction phase of the KDD process.

2.5.3 Pattern Set Discovery

Usually, the algorithms used for local pattern discovery produce rather large pattern collections. As the contained patterns are normally judged only by their individual merits without consideration of their correlation to other patterns, the pattern collections may exhibit high redundancy. Both the large size and the redundancy of a pattern collection may be a hindrance for its subsequent utilisation. Obviously, a manual interpretation of the patterns by humans is only practicable for an acceptably small number of patterns. Similarly, many data mining methods, especially machine learning methods for global modelling, are prone to large input data, as their computational efforts depend mainly on this quantity. The main goal of the pattern set discovery step is to solve these problems by reducing the pattern collection obtained by local pattern discovery to a pattern set that features only little redundancy and is highly informative for the considered data mining task. According to this, the pattern set discovery has a similar role as the feature selection step of the KDD process.

2.5.4 Global Modelling

In the two previous steps a large collection of local patterns was generated and afterwards reduced to a significantly smaller pattern set that consists of highly informative and non-redundant local patterns. The last step of the LeGo framework, the global modelling, aims to decode the information contained in the compressed pattern set into a valuable global model. To this end, mainly two approaches for handling the modelling task may be identified.

The first approach is straightforward since it basically uses the extracted patterns as binary features of a new data set as described above. In this way, every machine learning algorithm that is able to handle such a data set may be employed for the global modelling step. As the pattern quality has a higher influence on the quality of the global model than the selection of the machine learning algorithm or the tuning of its parameter, this approach (and the employed algorithm) may clearly benefit of the high-quality features obtained in this way.

The second approach which is especially associated with rule learning treats the pattern set as an ensemble of classifiers, considering each pattern (or rule) as a weak classifier. Thus, different strategies for ensembles may be employed. On the one hand, decoding methods may be utilised for decoding the predictions of the covering patterns (or rules) into a single global prediction, obtaining an implicit global model in doing so. On the other hand, ensemble methods that employ the pattern set as meta features (e.g. stacking) may be used to obtain a global model.

2.5.5 Advantages of the LeGo Framework

In this section, we discuss the motivation for the LeGo framework and its advantages. Due to its exploratory and exhaustive nature, the employment of the LeGo framework is usually more expensive and time-consuming than a direct induction of a global model by conventional means. However, the utilisation of this framework is motivated by several advantages that we will discuss in this section.

As an automated pattern generation is at least implicitly a component of many effective machine learning algorithms (e.g. rule induction), one may expect that a more exhaustive or exploratory local pattern discovery step will increase the accuracy of the global model induction. On the one hand, local pattern discovery should recognise most of the globally valuable patterns because they usually will exhibit locally a similarly high quality, too. In this way, the extracted local patterns should be sufficient to generate a valuable global model. On the other hand, the inherent pattern discovery of global modelling algorithms often is accomplished by a greedy search and consequently may get stuck in local optima. In contrast to this, local pattern discovery algorithms tend to select a set of high-quality patterns that typically subsumes most of the local optima. Thus, a bigger selection of locally optimal patterns may be considered in the following global modelling step. Clearly, both points argue for the employment of local pattern discovery as a preliminary pre-processing step for the subsequent global modelling as this approach should expectedly increase the quality of the induced model.

Due to its modularity, the LeGo framework is both a construction kit and experimental laboratory for global modelling. On the one hand, it allows to remodel or adjust existing data mining methods and to construct entirely new ones by selecting methods for each step. In this way, the framework enables the analysis and improvement of the employed data mining methods as they may be utilised in different contexts and related to other methods for the same step. The analysis may result in insights about alternative, better performing options for the individual steps, about potential synergies between employed methods, and about important research topics that may offer room for improvement. On the other hand, the results of each step may be used to obtain a better understanding of valuable local patterns. First, the induced global models may be used to identify valuable candidate patterns since the induced patterns are usually highly-informative and non-redundant. Therefore, the manual inspection or utilisation of these patterns may be worthwhile. Analogously, the pattern set obtained by the pattern set discovery shows what patterns were selected into it and how they are related to each other.

Additionally, the modular nature of the LeGo framework turns out to be advantageous for its optimal configuration and for the performance of the induced global model. Since the local pattern discovery and pattern set discovery steps may be respectively performed independently of the subsequent step(s), its intermediate results may be stored and utilised for different configurations of these steps. Obviously, the computational effort for trying out different configurations is perceivably reduced in this way. Thus, the identification of an optimal configuration for a given data mining task becomes more viable.

3 Theory Formation

In this chapter, we deal with the first aspect of the question, how a set of local patterns may be employed to obtain a global model. As mentioned before, this problem may be solved by the generic LeGo framework (see Section 2.5) that partitions this problem into three subsequent steps: the local pattern discovery, the pattern set discovery, and the global modelling. Since many methods are available for each of these three steps, an immense number of configurations of the LeGo framework are possible. But the question, how a set of local patterns may be utilised to obtain optimal predictions, has not been systematically investigated yet. Similar to the partitioning in three steps, this question comprises three subquestions:

- **Local pattern discovery:** How is an optimal local pattern set discovered?
- **Pattern set discovery:** How is an optimal subset selected from a local pattern set?
- **Global modelling:** How is a pattern set optimally employed as a global model?

We will concentrate on the empirical comparison of a limited selection of methods for each step and on the identification of the best choice(s) among these selected methods.

This chapter is organised as follows. In the subsequent Sections 3.1 to 3.3, we briefly address each of the above-mentioned questions and present the associated methods that we will compare in this context, respectively. Afterwards, we describe the experimental setup in Section 3.4 and discuss the results of our experiments that correspond to these questions in Section 3.5. At last, our conclusions on these experiments are summarised in Section 3.6.

3.1 Local Pattern Discovery

In the first step of the LeGo framework, a local pattern discovery algorithm (see Section 2.5.2) extracts an usually large set of local patterns which will be processed in the subsequent pattern set discovery and global modelling steps. Since these local patterns are the starting point for the later steps, we want to investigate what effect the utilisation of different local pattern discovery algorithms has on the resulting global model. To this end, we compare the following two algorithms:

- **BSD:** The bitset-based subgroup discovery algorithm (abbreviated BSD) is a novel branch-and-bound algorithm for the efficient generation subgroup patterns (see Section 2.3.2.3).

- **CHARM:** The association rule mining algorithm CHARM extracts efficiently closed association rules (see Section 2.3.3.5).

This decision is driven by the consideration to use two algorithms that differ in their generation approaches. On the one hand, CHARM extracts the set of frequent closed itemsets which is as mentioned before only a subset of all frequent itemsets. On the other hand, BSD generates efficiently a pre-defined number of patterns. Additionally, the computational efforts of the subsequent steps depend mainly on the number of local patterns. Both algorithms generate a manageable number of patterns, either inherently or by adjustment. Beside these two points, both algorithms differ also in several aspects (e.g. the employed evaluation methods or search strategy).

3.2 Pattern Set Discovery

The previous local pattern discovery step generates usually a large set of local patterns that normally is highly redundant and impractical for both human interpretation or data mining. Hence, the second phase of the LeGo framework, the pattern set discovery (see Section 2.5.3), reduces the large set of local patterns to a smaller subset that preferably should feature only little redundancy and should be highly informative in respect to the current data mining task. Ideally, the reduced set should contain only those local patterns that are needed to generate an optimal global model in the subsequent global modelling step.

To estimate the impact of this step on the global model, we consider five representatives of pattern set discovery methods.

- **All selector:** The first and most simple one is not obviously a pattern set discovery method, as it selects simply all previously generated patterns for the following global modelling step. Therefore, this "all selector" (abbreviated ALL) may be considered as the neutral counterpart to the global modelling techniques. In this way, we can check if the removal of the pattern set discovery step has an effect on the performance of the resulting global model.

The next two pattern set discovery methods are confidence selectors which basically select local patterns on the basis of their confidence value (see Equation 2.38).

- **Minimum confidence selector:** The minimum confidence selector (abbreviated MC) keeps all patterns that exceed a given minimum confidence threshold.

- **Greedy confidence selector:** The greedy confidence selector (abbreviated GC) retains only the (k) most confident patterns.

In either case, the resulting pattern set depends only on the individual quality of its members.

In contrast to this, the remaining two pattern set discovery methods, joint entropy and exclusive coverage, that are taken from [KH06b] consider the quality of the pattern set as a whole. These methods have in common that they select a pattern set of a pre-defined size k that maximises an evaluation measure for pattern sets. Due to the high number of possible pattern sets (of size k), the exact computation of the optimal solution is usually not feasible. Hence, the optimal solution is often only approximated (e.g. by an heuristic approach). Details on the efficient search for the optimal solution and implementation suggestions may be found in [KH06a].

Joint entropy and exclusive coverage differ in the utilisation of their eponymous evaluation methods for pattern sets. These methods share that the performance of a pattern set is computed based on the covering information which instances are covered by a pattern in this set.

- **Joint entropy:** Joint entropy (abbreviated JE) has been proposed for maximally informative k-itemsets [KH06a] but can also be applied to the general pattern set discovery task. Essentially, all patterns are treated as binary features so that the joint entropy for each pattern set is equal to the joint entropy of its features. The entropy measures the uniformity of the distribution of instances over different contingencies (by what patterns an instance is covered or not). For a considered pattern set R, each of its subsets $R' \subseteq R$ defines such a contingency and implies a correspondent subset of the data set D:

$$D_{R'} = \bigcap_{r_i \in R'} D_{r_i} \cap \bigcap_{r_j \in R \setminus R'} \left(D \setminus D_{r_j} \right) \tag{3.1}$$

  Each instance in this subset is covered by all the patterns in the pattern set $R'$and is not covered by any pattern in its complement $R \setminus R'$.

  Using these subsets, the joint entropy of a pattern set may be calculated as follows:

$$h_{je}(R, D) = -\sum_{R_i \subset R} \frac{D_{R_i}}{|D|} \log \frac{D_{R_i}}{|D|} \tag{3.2}$$

  A pattern set that maximises the joint entropy will optimise the power to distinguish between individuals [KH06b].

- **Exclusive coverage:** Exclusive coverage (abbreviated EC) tries to reduce the amount of overlap between patterns. Thus, a pattern set is more favourable if many instances are covered only by a single pattern. Essentially, exclusive coverage counts the coverage that is exclusive for each pattern.

$$h_{ec}(R,D) = \sum_{r_i \in R} \left| D_{r_i} \setminus \bigcup_{r_j \neq r_i} D_{r_j} \right| \tag{3.3}$$

3.3 Global Modelling

After the generation of the local patterns and their reduction to a pattern set, the last step of the LeGo framework, the global modelling (see Section 2.5.4), aims to obtain a global model based on the reduced pattern set. Basically, two approaches may be employed to solve this problem:

- **Generation of a global model:** The first approach employs the pattern set to induce an explicit global model. The model generation has to be performed only once and may take place in advance (before the prediction phase).

- **On-the-fly decoding:** The second approach decodes the individual predictions of the covering local patterns into a global prediction when needed (in the prediction phase).

For the first approach, there are many choices for global modelling techniques that may be applied on pattern sets. Essentially, any learning algorithm could be used to generate a global model at this point. However, we decide not to build an explicit global model but to utilise the obtained pattern set directly for classification, as all the selected patterns may influence the predictions in this way.

To this end, we treat the rules of the pattern set as an ensemble of local models whose predictions are decoded into a global prediction. Basically, any decoding method that is suitable for the decoding of a regular ensemble of classifiers may be used for this purpose. They usually have to be adjusted at least slightly, since local models influence only the predictions of instances they cover (in contrast to global models that cover all instances). For the decoding of the predictions, we consider two groups of techniques. The first group are voting methods which we already encountered in the context of ensemble learning in Section 2.4.3. They use the predictions of all covering rules as votes for the final prediction. The voting weight of a single rule may be based either on its heuristic quality or on its position in a ranking of all rules according to these heuristic qualities. The second group are known probabilistic methods which use estimated probabilities to predict the most probable class. The required probabilities are estimated on the basis of the commonly used Laplace-corrected precision $h_{laplace}$, which we introduced in Section 2.3, of each rule.

3.3.1 Voting Methods

Since a pattern set is quite similar to an ensemble of full-fledged classifiers, voting methods for classifier ensembles may be applied to pattern sets as well. However, the voting procedure has to be adjusted, as not all patterns will cover every instance. To this end, the individual predictions of the covering patterns are interpreted as votes for the predicted class. In this context, the different voting methods differ only in the weights they assign to the vote of a rule. Essentially, the classification, which is based on predictions of the pattern set, works as follows:

$$c_R(d) = \arg\max_{l \in L} \sum_{r \in R(d)} vote(r, l, d), \tag{3.4}$$

where $R(d)$ is the set of rules covering the instance d and $vote(r, l, d)$ is the voting weight that depends on the chosen voting method.

- **Best rule:** The first voting method best rule (abbreviated *BR*) considers, as hinted by its name, only the best pattern which covers the instance whose class label is to be predicted. To this end, the quality of the covering patterns is determined by the heuristic $h_{laplace}$ (see Equation 2.25). At first sight, best rule does not seem to be a voting method, but it is possible to choose voting weights that simulate its behaviour (by ordering the rules descendingly according to their quality and using exponentially decaying weights). Essentially, this method corresponds to using a decision list in which the rules are sorted according to their heuristic quality values.

The next two voting methods are voting and weighted voting which we already encountered in the context of ensemble learning. These methods have in common that they consider the votes of all covering rules.

- **Voting:** Voting (abbreviated *V*) assigns a weight of one to all covering rules, essentially this can be considered as counting the covering rules separately for each class:

$$vote_V(r, l, d) = \begin{cases} 1, & \textit{if } r \supseteq d \\ 0, & \textit{otherwise.} \end{cases} \tag{3.5}$$

- **Weighted voting:** Weighted voting (abbreviated *WV*) uses the heuristic $h_{laplace}$ to determine the voting weights of each rule, so basically the Laplace weights are counted for each class:

$$vote_{WV}(r,l,d) = \begin{cases} h_{laplace}(r,D), & \textit{if } r \supseteq d \\ 0, & \textit{otherwise.} \end{cases} \tag{3.6}$$

The last two voting methods linear weighted voting and inverse weighted voting [Mut04] differ from the two previously presented voting methods, voting and weighted voting, as they use a ranking that is based on the Laplace value of each covering rule to obtain the weighting of the votes. So, each rule r obtains a rank $rank(r)$ according to the descending Laplace sorting. The ranks are represented by integers, beginning with one for the best rule and ending with the total number of rules for the worst ($rank_{max} = |R(d)|$).

- **Linear weighted voting:** Linear weighted voting (abbreviated *LV*) employs for the calculation of the voting weights a linear monotonic decreasing function of the rank:

$$vote_{LV}(r,l,d) = \begin{cases} 1 - \frac{rank(r)}{rank_{max}+1}, & \textit{if } r \supseteq d \\ 0, & \textit{otherwise.} \end{cases} \tag{3.7}$$

- **Inverse weighted voting:** Inverse weighted voting (abbreviated *IV*) determines the voting weights with the help of an inverse proportional function of the rank:

$$vote_{IV}(r,l,d) = \begin{cases} \frac{1}{rank(r)}, & \textit{if } r \supseteq d \\ 0, & \textit{otherwise.} \end{cases} \tag{3.8}$$

3.3.2 Bayesian Decoding

Bayesian decoding (abbreviated *BD*) belongs to the probabilistic decoding methods. The main idea of these methods is to estimate class probabilities with the help of the covering patterns and their associated class probability distributions. To this end, the covering rules are determined first. For each covering rule, the conditional probability distribution that an instance belongs to a class under observation of the covering rule is estimated. Using these estimated local class probability distributions, the aggregated conditional probability distribution that an instance belongs to a class under observation of all the covering rules is approximated. On the basis of the aggregated class probability distribution, the supposedly most probable class is determined and predicted for the instance.

For a given instance d and a pattern set R, Bayesian decoding aims to estimate the probabilities that the instance belongs to a class l under the observation of the rules $R(d) = \left\{r_1, r_2, \dots r_{|R(d)|}\right\}$ that cover the instance, namely $\Pr(l|r \supseteq d)$, and predicts the most probable class according to these class probabilities:

$$c_R(d) = \arg\max_{l \in L} \Pr(l|R(d)). \tag{3.9}$$

These probabilities may be translated in an computable form by applying the Bayes theorem. This leads to the following formula:

$$\Pr(l|R(d)) = \frac{\Pr(R(d)|l) \cdot \Pr(l)}{\Pr(R(d))} \tag{3.10}$$

As the denominator $\Pr(R(d))$ does not depend on the considered classes, it does not affect the relative order of their estimated probabilities and may consequently be ignored. If we additionally assume that the observation of one of the rules $r_j \in R(d)$ is independent of the occurrence of the other rules, we can make the following naïve assumption:

$$\Pr(R(d)|l) = \Pr(r_1 \wedge r_2 \wedge \dots \wedge r_{|R(d)|}|l) = \prod_{r \in R(d)} \Pr(r|l) \tag{3.11}$$

In summary, the classification works as follows:

$$\arg\max_{l \in L} \Pr(l) \cdot \prod_{r \in R(d)} \Pr(r|l) \tag{3.12}$$

At last, we have to explain how the probabilities $\Pr(l)$ and $\Pr(r|l)$ may be estimated. The former may be determined simply by counting the training instances that belong to the class l and dividing this number by the total number of instances in the training data set D:

$$\Pr(l) = \frac{n_l}{|D|} \tag{3.13}$$

The latter probability $\Pr(r|l)$ can be estimated quite similarly. First, we determine the number of training instances that are covered by the rule r and belong to the class l and divide this number by the total number of instances that belong to this class. Usually, some rules do not cover any instances of a given class. Consequently, the total probability for this class would be zero, since a single zero probability will lead to a product of zero. To avoid this problem, we apply an adjusted Laplace correction (see Equation 2.25) to the estimated probabilities:

$$\Pr(r|l) = \frac{n_{r,l} + 1}{n_l + |R(d)|} \tag{3.14}$$

Essentially, the number of instances that are covered by the rule r and belong to the class is increased by one, increasing the total sum by the number of covering rules $|R(d)|$.

3.4 Experimental Setup

In our experiments, we aim to identify the best choice(s) among a limited selection for each step of the LeGo framework by an empirical comparison of their prediction quality measured by accuracy (see Equation 2.59). We will describe the implementation and configuration details of the methods for each step, and the evaluation of the results in this section.

3.4.1 Implementation of Algorithms

For our experiments, we use our own extended implementation of the LeGo framework (see Section 2.5) which has been integrated into the generic data mining framework Weka [WFHP16]. Its main extension is an efficient storage mechanism. The results of each step are stored and may be used several times for different configurations of the subsequent step(s). This is possible as each step of the LeGo framework only needs either the training data or the results of the previous one (in addition to the necessary configuration parameters) for its computations. In this way, the computational efforts of our experiments are noticeably reduced, as unnecessary repetitions of previous steps are avoided. For example, the local pattern discovery step is computed exactly twice: once for BSD and CHARM, respectively. These results may then be used as the input for every subsequent pattern set discovery algorithm. As our experimental results are evaluated by cross-validation, the stored information is organised by folds and comprises, dependent on the current step, either the discovered local patterns, the selected pattern set or the predictions for each training instance for each fold. The associated training instances are, except for the last step, stored, too.

For the local pattern discovery step, we selected, as already mentioned, two methods for comparison: BSD for the discovery of subgroups and CHARM for the discovery of association rules. We employ the Vikamine implementation of BSD [AL12] and our own implementation of CHARM. Both algorithms are initialised appropriately to generate classification patterns for each class of the (multi-class) learning problems. To this end, they are repeatedly executed to generate patterns for each class separately. Afterwards, the intermediate results of each algorithm are stored in a single set of local patterns. In each repetition, BSD uses another class-label pair as the concept of interest, generating local patterns for the currently selected class. For CHARM, the data set is partitioned into segments according to their class labels. Afterwards, CHARM is started on each segment separately, generating classification patterns for the associated class.

Vikamine's implementation of BSD has been integrated in our extended Weka framework and has been configured as follows. We choose weighted relative accuracy (see Equation 2.27) for the pattern evaluation, as it is a solid choice for pattern discovery. According to this heuristic, the best 100 patterns per class are generated. The maximum pattern length is set to 5 conditions. Aside of BSD's integrated relevancy filtering (see BSD in Section 2.3.2.3), no further filters are applied. Our own implementation of CHARM which has also been integrated in our extended Weka framework is set-up as follows. We employ both a relative and an absolute minimum support threshold collaboratively. The relative minimum support threshold is defined as 3% of the training set size which is equal to the class size of the current segment. The absolute minimum support threshold is set to two training instances. A pattern must exceed both thresholds to be considered frequent.

For the second phase, we implemented the five pattern set discovery methods that we described briefly in the previous section: the all selector (which basically returns its input), the minimum confidence selector, the greedy confidence selector, exclusive coverage, and joint entropy. For the minimum confidence selector, we set the minimum confidence threshold to 0.5. The selected pattern set size is set to 25 patterns for the greedy confidence selector, exclusive coverage, and joint entropy. We consider the implementation suggestions in [KH06a] and choose to employ a forward selection as proposed for the methods exclusive coverage and joint entropy. Thus, the pattern set is assembled incrementally by adding in each step the pattern which yields the highest reward for the given heuristic until the predetermined size (in our case 25 patterns) for the pattern set is reached.

For the third phase, we implemented the global modelling methods: best rule, voting, weighted voting, inverse weighted voting, linear weighted voting, and Bayesian decoding as described in the previous section. As these methods are not parametrised, there are no further configuration details to be mentioned.

For the comparison of two patterns, we use the following pattern properties for tie breaking in each step (in descending order of relevance): the heuristic value of the patterns, the number of instances that are correctly predicted, the number of instances of the predicted class, and the size of the pattern (prefer-

Table 3.1.: Data sets used in the experiments with the number of instances, number of nominal and numeric attributes, and number of classes.

		Attributes		
Data set	Instances	Nominal	Numeric	Classes
Aneal.orig	798	29	9	6
Autos	205	10	15	7
Balance-scale	625	0	4	3
Breast-cancer	286	9	0	2
Breast-w	699	0	9	2
Bridges Version 1	107	9	3	6
Cars	1728	6	0	4
CMC	1473	7	2	3
Diabetes	768	0	8	2
Ecoli	336	0	7	8
Glass	214	0	9	7
Heart-c	303	7	6	5
Heart-h	294	7	6	5
Heart-statlog	270	0	13	2
Hepatitis	155	14	5	2
Iris	150	0	4	3
Labor	57	8	8	2
Lymph	148	15	3	4
Postoperative-patient-data	90	8	0	3
Solar-flare-c	1389	10	0	9
Tic-tac-toe	958	9	0	2
Titanic	2201	3	0	2
Vowel	990	3	10	11
Yeast	1484	0	8	10
Zoo	101	15	1	7

ring larger patterns). If these criteria are not able to discriminate between two patterns, we choose one of the patterns at random.

3.4.2 Evaluation

For the evaluation of the considered methods, we configure every valid combination in our LeGo framework by selecting one method of the local pattern discovery, pattern set discovery and global modelling step, respectively. Their performance is measured by their accuracy that is estimated by a stratified 10-fold cross-validation (see Section 2.4.4.1). The detailed resulting performance scores are listed in Appendix A. On the basis of the previously estimated accuracy, we calculate the average accuracy per data set of each considered method. To this end, we average the estimated accuracy values of the learning algorithms that involve a considered method (e.g. the accuracy values of all learning algorithms that employed the minimum confidence selector for a given data set). In this way, we obtain a measure of the general performance of a method independent of the selected methods in the two other steps.

For each step, we evaluate the performance of the associated methods by statistical tests (see Section 2.4.4.2) using their average accuracy values for

Table 3.2.: Local pattern discovery: average accuracy (including standard deviation) per data set of each method.

Data Set	BSD	CHARM
Anneal.orig	.6583 ± .2294	.7813 ± .0324
Autos	.4747 ± .1154	.4318 ± .1153
Balance-scale	.5961 ± .1938	.5350 ± .2585
Breast-cancer	.7077 ± .0204	.6229 ± .1611
Breast-w	.8181 ± .1365	.7140 ± .2008
Bridges Version 1	.4994 ± .0970	.4982 ± .1021
Cars	.6170 ± .1693	.6088 ± .1843
CMC	.4431 ± .0344	.4133 ± .0637
Diabetes	.6689 ± .0372	.6129 ± .1410
Ecoli	.5417 ± .1571	.5442 ± .2250
Glass	.4623 ± .1095	.4730 ± .1717
Heart-c	.7436 ± .0867	.6512 ± .1598
Heart-h	.7707 ± .0682	.6653 ± .1826
Heart-statlog	.7496 ± .0705	.6360 ± .1705
Hepatitis	.7307 ± .0732	.7731 ± .0953
Iris	.7109 ± .1803	.7540 ± .1428
Labor	.6040 ± .1087	.6670 ± .0692
Lymph	.6436 ± .1244	.6184 ± .1564
Postoperative-patient-data	.5859 ± .1288	.6489 ± .1289
Solar-flare-c	.7548 ± .1455	.7068 ± .2797
Tic-tac-toe	.7503 ± .0999	.6643 ± .1727
Titanic	.6730 ± .1693	.4892 ± .2049
Vowel	.2688 ± .1364	.3006 ± .2284
Yeast	.3747 ± .1188	.3526 ± .1321
Zoo	.6654 ± .2309	.7105 ± .2212

comparison. For the evaluation of the local pattern discovery step, we employ the Wilcoxon signed ranks test, as it is suitable for the comparison of two methods (in our case the two local pattern discovery algorithms BSD and CHARM). For the evaluation of the other two steps, in which more than two methods are involved, we use the Friedman test and, if necessary, a post hoc Nemenyi test. The results of the latter are visualised by a critical distance chart, respectively. For all the statistical tests, a significance level of 5% is chosen.

3.4.3 Data

For our experiments, we use 25 data sets of the UCI repository [Lic13]. These data sets were chosen for a great variety of the number of instances and classes, and different ratios between numerical and nominal attributes. Numerical attributes are discretised [FI93] separately for each cross-validation fold, using only the information contained in its training data. Missing (numerical and nominal) attribute values are ignored. The statistical properties of the utilised data sets are displayed in Table 3.1 which contains the number of classes, instances, and attributes (separately for numerical and nominal attributes).

Table 3.3.: Pattern set discovery: average accuracy (including standard deviation) per data set of each method.

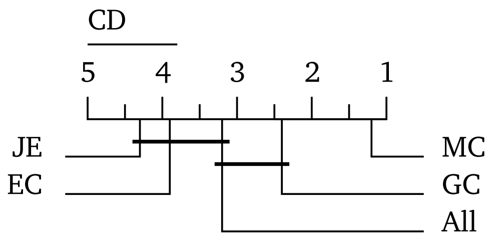
Data Set	ALL	EC	GC	JE	MC
Anneal.orig	.6824 ± .1723	.8096 ± .0476	.7617 ± .0000	.5328 ± .2541	.8124 ± .0428
Autos	.5197 ± .1056	.4333 ± .1186	.3906 ± .0496	.3388 ± .0305	.5839 ± .0360
Balance-scale	.5157 ± .2776	.4522 ± .2188	.6464 ± .1783	.4704 ± .2149	.7431 ± .0146
Breast-cancer	.6349 ± .1529	.6049 ± .1866	.7125 ± .0132	.6555 ± .0792	.7186 ± .0239
Breast-w	.8681 ± .1655	.7415 ± .2586	.6552 ± .0000	.6508 ± .0180	.9147 ± .0880
Bridges Vers. 1	.4653 ± .1508	.4393 ± .0235	.5605 ± .0095	.4386 ± .0664	.5903 ± .0359
Cars	.5673 ± .2548	.4916 ± .1687	.7002 ± .0000	.5760 ± .1488	.7292 ± .0433
CMC	.4152 ± .0779	.3986 ± .0550	.4270 ± .0000	.4261 ± .0418	.4742 ± .0140
Diabetes	.6394 ± .1346	.6118 ± .1387	.6563 ± .0052	.5954 ± .1093	.7014 ± .0311
Ecoli	.5447 ± .1931	.3170 ± .1387	.6773 ± .0593	.4668 ± .1283	.7089 ± .0965
Glass	.4801 ± .1558	.3057 ± .1032	.5122 ± .0278	.4189 ± .0979	.6214 ± .0643
Heart-c	.7253 ± .1787	.6453 ± .1276	.7610 ± .0164	.5583 ± .0775	.7972 ± .0446
Heart-h	.7023 ± .2104	.7205 ± .0982	.7774 ± .0022	.5937 ± .1641	.7963 ± .0518
Heart-statlog	.7130 ± .1935	.6571 ± .1344	.7219 ± .0454	.5735 ± .1196	.7988 ± .0366
Hepatitis	.7270 ± .1393	.7669 ± .0508	.7893 ± .0057	.6902 ± .0859	.7861 ± .0414
Iris	.7839 ± .1927	.6094 ± .1239	.7706 ± .0967	.6767 ± .1169	.8217 ± .1697
Labor	.6836 ± .0804	.6069 ± .0737	.6778 ± .0202	.5086 ± .0812	.7006 ± .0486
Lymph	.6833 ± .1761	.5869 ± .1112	.6342 ± .0811	.5123 ± .0683	.7383 ± .1256
Postop.-p.-d.	.5259 ± .1954	.5463 ± .1193	.7111 ± .0000	.6472 ± .0699	.6565 ± .0777
Solar-flare-c	.6256 ± .2727	.6202 ± .3255	.8522 ± .0012	.7045 ± .1335	.8513 ± .0008
Tic-tac-toe	.6942 ± .1936	.6947 ± .1832	.7649 ± .0011	.6169 ± .1135	.7657 ± .0980
Titanic	.5114 ± .2037	.5975 ± .1960	.6202 ± .2116	.5624 ± .2089	.6140 ± .2060
Vowel	.3949 ± .1644	.1797 ± .0561	.1461 ± .0389	.1550 ± .0360	.5479 ± .1293
Yeast	.3696 ± .1810	.3010 ± .0533	.3766 ± .0434	.2681 ± .0945	.5029 ± .0217
Zoo	.8846 ± .1185	.7841 ± .0435	.4064 ± .0000	.4459 ± .0395	.9188 ± .0398
Average Rank	3,20	3,92	2,40	4,32	1,16

3.5 Experimental Results

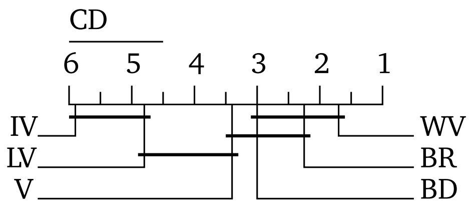
In this section, we discuss the results of our experiments. To this end, we will separately discuss each step, aiming to identify the best method(s) respectively.

3.5.1 Local Pattern Discovery

For the local pattern discovery step, we compare the two learning algorithms BSD and CHARM. The summary of the average performance of these algorithms is depicted in Table 3.2. BSD, which generates a pre-defined number of the best patterns according to weighted relative accuracy, wins 16 times against CHARM, which generates all confident closed patterns. Vice versa CHARM outperforms BSD in 9 cases. Despite the slightly better performance of BSD, the performances of both algorithms do not differ significantly according to the applied Wilcoxon signed ranks test, Nevertheless, both algorithms are clearly preferable for specific data sets, e.g.titanic for BSD or anneal.orig for CHARM. So, we conclude that these two local pattern discovery algorithms are interchangeable in this process.



(a) pattern set discovery



(b) global modelling

Figure 3.1.: Critical distance charts.

3.5.2 Pattern Set Discovery

For the pattern set discovery, we compare the all selector, the minimum confidence selector, the greedy confidence selector, and the two forward selectors that utilise exclusive coverage and joint entropy, respectively. Their average performances on each data set are summarised in Table 3.3. The Friedman test is used to evaluate these results. It shows that the considered pattern set discovery methods exhibit significantly different performances. Hence, a post hoc Nemenyi test is applied whose results are depicted in the critical distance chart in Figure 3.1(a).

The Nemenyi test identifies three groups of methods whose performances do not differ significantly. The third and worst group contains the all selector and the two forward selectors that employ exclusive coverage and joint entropy. The latter two selectors are slightly but not significantly worse than the all selector. However, the observed decrease in performance is not significant, while the 25 selected patterns are only a very small fraction of the total set of discovered local patterns. Additionally, the two selectors employed a forward selection which is a greedy approximation of the optimal solution. So, there may still be some potential that has not been exploited yet.

The second group consists of the all selector and the greedy confidence selector. Both algorithms do not differ in performance, but the greedy confidence selector selects only 25 local patterns. Consequently, we obtain in this way a pattern set that both retains the predictive performance of the total set of local patterns and reduces the computational efforts of the subsequent global modelling step.

The first and best group consists of a single member: the minimum confidence selector. It performs significantly better than all other considered methods. Hence, it is clearly the optimal choice amongst our selection if we do not consider the pattern set size. The number of selected patterns is obviously the reason for the superiority of the minimum confidence selector which selects all confident local patterns, as this is the only difference to the second best algorithm greedy confidence selector which selects only the 25 most confident patterns. The multiplicity of confident local patterns is advantageous as it allows a better coverage of the pattern space and higher diversity of predictions.

In summary, the employment of a pattern set discovery step is obviously beneficial as all selectors which selected a real subset of the total of local patterns were either comparable to or better than the all selector which is essentially analogous to skipping the pattern set discovery step. Confidence seems to be a simple but efficient criterion in this scenario. Since both local pattern discovery algorithms avoid to generate too specific patterns (by a maximum rule length criterion and a minimum support threshold, respectively), confidence is in this case not prone to overfitting and may improve the prediction quality of the selected pattern set.

3.5.3 Global Modelling

For the global modelling, we compare the decoding methods Bayesian decoding, best rule, voting, weighted voting, inverse weighted voting, and linear weighted voting. Their average performance on each data set is summarised in Tables 3.4 and 3.5. Analogously to the pattern set discovery step, the Friedman test is used for the evaluation of these results. Again, it shows that the considered global modelling methods feature significantly different performances. Hence, a post hoc Nemenyi test is applied whose results are depicted in Figure 3.1(b). Four groups of comparable global modelling methods may be identified.

The fourth and worst group comprises only the ranking based decoding methods inverse weighted voting and linear weighted voting. Both methods use the quality of the covering patterns only for the calculation of a list of ascending ranking scores. In this way, the exact quality values are not incorporated directly. Patterns with close quality values may be assigned highly different weights by inverse weighted voting and linear weighted voting as these weights exhibit a hyperbolic or linear decline, respectively.

The third group consists of linear weighted voting and voting. As voting also ignores the exact quality values, this is not very surprising. Nevertheless,

Table 3.4.: Global modelling (part A): average accuracy (including standard deviation) per data set of each method.

Data Set	BD	BR	IV
Anneal.orig	.7783 ± .0683	.7266 ± .1609	.6689 ± .2160
Autos	.4578 ± .0839	.4903 ± .1200	.3975 ± .1347
Balance-scale	.6790 ± .0549	.7013 ± .0443	.3068 ± .2768
Breast-cancer	.7053 ± .0124	.7141 ± .0088	.5929 ± .1650
Breast-w	.7356 ± .0984	.7980 ± .1163	.7079 ± .2268
Bridges Version 1	.5142 ± .0653	.5184 ± .0561	.4184 ± .1554
Cars	.7014 ± .0039	.6821 ± .0494	.4373 ± .2579
CMC	.4520 ± .0248	.4527 ± .0245	.3804 ± .0738
Diabetes	.6857 ± .0299	.6920 ± .0331	.5530 ± .1369
Ecoli	.5703 ± .0952	.5715 ± .1335	.4236 ± .2281
Glass	.5540 ± .0858	.4732 ± .0807	.3548 ± .1726
Heart-c	.7166 ± .0835	.7380 ± .0905	.6035 ± .1857
Heart-h	.7322 ± .0786	.7671 ± .0654	.6360 ± .2174
Heart-statlog	.7193 ± .0735	.7378 ± .0704	.6007 ± .1980
Hepatitis	.7478 ± .0584	.7875 ± .0360	.6670 ± .1460
Iris	.5420 ± .1283	.8007 ± .1369	.6787 ± .1548
Labor	.6530 ± .1119	.6523 ± .0996	.5950 ± .0816
Lymph	.5620 ± .1232	.6729 ± .1189	.5735 ± .1526
Postoperative-patient-data	.7044 ± .0133	.6856 ± .0517	.5067 ± .1748
Solar-flare-c	.8511 ± .0012	.8259 ± .0467	.5567 ± .3217
Tic-tac-toe	.7501 ± .0574	.8390 ± .1253	.5770 ± .1664
Titanic	.4131 ± .1801	.7317 ± .0434	.4344 ± .1743
Vowel	.3166 ± .1726	.3067 ± .1763	.2172 ± .1897
Yeast	.4030 ± .0650	.4070 ± .0787	.2570 ± .1758
Zoo	.7057 ± .2316	.6957 ± .2274	.6330 ± .2011
Average Rank	3.00	2.24	5.88

the linear weighted voting seems to be more on a par to voting than inverse weighted voting is. A reason for this could be that the above-mentioned decay of weights is less severe for linear weighted voting than for inverse weighted voting (linear versus hyperbolic).

The second group contains voting, best rule, and Bayesian decoding. Although the latter two methods utilise the quality values of the covering patterns and feature therefore slightly better performances, they are comparable to voting which ignores, as mentioned above, these values. However, the two methods belong to the next group of comparable methods, too.

The first and best group comprises the three methods weighted voting, best rule, and Bayesian decoding which employ the accuracy values of the covering patterns. The performance of the probabilistic Bayesian decoding is a little bit lower than the performance of the other two methods. Bayesian decoding assigns for every pattern an estimated probability to each class. In this way, it may happen that classes that are predicted by many weak patterns are preferred as the voting scores of classes that are covered by fewer but stronger patterns slowly decrease. So, we assume that single (or fewer) strong patterns are preferable to a bigger number of weak patterns, especially if those patterns exhibit (high) redundancy. This assumption is supported by the fact

Table 3.5.: Global modelling (part B): average accuracy (including standard deviation) per data set of each method.

Data Set	LV	V	WV
Anneal.orig	.6883 ± .1912	.7106 ± .1750	.7461 ± .1774
Autos	.4295 ± .1217	.4673 ± .1072	.4770 ± .1042
Balance-scale	.3869 ± .2607	.6147 ± .1255	.7047 ± .0377
Breast-cancer	.6070 ± .1710	.6532 ± .1313	.7190 ± .0183
Breast-w	.7647 ± .2128	.7677 ± .2125	.8227 ± .1398
Bridges Version 1	.4865 ± .0837	.5207 ± .0714	.5345 ± .0823
Cars	.5188 ± .1803	.6364 ± .1441	.7012 ± .0712
CMC	.4033 ± .0530	.4288 ± .0485	.4522 ± .0241
Diabetes	.5871 ± .1257	.6342 ± .1133	.6932 ± .0339
Ecoli	.4928 ± .2258	.5716 ± .2002	.6280 ± .1719
Glass	.4316 ± .1478	.4737 ± .1452	.5186 ± .1175
Heart-c	.6744 ± .1491	.7086 ± .1341	.7433 ± .0922
Heart-h	.6721 ± .1907	.7304 ± .1308	.7704 ± .0715
Heart-statlog	.6426 ± .1710	.7074 ± .1387	.7493 ± .0794
Hepatitis	.7488 ± .0671	.7780 ± .0562	.7826 ± .0470
Iris	.7433 ± .1254	.8040 ± .1087	.8260 ± .1305
Labor	.6343 ± .0851	.6393 ± .0860	.6390 ± .0990
Lymph	.6374 ± .1320	.6617 ± .1505	.6784 ± .1229
Postoperative-patient-data	.5478 ± .1412	.6044 ± .1200	.6556 ± .0829
Solar-flare-c	.6156 ± .2583	.7136 ± .2261	.8217 ± .0495
Tic-tac-toe	.5972 ± .1673	.7309 ± .0443	.7496 ± .0572
Titanic	.5522 ± .2054	.5829 ± .2134	.7722 ± .0193
Vowel	.2514 ± .1847	.2965 ± .1848	.3201 ± .2003
Yeast	.3082 ± .1289	.3993 ± .0942	.4076 ± .0867
Zoo	.6850 ± .2202	.7056 ± .2387	.7026 ± .2338
Average Rank	4.80	3.40	1.68

that the method best rule, whose predictions are based only on a single but strong pattern, ranks before Bayesian decoding, which uses all covering patterns. Nevertheless, weighted voting, which performed best, also exploits the confidence scores of all covering patterns. However, this increases only the voting scores of the predicted class by summation. Thus, the above-mentioned problem of Bayesian decoding is mitigated because the weak patterns of a class do not influence the scores of other classes and the voting score of the class increases only slowly.

In summary, the members of the best group global modelling methods weighted voting, best rule, and Bayesian decoding that exploit the quality values of the local patterns performed better than the methods voting, linear weighted voting, and inverse weighted voting which use this information only indirectly as a coarsened or derived vote. Voting that assigns only the value of 1 to the predicted class independent of the pattern quality is still comparable to best rule and Bayesian decoding. Inverse weighted voting and linear weighted voting that derive ranking based votes are significantly worse than the above-mentioned best group.

3.6 Summary

In this chapter, we concerned ourselves with the question, how a set of local patterns may be utilised to obtain optimal predictions. Since the generic LeGo framework solves this problem by partitioning it into three steps: the local pattern discovery, the pattern set discovery, and the global modelling, we split the problem into three subquestions that we aimed to investigate. To this end, we performed an empirical comparison of a selection of methods for each of the three associated steps:

- **Local pattern discovery:** How is an optimal local pattern set discovered?

  For the local pattern discovery step, we compared the two discovery algorithms BSD and CHARM. The performance of both algorithms did not differ significantly, even as the discovery approaches of these algorithms differ in various aspects (e.g. the employed evaluation methods or search strategy). In summary, the two employed local pattern discovery algorithms are interchangeable in this process.

- **Pattern set discovery:** How is an optimal pattern set selected?

  For the pattern set discovery step, we compared five methods: the all selector which is essentially analogous to skipping the pattern set discovery step, the two confidence-based methods minimum confidence selector and greedy confidence selector, and the two methods exclusive coverage and joint entropy that employed a forward selection. Since all the methods were better or at least comparable to the all selector, the pattern set discovery step is clearly worthwhile. The two confidence-based methods performed better than the other three methods. In the case of the minimum confidence selector, the performance was significantly better than those of the remaining methods. Since both employed local pattern discovery algorithms feature an integrated overfitting avoidance, confidence's known proneness to it does not come into effect. Hence, confidence is a simple but efficient criterion in this setting.

- **Global modelling:** How is a pattern set optimally employed as a global model?

  For the global modelling step, we compared six methods: Bayesian decoding, best rule, inverse weighted voting, linear weighted voting, voting, and weighted voting, that differ in their approach to exploiting the covering information. Weighted voting, best rule, and Bayesian decoding that exploit the exact quality values of the local patterns performed better than the methods voting, linear weighted voting, and inverse weighted voting which use this information only indirectly either as a binary or derived ranking based vote.

In summary, we conclude that the optimal way to utilise local patterns for prediction is to select a pattern set according to the confidence values of the patterns and simply add up these confidence values for a global prediction.

4 Probability Estimation

In this chapter, we focus on the question, how a set of local patterns may be utilised to obtain proper class probabilities. Such a class probability represents the likelihood that a data instance belongs to the associated class. The knowledge of these probabilities may be advantageous, since occasionally a simple prediction may turn out to be inadequate. Many practical applications require a finer distinction between instances than is provided by the simple prediction of class labels. For example, one may want to be able to provide a confidence score for the certainty of a prediction or rank instances according to their probability of belonging to a given class. These and many other tasks may be solved if class probabilities are available.

Unfortunately, class probabilities are usually unknown and may not be determined exactly. This problem may be solved by the estimation of class probabilities on the basis of the available data. Basically, the estimated class probabilities for a given instance are based on the correlation between its attribute values and the observed attribute values for each class. Consequently, the considered data may hamper the class probability estimation in two ways. First, data sets are usually only samples that cover only small parts of the whole data space. Such a sample may not be representative for the whole and thus lead to falsified probability estimations. Second, the data may contain noise which may also have a negative effect on the probability estimation, as it may distort the relation between attribute values and classes.

Thus, we are going to investigate how class probabilities may be estimated as accurately as possible, using a given set of local patterns. This problem may be divided into two sub task: the probability estimation and the probability aggregation. As we will see, the probability estimation may be further split into the basic probability estimation and shrinkage tasks. We are going to examine the three aforementioned tasks in this chapter:

- **Basic probability estimation:** The first and fundamental task aims to determine the class probabilities of a given data instance under observation of a single covering local pattern. For this purpose, we employ basic probability estimation methods which estimate probability distributions on the basis of the statistical properties of the considered pattern.
- **Shrinkage:** The second task addresses the meta technique shrinkage that intends to improve the probability estimates of (low coverage) patterns by smoothing the estimates of the basic methods.

- **Probability aggregation:** The third and last task deals with the aggregation of the class probability estimates of several covering patterns. To this end, we utilise well-known aggregation methods to combine the probability estimations generated by the first task. In this way, a single, preferably more accurate class probability distribution is obtained.

The remaining chapter is organised as follows. First, we will explain the motivation for our work on probability estimation for rule learning in Section 4.1. Afterwards, we briefly describe the basics of probabilistic rule learning in Section 4.2, recapitulate the basic probability estimation techniques in Section 4.3 and explain our adapted shrinkage approach in Section 4.4. Next, we investigate how the class probability distributions of multiple covering patterns may aggregated into a single one in Section 4.5. The performance of the introduced probability estimation techniques are evaluated in our experiments which are described in Section 4.6 and analysed in Sections 4.7 and 4.8. In the end, we summarise our conclusions in Section 4.9.

4.1 Motivation

The main focus of learning algorithms, such as decision tree learners and pattern discovery algorithms, is to produce a comprehensible explanation for a class variable. However, many practical applications, as mentioned before, require a finer distinction between instances than is provided by a prediction of class labels. For example, one would like to provide a confidence score that estimates the certainty of a prediction, rank instances according to their probability of belonging to a given class, make cost-sensitive predictions, or combine multiple predictions into a single one. All these problems may be solved straightforwardly if we can predict a probability distribution over all classes instead of the simple prediction of a single class value.

As decision trees share some similarities with rule-based local patterns and global models (for instance, a decision tree may be modelled as a decision list), we took the preliminary work on probability estimation on the basis of probabilistic decision trees (so-called probability estimation trees (PETS)) as a starting point for our considerations. A straightforward approach to estimate probability distributions using classification patterns is to compute the fractions of the covered instances for each class. However, this naïve approach has obvious disadvantages, such as that patterns that cover only a few instances may lead to extreme probability estimates. Thus, the probability estimates need to be smoothed.

A very simple but quite powerful technique for improving class probability estimates is the utilisation of *m*-estimates, or their special case, the Laplace-estimate [Ces90]. [PD03] showed that unpruned decision trees with Laplace-corrected probability estimates at the leaves produce quite reliable probability estimates. [FFH03] proposed a recursive computation of the m-estimate, which uses the probability distribution at level k as the prior probabilities for level $k + 1$. [WZ06] used a general shrinkage approach, which interpolates the estimated class distribution at the leaf nodes with the estimates in interior nodes on the path from the root to the leaf. Instead of trying to improve the probability estimates for each individual leaf, one can also resort to averaging multiple estimates, thereby reducing the variance of the resulting probability estimates. For example, a technique based on bagging multiple unpruned decision trees was used in [Dom99] to obtain improved probability estimates, which were subsequently used for cost-sensitive classification.

An interesting observation is that, contrary to classification, class probability estimation by decision trees typically works better on unpruned trees than on pruned trees. The explanation for this is simply that, as all instances in a leaf receive the same probability estimate, pruned trees provide a much coarser ranking than unpruned trees. [HV09] have provided a simple but elegant analysis of this phenomenon, which shows that replacing a leaf with a subtree can only lead to an increase in the area under the ROC curve (abbreviated AUC), a commonly used measure for the ranking capabilities of an algorithm. Of course, this only holds for the AUC estimate on the training data, but it still may provide a strong indication why unpruned PETs typically also outperform pruned PETs on the test set.

In contrast to the amount of work on probability estimation by decision trees, there has been hardly any systematic work on probability estimation on the basis of local patterns. A key difference between probability estimation based on decision trees and local patterns is that, in the case of decision trees, probability estimates will not change the prediction for an instance, because the predicted class only depends on the estimated probability distribution of a single leaf of the tree, and such local probability estimates are typically monotone in the sense that they all maintain the majority class as the class with the maximum probability. In the case of pattern discovery, on the other hand, each instance may be classified by multiple rules, which may possibly predict different classes. As many tie breaking strategies depend on the class probabilities, a local change in the class probability of a single rule may change the global prediction of the pattern-based classifier.

Because of these non-local effects, it is not evident that the same methods that work well for decision tree learning will also work well for pattern discovery. For example, the above-mentioned argument that unpruned trees will lead to a better (training set) AUC than pruned trees, does not straightforwardly carry over to pattern discovery, because the replacement of a leaf with a subtree is a local operation that only affects the instances that are covered by this leaf. In pattern discovery, on the other hand, each instance may be covered by multiple patterns, so that the effect of replacing one pattern with multiple, more specific patterns is less predictable. Moreover, every instance will be covered by some leaf in a decision tree, whereas each pattern-based classifier needs to induce a separate default pattern that covers the instances that are covered by no other pattern.

4.2 Probabilistic Patterns

In this part of our work, we will focus on, as mentioned before, probabilistic patterns which are an extension of regular classification patterns of the form

$$condition_1 \wedge \cdots \wedge condition_{|r|} \implies l \tag{4.1}$$

that we have dealt with so far. Contrary to such regular patterns, probabilistic patterns do not simply predict a single class label $l$but a probability distribution over all possible class values:

$$condition_1 \wedge \cdots \wedge condition_{|r|} \implies \underbrace{\big(\Pr(l_1|r \supseteq d), \cdots, \Pr(l_{|L|}|r \supseteq d)\big)}_{\text{class probabilities}} \tag{4.2}$$

This probability distribution consists of all probabilities that a covered instance d belongs to any of the classes in the data set, so we get one conditional class probability per class

$$\Pr(l_i|r \supseteq d). \tag{4.3}$$

When a probabilistic rule is used for classification, covered instances are then classified with the most probable class. Note that all instances which are classified by the same pattern receive the same probability distribution. Therefore, the probability distribution of each pattern can be calculated in advance.

Obviously, the set of class probabilities can be denoted as a vector of probabilities sorted by the class ordering:

$$\vec{\Pr}(L|r \supseteq d) = (\Pr(l_1|r \supseteq d), \cdots, \Pr(l_{|L|}|r \supseteq d)) \tag{4.4}$$

On the vector $\vec{\Pr}(L|r \supseteq d)$ we define the following maximum function

$$\max\big(\vec{\Pr}(L|r \supseteq d)\big) = \max_{l \in L} \Pr(l|r \supseteq d). \tag{4.5}$$

On sets of class probability vectors $\bigcup_{j=1}^{k}\{\vec{\Pr}(L|r_j \supseteq d)\}$, we define the average function

$$\text{avg}\Big(\bigcup_{j=1}^{k}\{\vec{\Pr}(L|r_j \supseteq d)\}\Big) = \frac{1}{k}\sum_{j=1}^{k}\vec{\Pr}(L|r_j \supseteq d) \tag{4.6}$$

and the multiplication

$$\begin{aligned}&\operatorname{mult}\left(\bigcup_{j=1}^{k}\left\{\vec{\Pr}(L|r_j \supseteq d)\right\}\right) \\ =&\frac{\left(\prod_{j=1}^{k}\Pr(l_1|r_j \supseteq d),\cdots,\prod_{j=1}^{k}\Pr(l_{|L|}|r_j \supseteq d)\right)}{\left\|\left(\prod_{j=1}^{k}\Pr(l_1|r_j \supseteq d),\cdots,\prod_{j=1}^{k}\Pr(l_{|L|}|r_j \supseteq d)\right)\right\|_1}\end{aligned} \tag{4.7}$$

Obviously, the results of the average and multiplication functions are again class probability vectors.

4.3 Basic Probability Estimation

In this section, we will review the three basic probability estimation methods that we employ in our experiments. The probability estimates of these methods are based on the relation between the number of instances covered by the pattern n_r and the number of instances that are covered by the pattern but also belong to a specific class $n_{r,l}$. The exact differences between these methods are minor modifications of the calculation of this relation.

- **Precision:** The simplest approach to pattern probability estimation is precision. It estimates directly a class probability distribution of a pattern by the fraction of instances that belong to each class:

$$\Pr_{prec}(l|r \supseteq d) = \frac{n_{r,l}}{n_r} \tag{4.8}$$

  This naïve approach has several well-known disadvantages, most notably that patterns with a low coverage may lead to extreme probability values. For this reason, [Ces90] suggested the use of the Laplace- and m-estimates.

- **Laplace-estimate:** The Laplace-estimate modifies the above-mentioned relation by adding one additional instance to the counts $n_{r,l}$ for each class $l \in L$. Hence, the number of covered instances n_r is increased by the number of classes $|L|$:

$$\Pr_{laplace}(l|r \supseteq d) = \frac{n_{r,l}+1}{n_r+|L|} \tag{4.9}$$

  It may be viewed as a trade-off between $\Pr_{prec}(l|r \supseteq d)$ and an a priori probability of $\Pr(l) = {}^{1}/_{|L|}$ for each class. Thus, it implicitly assumes a uniform class distribution.

- *m*-**Estimate:** The m-estimate [Ces90] generalises this idea by making the dependency on the prior class distribution explicit, and introducing a parameter m, which allows to trade off the influence of the a priori probability $\Pr(l)$ and $\Pr_{prec}$:

$$\Pr_{m\text{-}estimate}(l|r \supseteq d) = \frac{n_{r,l} + m \cdot \Pr(l)}{n_r + m} \tag{4.10}$$

  The m-parameter may be interpreted as a number of instances that are distributed according to the prior probability, which are added to the class frequencies $n_{r,l}$. The prior probability is typically estimated from the data using $\Pr(l) = {}^{n_l}/_{|D|}$ (but one could, e.g., also use the above-mentioned Laplace-correction if the class distribution is very skewed). Clearly, the Laplace-estimate is a special case of the m-estimate with $m = |L|$ and $\Pr(l) = \frac{1}{|L|}$.

4.4 Shrinkage

Frequently, patterns cover only a small number of instances. This fact can be disadvantageous for the pattern-based probability estimation, as using a small number of instances for the estimation may lead to coarse and uncertain probabilities that need to be smoothed. Shrinkage, which is regularly used in statistical language processing [CG99, MS99], is a general approach for smoothing such probabilities, which has been successfully applied in various research areas. Its key idea is to "shrink" probability estimates towards the estimates of its generalised patterns r_k, which cover more instances. This is quite similar to the idea of the Laplace- and m-estimates, with two main differences. First, shrinkage does not only incorporate the prior probability (which would correspond to a pattern covering all instances) but also interpolates between several different generalisations of a pattern. Second, the weights for the trade-off between these probabilities are not specified a priori (e.g. as it is the case for the m-parameter in the m-estimate), but they are estimated from the data.

In general, shrinkage estimates the probability $\Pr(l|r \supseteq d)$ as follows:

$$\Pr_{shrinkage}(l|r \supseteq d) = \sum_{j=0}^{|r|} w_{j,l} \Pr(l|r_j), \tag{4.11}$$

where $w_{j,l}$ are weights that interpolate between the probability estimates of the generalised patterns r_j. In our implementation, we use only the generalisations of a pattern that can be obtained by deleting a final sequence of conditions.

Thus, for a pattern r of length $|r|$, we obtain $|r|+1$ generalisations r_j, where r_0 is the pattern covering all examples, and $r_{|r|} = r$ is the original one.

For instance, the rule $condition_1 \wedge condition_2 \wedge condition_3 \implies l$ has the following 4 (size + 1) generalisations:

- $r_3 : condition_1 \wedge condition_2 \wedge condition_3 \implies l$
- $r_2 : condition_1 \wedge condition_2 \implies l$
- $r_1 : condition_1 \wedge \implies l$
- $r_0 : true \implies l$

The weights $w_{j,l}$ can be estimated in various ways. We employ a shrinkage method proposed by [WZ06] which is intended for decision tree learning but can be straightforwardly adapted to our task. The authors propose to estimate the weights $w_{j,l}$ with an iterative procedure which averages the probabilities obtained by removing training instances covered by this pattern. In effect, we obtain two probabilities per pattern generalisation and class: the removal of an instance of class l leads to a decreased probability $\Pr_-(l|r_j \supseteq d)$, whereas the removal of an instance of a different class results in an increased probability $\Pr_+(l|r_j \supseteq d)$. Weighting these probabilities with the relative occurrence of training instances belonging to this class, we obtain a smoothed probability

$$\Pr_{Smoothed}(l|r_j \supseteq d) = \frac{n_{r_j,l}}{n_{r_j}} \cdot \Pr_-(l|r_j \supseteq d) + \frac{n_{r_j} - n_{r_j,l}}{n_{r_j}} \cdot \Pr_+(l|r_j \supseteq d) \tag{4.12}$$

Using these smoothed probabilities, this shrinkage method computes the weights of these nodes in linear time (linear in the number of covered instances) by normalising the smoothed probabilities separately for each class.

$$w_{j,l} = \frac{\Pr_{Smoothed}(l|r_j \supseteq d)}{\sum_{i=0}^{|r|} \Pr_{Smoothed}(l|r_i \supseteq d)} \tag{4.13}$$

Multiplying the weights with their corresponding probability, we obtain "shrinked" class probabilities for the instance.

4.5 Probability Aggregation

In the previous two sections, single covering patterns were utilised for the class probability estimation, respectively. We suppose that the probabilities obtained in this way may be improved by the employment of multiple patterns for the class probability estimation, similarly to the predictive improvement of ensemble learning. Thus, we are going to investigate now how the probability estimates of two or more covering patterns may be aggregated into a single probability distribution.

To this end, we assume that there are $|E|$ pattern sets R_i available that were arbitrarily generated (e.g. by an ensemble learning method). For prediction, we determine first the covering patterns of each pattern set $R_i(d)$ for a given instance d.

Let $\text{Cov}_i(d)$ denote the set of all class probability distributions that originate from a pattern in the set of covering patterns

$$\text{Cov}_i(d) = \left\{\vec{\Pr}(L|r \supseteq d)|r \in R_i(d)\right\}. \tag{4.14}$$

From this set of class probability distributions of the covering patterns, we try to estimate a class probability distribution for the given instance d. For this purpose, we have to decode the probability estimations of these covering patterns $\vec{\Pr}(L|r \supseteq d)$ into a single normalised global class probability distribution $\vec{\Pr}_{global}(d)$.

For our approach, we consider four aggregation methods. The first three methods have in common that they average the class probability distributions of (some of) the covering patterns.

- **Best rule:** Only the most confident covering rule $\vec{\Pr}(L|r \supseteq d)$ of each covering rule set $R_i(d)$ is determined.

$$\vec{\Pr}_i = \underset{\vec{\Pr}(L|r \supseteq d) \in \text{Cov}_i}{\arg\max} \left(\vec{\Pr}(L|r \supseteq d)\right) \tag{4.15}$$

  Afterwards, the class probability distributions of these rules are averaged and the result is normalised:

$$\vec{\Pr}_{global}(d) = \frac{\text{avg}\left(\vec{\Pr}_1, \cdots, \vec{\Pr}_{|E|}\right)}{\left\|\text{avg}\left(\vec{\Pr}_1, \cdots, \vec{\Pr}_{|E|}\right)\right\|_1}. \tag{4.16}$$

- **Macro averaging:** All covering rules are determined and their class probability distributions are macro-averaged in two steps. First, the class probability distributions $\text{Cov}_i(d)$ of each covering rule set $R_i(d)$ are averaged and normalised:

$$\vec{\Pr}_i = \frac{\text{avg}\left(\text{Cov}_i(d)\right)}{\|\text{avg}\left(\text{Cov}_i(d)\right)\|_1}. \tag{4.17}$$

  Next, these local class probabilities are averaged as above (see Equation 4.16).

- **Micro averaging:** All covering rules are determined and their class probability distributions are micro-averaged. Essentially, this means that all learned rules are pooled, and the average is formed over the resulting set of rules:

$$\vec{\Pr}_{global}(d) = \frac{\operatorname{avg}\left(\mathrm{Cov}_1(d) \cup \cdots \cup \mathrm{Cov}_{|E|}(d)\right)}{\left\| \operatorname{avg}\left(\mathrm{Cov}_1(d) \cup \cdots \cup \mathrm{Cov}_{|E|}(d)\right) \right\|_1} \tag{4.18}$$

- **Bayesian decoding:** All covering rules are pooled as above (see micro averaging), but their class probability distributions are multiplied with each other and with the vector of the a priori class probabilities (see Equation 3.13)

$$\vec{\Pr}_{prior} = \left(\Pr(l_1), \cdots, \Pr(l_{|L|})\right).$$

  Thus, $\vec{\Pr}_{global}(d)$ is calculated as follows

$$\vec{\Pr}_{global}(d) = \frac{\operatorname{mult}\left(\bigcup_{i=1}^{|E|} \mathrm{Cov}_i(d) \cup \left\{\vec{\Pr}_{prior}\right\}\right)}{\left\| \operatorname{mult}\left(\bigcup_{i=1}^{|E|} \mathrm{Cov}_i(d) \cup \left\{\vec{\Pr}_{prior}\right\}\right) \right\|_1} \tag{4.19}$$

4.6 Experimental Setup

For the evaluation of the performance of the previously introduced probability estimation methods, we perform two experiments within the WEKA framework [WFHP16]. In our first experiment, we investigate the performance of the basic probability estimation techniques and the shrinkage method, and analyse the impact of the pattern discovery algorithm on the probability estimation. In our second experiment, we evaluate the impact of the previously introduced aggregation methods on the probability estimation on the basis of multiple patterns.

The common experimental setup of these experiments will be described in this section. To this end, we summarise first the pattern discovery algorithm that was employed in both experiments to generate the probabilistic patterns. Afterwards, we explain the evaluation of the first and second experiments. The specific configurations and results of these experiments will be discussed in the subsequent sections.

4.6.1 Pattern Discovery

The probability estimation on the basis of local patterns does not impose any obvious requirements on the employed local pattern discovery algorithm. Thus, every local pattern discovery algorithm could be used to generate the utilised patterns. We decided to employ the rule learning algorithm JRip, the Weka [WFHP16] implementation of Ripper (see Section 2.4.5), instead of the algorithms BSD or CHARM, which we used in our previous experiments, for the pattern discovery, since Ripper is a widely known and accurate algorithm. Furthermore, Ripper (and consequently JRip) features two interesting components: its integrated incremental reduced error pruning and its two generation modes, whose effects on the probability estimation we are going to investigate in our experiments, too.

JRip allows to switch its integrated incremental reduced error pruning on or off (see Section 2.4.5). In this way, we are able to test if pruning has the same impact on the probability estimation for pattern discovery as it has on decision tree learning (see Section 4.1). Note, however, that with turned off incremental reduced error pruning, JRip still performs pre-pruning using a minimum description length heuristic. Ripper features two generations modes: the unordered and ordered mode whose effects on the probability estimation will be examined as well. Regrettably, JRip does not support the unordered mode, so we had to add a reimplementation of that mode to it. The Weka framework supports a general one-against-all procedure that can also be combined with JRip. Unfortunately, we could not use this procedure because it does not allow to directly access the pattern statistics that are needed for the probability estimation. We also added a few other minor modifications which were necessary for the probability estimation, e.g. the collection of statistical counts of the patterns.

4.6.2 Evaluation

Both experiments are evaluated on 30 data sets of the UCI repository [Lic13] which differ in the number of attributes (and their categories), classes, and training instances (see Table 4.1). As a performance measure, we use the weighted area under the ROC curve (wAUC), as used for probabilistic decision trees by [PD03]. Its key idea is to extend the binary AUC to the multi-class case by computing a weighted average of the AUCs of the one-against-all problems, where each class l is paired with all other classes:

$$wAUC(c, D) = \sum_{l \in L} \frac{n_l}{|D|} AUC(c, D_l, D \setminus D_l) \tag{4.20}$$

Table 4.1.: Data sets used in the experiments with the number of instances, number of nominal and numeric attributes, and number of classes.

		Attributes		
Data set	Instances	Nominal	Numeric	Classes
Aneal.orig	798	29	9	6
Audiology	226	69	0	24
Autos	205	10	15	7
Balance-scale	625	0	4	3
Breast-cancer	286	9	0	2
Breast-w	699	0	9	2
Colic	368	15	7	2
Credit-a	690	9	6	2
Credit-g	1000	13	7	2
Diabetes	768	0	8	2
Glass	214	0	9	7
Heart-c	303	7	6	5
Heart-h	294	7	6	5
Heart-statlog	270	0	13	2
Hepatitis	155	14	5	2
Hypothyroid	3772	22	7	4
Ionosphere	351	0	34	2
Iris	150	0	4	3
Kr-v-kp	3196	37	0	2
Labor	57	8	8	2
Lymph	148	15	3	4
Primary-tumor	339	17	0	22
Segment	2310	0	19	7
Sick	3772	22	7	2
Sonar	208	0	60	2
Soybean	683	35	0	19
Vehicle	846	0	18	4
Vote	435	16	0	2
Vowel	990	3	10	11
Zoo	101	15	1	7

For the evaluation of the results, we either use the Friedman test with a post hoc Nemenyi test for the comparison of multiple methods or the Wilcoxon signed ranks test for the comparison of two methods as proposed in [Dem06] (see Section 2.4.4). The significance level was set to 5% for all tests.

4.7 Experiment 1: Probability Estimation

In our first experiment on probability estimation, we investigate the probability estimation for single local patterns using the previously introduced basic probability estimation methods and shrinkage. We evaluate the performance of these methods and the influence of the employed local pattern discovery algorithm on the probability estimation. To this end, we consider four question:

- **Basic probability estimation:** Which basic probability estimation method exhibits the best performance?

- **Shrinkage:** Does the introduced shrinkage method improve the performance of the basic probability estimation methods?
- **Pattern discovery - ordered vs. unordered generation:** What is the influence of the local pattern discovery in respect to the ordered and unordered generation mode?
- **Pattern discovery - pruning** What is the impact of pruning on the performance of the probability estimation?

To answer these questions, we use the following experimental setup. We pair each of the four configurations of our extended JRip (using its unordered or ordered mode and switching pruning on or off) with the 5 different basic probability estimation methods: precision, the Laplace-estimate, and the m-estimate with $m \in \{2, 5, 10\}$ (abbreviated P, L, M=2, M=5, and M=10, respectively). All basic probability estimation methods are both used stand-alone as a basic probability estimate (abbreviated with B) or in combination with the described shrinkage method (abbreviated with S). As a baseline, we also include the performance of the pruned and unpruned regular JRip, accordingly. Our unordered implementation of JRip that uses the Laplace-estimate stand-alone for the probability estimation is comparable to the unordered version of Ripper, which is, as mentioned before, not implemented in JRip.

In the test phase, all covering patterns are selected for a given test instance. Using this reduced pattern set, we determine the pattern whose prediction is most probable. For this purpose, we select the most probable class of each pattern and use this class value as the prediction for the given test instance and the class probability for comparison. Ties are solved by predicting the least represented class. If no covering rules exist, the class probability distribution of the default rule is used. We only discuss summarised results in this section, detailed results may be found in Appendix B.

4.7.1 Basic Probability Estimation

In our first evaluation, we compare the performance of the basic probability estimation methods. To this end, we calculate their average weighted AUC per data set (see Table 4.2), considering only the experimental configurations that employ the basic probability estimation methods stand-alone. The Friedman test shows significant differences in performance for these methods. Hence, we employ a post hoc Nemenyi test whose results are depicted in Figure 4.1. Regarding this critical distance chart, three groups of methods whose performances do not differ significantly may be identified.

The last and worst group consists of the original JRip algorithm, which ranks last, and precision, which performs insignificantly better than the former. The

Table 4.2.: Basic probability estimation: average weighted AUC of each basic probability estimation method, including their average rank.

Data Set	JRip	P	L	M=2	M=5	M=10
Anneal.orig	.9316	.9581	.9577	.9597	.9591	.9578
Audiology	.8850	.8831	.8526	.8739	.8676	.8600
Autos	.8771	.8863	.8846	.8891	.8869	.8864
Balance-scale	.8378	.8438	.8606	.8595	.8613	.8616
Breast-cancer	.5942	.5856	.5909	.5909	.5912	.5918
Breast-w	.9505	.9540	.9538	.9537	.9535	.9526
Colic	.7794	.7891	.8013	.8013	.8043	.8087
Credit-a	.8579	.8717	.8811	.8811	.8818	.8825
Credit-g	.5887	.6495	.6546	.6547	.6549	.6553
Diabetes	.6903	.7219	.7286	.7287	.7285	.7278
Glass	.8043	.8114	.8102	.8104	.8114	.8121
Heart-c	.7965	.8047	.8317	.8320	.8325	.8335
Heart-h	.7429	.7509	.7710	.7719	.7711	.7716
Heart-statlog	.7715	.7861	.8142	.8135	.8154	.8153
Hepatitis	.6714	.7001	.7070	.7081	.7062	.7069
Hypothyroid	.9795	.9844	.9875	.9863	.9876	.9873
Ionosphere	.8918	.9063	.9172	.9178	.9190	.9195
Iris	.9650	.9300	.9332	.9332	.9332	.9331
Kr-vs-kp	.9940	.9963	.9968	.9968	.9967	.9965
Labor	.7956	.8095	.7921	.7909	.7875	.7814
Lymph	.7723	.8088	.8192	.8192	.8137	.8152
Primary-tumor	.6455	.6618	.6466	.6692	.6681	.6678
Segment	.9853	.9747	.9780	.9779	.9780	.9778
Sick	.9351	.9458	.9520	.9523	.9524	.9522
Sonar	.7664	.7903	.7984	.7985	.7985	.7980
Soybean	.9719	.9799	.9759	.9814	.9781	.9774
Vehicle	.8136	.8463	.8591	.8591	.8588	.8588
Vote	.9467	.9555	.9616	.9595	.9616	.9613
Vowel	.8968	.9109	.9132	.9152	.9157	.9135
Zoo	.9201	.9313	.9286	.9436	.9433	.9407
Average Rank	5.200	4.400	3.350	2.600	2.483	3.000

second group comprises precision and the Laplace-estimate. So, the Laplace-estimate offers a slight but not significant improvement over precision. The first and best group consists of the Laplace-estimate and all configurations of the m-Estimate. Since the m-Estimate - independent of the chosen m-parameter - ranks before all other methods, we conclude that the m-Estimate is the preferable option amongst the considered probability estimation methods. The m-Estimate with $m = 5$ ranks before the other two configurations of the m-Estimate, but this difference is insignificant. Thus, the results we obtain are similar to those of the probability estimation on the basis of probability estimation trees (see Section 4.1).

4.7.2 Shrinkage

In our second evaluation, we investigate the impact of shrinkage on the probability estimation. To this end, we compare the average weighted AUC per data

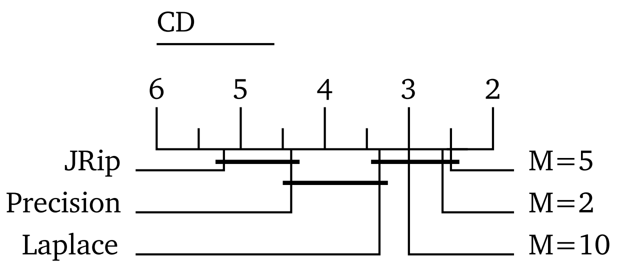


Figure 4.1.: Basic probability estimation: critical distance chart.

set over all configurations that employ shrinkage to the similarly computed weighted AUC that uses the basic probability estimation methods stand-alone (see Table 4.3). The Wilcoxon signed ranks test shows for this comparison no significant differences in performance. Shrinkage which worsens the performance slightly is still comparable to the basic probability estimation without shrinkage in general. Hence, we consider the performance of shrinkage for each basic probability estimation method separately using the comparison approach as described before (see Table C.1 and C.2 in Appendix C).

For precision and the m-Estimates with $m = 5$ and $m = 10$, the Wilcoxon signed ranks test does not show significant differences in performance, too. For the m-Estimate with $m = 2$ and the Laplace-estimate, the Wilcoxon signed ranks test rejects the null hypothesis that the probability estimation with and without shrinkage performs equally. However, shrinkage worsens the performance of the probability estimation in these two cases. In summary, shrinkage does not yield any significant increases in performance for the probability estimation, independent of the employed basic probability estimation method. Thus, we do not consider any experimental configurations that employ shrinkage in the following evaluations of the first experiment and ignore it completely in our second experiment. Nevertheless, the basic idea of shrinkage could still offer potential performance increases as alternative approaches have not been examined or developed, yet.

4.7.3 Pattern discovery - Ordered vs. Unordered Generation

In the third evaluation, we compare the performance of the ordered and unordered pattern discovery modes. For this purpose, we calculate the average performance for each of these two modes separately, as described above, considering only the configurations that employ the basic probability estimation methods stand-alone. The Wilcoxon signed ranks test shows that there is a significant difference in performance for ordered and unordered pattern sets. The latter performs significantly better than the former. This result conforms

Table 4.3.: Shrinkage: average weighted AUC of the probability estimation with and without shrinkage (stand-alone), including the sign, absolute value, and the rank of the difference.

			Difference		
Data Set	Stand-alone	Shrinkage	Sign	Abs.	Rank
Anneal.orig	.9585	.9553	+	.0032	19
Audiology	.8674	.8579	+	.0095	7
Autos	.8867	.8832	+	.0035	15
Balance-scale	.8574	.8391	+	.0183	1
Breast-cancer	.5901	.5837	+	.0064	11
Breast-w	.9535	.9569	-	.0033	16
Colic	.8009	.8101	-	.0092	8
Credit-a	.8796	.8812	-	.0015	25
Credit-g	.6538	.6566	-	.0028	21
Diabetes	.7271	.7253	+	.0018	24
Glass	.8111	.8155	-	.0044	13
Heart-c	.8269	.8096	+	.0173	2
Heart-h	.7673	.7655	+	.0018	23
Heart-statlog	.8089	.7979	+	.0111	5
Hepatitis	.7057	.7166	-	.0109	6
Hypothyroid	.9866	.9876	-	.0010	27
Ionosphere	.9160	.9150	+	.0009	28
Iris	.9325	.9288	+	.0037	14
Kr-vs-kp	.9966	.9951	+	.0015	26
Labor	.7923	.7915	+	.0008	29
Lymph	.8152	.8040	+	.0112	4
Primary-tumor	.6627	.6714	-	.0087	9
Segment	.9773	.9638	+	.0135	3
Sick	.9509	.9487	+	.0022	22
Sonar	.7967	.7902	+	.0065	10
Soybean	.9785	.9753	+	.0033	18
Vehicle	.8564	.8570	-	.0006	30
Vote	.9599	.9566	+	.0033	17
Vowel	.9137	.9089	+	.0048	12
Zoo	.9375	.9405	-	.0030	20

with our expectation that the unordered generation mode of JRip should discover patterns that are more suitable for the probability estimation than those patterns that are generated by the ordered generation mode. Optimally, a probabilistic pattern should feature a preferably high but accurate (estimated) probability for the most probable class which is predicted for this reason. The pattern discovery process should be aimed at the generation of patterns whose probability estimates on the basis of the covered instances satisfy this assumption. Additionally, each class should be represented by a sufficient high number of accurate patterns in the utilised pattern set to guarantee the diversity of the probability estimation.

In regard to these assumptions, the two generation modes differ. First, the probability estimation is based on all instances that are covered by the considered pattern. In contrast to this, the pattern generation considers only the instances that have not been covered by previous patterns (in the decision list)

Table 4.4.: Ordered and unordered pattern discovery: average weighted AUC of the probability estimation using ordered and unordered pattern discovery, including the sign, absolute value, and the rank of the difference.

			Difference		
Data Set	Ordered	Unordered	Sign	Abs.	Rank
Anneal.orig	.9278	.9860	-	.0582	11
Audiology	.8340	.8913	-	.0573	12
Autos	.8643	.9055	-	.0412	15
Balance-scale	.8136	.8829	-	.0693	6
Breast-cancer	.5882	.5855	+	.0026	30
Breast-w	.9463	.9641	-	.0179	23
Colic	.7783	.8327	-	.0544	13
Credit-a	.8666	.8942	-	.0277	18
Credit-g	.5997	.7107	-	.1110	2
Diabetes	.6951	.7573	-	.0621	9
Glass	.8060	.8206	-	.0146	26
Heart-c	.8083	.8282	-	.0199	22
Heart-h	.7480	.7849	-	.0369	16
Heart-statlog	.7871	.8197	-	.0326	17
Hepatitis	.6290	.7932	-	.1643	1
Hypothyroid	.9819	.9923	-	.0103	28
Ionosphere	.9039	.9270	-	.0231	19
Iris	.8868	.9746	-	.0878	4
Kr-vs-kp	.9941	.9976	-	.0035	29
Labor	.7810	.8028	-	.0218	20
Lymph	.7624	.8568	-	.0945	3
Primary-tumor	.6349	.6992	-	.0643	8
Segment	.9487	.9923	-	.0436	14
Sick	.9392	.9604	-	.0212	21
Sonar	.7604	.8266	-	.0662	7
Soybean	.9704	.9834	-	.0131	27
Vehicle	.8273	.8862	-	.0589	10
Vote	.9504	.9662	-	.0158	25
Vowel	.9026	.9201	-	.0175	24
Zoo	.8984	.9796	-	.0812	5

in both generation modes. Thus, the statistical information that is used for the generation of a pattern differs usually from the statistical information that is used for the probability estimation. This difference is less severe in the unordered generation mode as all previously covering patterns predict the same class (in contrast to this, previously covering patterns may predict different classes in the ordered generation mode).

Second, the two generation modes treat the most represented class differently. The ordered generation mode does not induce patterns for this class (except for a default pattern). However, the unordered generation mode aims to generate patterns for each class, allowing more diverse patterns for the biggest class. Hence, the probability estimation for this class will in the most cases be more coarse in the ordered generation mode than in the unordered one. Considering the two aforementioned differences, we conclude that the unordered

Table 4.5.: Pruning: average weighted AUC of the probability estimation with or without pruning, including the sign, absolute value, and the rank of the difference.

			Difference		
Data Set	Pruned	Unpruned	Sign	Abs.	Rank
Anneal.orig	.9618	.9551	+	.0067	17
Audiology	.8670	.8679	-	.0009	28
Autos	.8584	.9149	-	.0566	2
Balance-scale	.8556	.8591	-	.0035	24
Breast-cancer	.5776	.6025	-	.0249	4
Breast-w	.9630	.9441	+	.0189	9
Colic	.8032	.7987	+	.0044	22
Credit-a	.8759	.8833	-	.0074	15
Credit-g	.6582	.6494	+	.0088	13
Diabetes	.7373	.7169	+	.0204	7
Glass	.8207	.8015	+	.0192	8
Heart-c	.8354	.8183	+	.0171	10
Heart-h	.7383	.7963	-	.0580	1
Heart-statlog	.8038	.8141	-	.0103	12
Hepatitis	.6833	.7280	-	.0448	3
Hypothyroid	.9895	.9837	+	.0058	18
Ionosphere	.9154	.9166	-	.0012	27
Iris	.9325	.9326	-	.0000	30
Kr-vs-kp	.9965	.9968	-	.0003	29
Labor	.7945	.7900	+	.0045	21
Lymph	.8192	.8112	+	.0080	14
Primary-tumor	.6643	.6610	+	.0033	25
Segment	.9736	.9809	-	.0073	16
Sick	.9490	.9529	-	.0039	23
Sonar	.7845	.8090	-	.0245	5
Soybean	.9811	.9760	+	.0051	19
Vehicle	.8680	.8449	+	.0230	6
Vote	.9523	.9674	-	.0151	11
Vowel	.9123	.9151	-	.0027	26
Zoo	.9350	.9400	-	.0050	20

generation mode should be more suitable for the probability estimation than the ordered one.

4.7.4 Pattern discovery - Pruning

In the fourth and last evaluation of our first experiment, we analyse the impact of pruning on the performance of the probability estimation. To this end, we consider only the configurations that employ the basic probability estimation stand-alone and the unordered generation mode. First, we evaluate the performance of the pruned and unpruned pattern sets over all the aforementioned configurations (see Table 4.5). The Wilcoxon signed ranks test does not show any significant differences between these two approaches. Hence, we repeat the evaluation for each basic probability estimation method separately (see Tables C.3 and C.4 in Appendix C). The associated Wilcoxon signed ranks tests shows mixed results for the four basic probability estimation methods. For the Laplace-estimate and the three instantiations of the m-estimate, the unpruned pattern sets perform better than the pruned ones, but this difference is only significant for the Laplace-estimate. Only for precision, the opposite is true, since the pruned pattern sets perform significantly better than the unpruned ones.

The integrated pruning method of Ripper takes place in the context of a decision list. Thus, the determination to prune or re-learn a pattern does not aim to increase the individual quality of the respective pattern but the quality of the resulting decision list. In the case of pruning, a more general pattern is obtained that covers more instances (that may belong to any class). The estimated probability distribution may be more certain as more evidence (in the form of an increased number of instances) is available. However, the probability of the most probable class may be decreased in this way. Consequently, a pruned pattern may be used less often in our classification scheme since only the most probable pattern is used for prediction. The results of re-learning a pattern is even more unpredictable in terms of the probability estimation. Apparently, the disadvantages of pruning seem to prevail the advantages of pruning in the context of probability estimation. The only exception to this observation is the probability estimation using precision. In this case, pruning offers an insignificant increase of performance. This may be attributed to the fact that precision is susceptible to overfitting which is reduced by pruning in general. In summary, we assume that the unpruned pattern discovery is in general the better choice.

4.8 Experiment 2: Probability Aggregation

In our second experiment on probability estimation, we investigate the aggregation of multiple probability estimations into a single one, using the previously described probability aggregation methods. For this purpose, we consider two questions:

- **Probability aggregation - number of covering patterns:** What is the influence of the number of covering patterns on the probability estimation?

- **Probability aggregation - comparison of probability aggregation methods:** Which probability aggregation method offers the best performance?

To answer these questions properly, multiple covering patterns for each data instance are necessary. As the utilised pattern discovery algorithm Ripper does not guarantee multiple covering patterns, we decide to employ an ensemble of classifiers. Consequently, each data instance is covered by at least one pattern per ensemble classifier. In this way, we are able to adjust the minimum number of covering patterns by changing the ensemble size, and obtain multiple covering patterns whose basic probability estimations may be aggregated by the considered methods.

For the configuration of this approach, we take the results of our first experiments into account. In accordance with these results, we select the m-Estimate with $m = 5$, which dominates the group of best performing probability estimation methods, for the probability estimation. As the employed shrinkage method worsens (insignificantly) the probability estimation, we abstain from using shrinkage in these experiments. Furthermore, the local pattern discovery algorithm JRip is adjusted to employ the unordered generation mode and omit pruning, discovering now only unordered unpruned pattern sets. The base classifier we obtain in this way is combined with the ensemble method bagging. The main motivation for the employment of bagging is that we can easily adjust the minimum number of covering patterns by changing the number of folds. Hence, we considered bagging with 10, 20, 50, or 100 samples, respectively. For the aggregation of multiple probability estimations, all previously mentioned aggregation methods best pattern, macro and micro averaging, and Bayesian decoding (abbreviated BR, Mac, Mic and BD, respectively) are applied to each classifier ensemble. These methods are evaluated mutually and compared to a bagged version of JRip that uses an analogous configuration (unordered generation mode without pruning) and the same bootstrap samples.

In all cases, the resulting class probability distribution $\vec{\Pr}_{global}(d)$ is used for the prediction. For this purpose, the most probable class, according to $\vec{\Pr}_{global}(d)$, is selected as the prediction for the given test instance d. Ties are solved by predicting the least represented class. If no covering patterns ($R_i(d) = \emptyset$) exist for a sampled pattern set R_i, the class probability distribution of the default pattern is used accordingly. We only discuss summarised results in this section, detailed results may be found in Appendix D.

4.8.1 Probability Aggregation - Number of Covering Patterns

In our first evaluation, we examined the general advantage of a higher number of patterns. For this purpose, we compared the average performance (over all data sets) of the four decoding methods mutually and to the average performance of the m-estimate in our first experiments that employed only the probability estimate of a single pattern. The results of this comparison are represented in Table 4.6. For each number of folds, the regular bagged JRip and each aggregation method performed better than the m-estimate in our first experiments where no aggregation methods were employed. So, one may con-

Table 4.6.: Probability aggregation: average weighted AUC of each aggregation method (per number of folds using the m-estimate with $m = 5$) and a default bagged JRip (added for comparison).

	Number of Folds			
Aggregation Method	10	20	50	100
No aggregation (1st experiments)	.8864			
Bagged JRip	.9042	.9103	.9148	.9167
Bayesian Decoding	.9233	.9266	.9287	.9295
Best Rule	.9217	.9256	.9285	.9291
Macro Averaging	.9239	.9268	.9293	.9299
Micro Averaging	.9248	.9276	.9295	.9302

clude that a higher number of patterns is clearly beneficial for the probability estimation task. Furthermore, an increase in the number of folds also raised the average performance of all aggregation methods.

Obviously, a higher number of patterns is beneficial independent of the employed decoding method as the performance of the probability estimation increases accordingly. However, the performance increase diminishes with a higher number of patterns. A possible explanation for this observation is that at the beginning (new) high-quality patterns may be found more easily, but later the chance to find less useful patterns, which either are redundant or less predictive, increases. Clearly, such patterns do not contribute to the overall probability meaningfully. On the one hand, the aggregation method best rule will not profit from these less accurate patterns as the most predictive pattern will not be replaced by these. On the other hand, redundant patterns may distort the probability aggregations of the other three methods, macro and micro averaging and Bayesian decoding, because they are considered several times in their calculation.

In summary, the utilisation of an increasing number of patterns has a positive effect on the probability estimation, but the gain diminishes slightly with an increasing number of patterns.

4.8.2 Probability Aggregation - Comparison of Probability Aggregation Methods

In our second evaluation, we compare the considered probability aggregation methods. As a starting point, Table 4.6 provides a first indication of their performance. As one can see they are ranked in a descending order of performance as follows: micro averaging, macro averaging, Bayesian decoding, and best rule. Next, we examine if this observation may be supported by a statistical test. To this end, we computed a Friedman test for each fold size separately. For 10 folds, the Friedman test showed significant differences between the em-

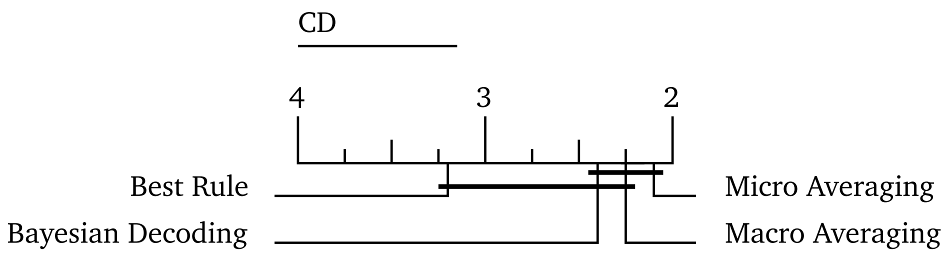


Figure 4.2.: Probability aggregation: critical distance chart for 10 folds.

ployed aggregation methods. The results of the subsequent Nemenyi test are depicted in Figure 4.2. Two groups of comparable methods could be identified. The worst group consists of best rule, Bayesian decoding and macro averaging. The first group comprises Bayesian decoding, macro averaging and micro averaging. Since micro averaging is the only member of the best group that does not also belong to the worst group of methods, we consider it as the best choice of probability aggregation in this scenario. For the other fold sizes, the Friedman test could not detect any significant differences between the considered aggregation methods. Nevertheless, the ranking of these methods (for every other fold size) was equal to the ranking, that we observed for 10 folds. For a lower number of folds, the calculation of average probability distributions by macro and micro averaging seems to profit from hi-quality patterns. With an increasing number of folds, the performance of the worst aggregation method best rule approaches to the performance of the other methods. Presumably, the high quality patterns become lost in an increasing number of (low quality) patterns with an increasing of number folds. Hence, considering only the best rule converges to averaging the probability distributions of all covering patterns. This observation confirms our assumption that at first high quality patterns are found and later the usefulness of the discovered patterns declines.

Nevertheless, we argue that micro averaging should be chosen for the probability aggregation as it ranked before the other three methods for all fold sizes (even if the difference in performance is insignificant).

4.9 Summary

In this chapter, we investigated the question, how class probabilities may be estimated as accurately as possible, using a given set of local patterns. This problem may be divided into two associated tasks: the probability estimation for a single pattern, and the probability aggregation for multiple patterns. For these tasks, we considered the following aspects:

- **Basic probability estimation:** For the basic probability estimation, we compared the performances of the methods m-Estimate (with m=2, 5, or 10), Laplace-estimate and precision mutually and to the performance of the original JRip algorithm. All methods performed better than JRip. The best performing methods were the three configurations of the m-Estimate. Among these configurations, the m-Estimate with m=5 ranked before the two others. However, the difference in performance is insignificant in this case.

- **Shrinkage:** Furthermore, we investigated the impact of shrinkage on the probability estimation. To this end, we adapted a shrinkage algorithm to the probability estimation for patterns whose key idea is to improve the probability estimation of the basic methods by considering the probability estimations of selected sub patterns. Unfortunately, the evaluation of this shrinkage approach showed that it performed worse than the probability estimation that employs the basic probability estimation methods stand-alone. In the case of the m-Estimate with $m = 2$ and the Laplace-estimate, this difference in performance was even significant. So, we advise against the employment of this specific shrinkage approach. However, other approaches could yield better results.

- **Pattern discovery:** Additionally, we examined the impact of the pattern discovery on the probability estimation. In our experiments, we employed a modified JRip, an reimplementation of the Ripper algorithm, that offered two configuration options. The first option allowed us to either utilise the ordered and unordered generation mode of JRip. The pattern sets discovered by the unordered generation mode performed significantly better than the sets obtained by the ordered generation mode. JRip's second option enabled us to switch the integrated reduced error pruning on or off. Unpruned pattern sets performed insignificantly better than the pruned ones with the exception of the Laplace-estimate where the difference in performance was even significant and precision where the pruned pattern sets performed insignificantly better.

- **Probability aggregation:** In our seconds experiments, we examined if the aggregation of several probability estimations into a single one is worthwhile and how this aggregation may be achieved. For this purpose, we employed the common aggregation methods macro and micro averaging, Bayesian decoding and best rule. We observed that the employment of several estimated probability distributions is clearly advantageous for all of the four methods. Amongst the considered methods, micro averaging stood out as it performed better than the other three methods in most of the cases. But its performance was not always significantly better than the performances of the other methods.

In summary, we advise to employ the basic probability estimation method *m*-Estimate on unpruned unordered pattern sets for the probability estimation. If multiple patterns are used for the probability estimation, their estimated probability distributions should be aggregated by micro averaging.

5 Theory Compression

In this chapter, we investigate the question, how a set of local patterns may be transformed into a compact and understandable model. Pattern discovery methods may generate sets of local patterns (either independent ones or as parts of a global model) that often may be hard to interpret. This is even more true if multiple local pattern sets have been generated (e.g. by an ensemble learning method). Furthermore, additional means (e.g. the utilisation of voting methods) may be necessary to obtain predictions with the help of the (covering) patterns. Both problems, the low interpretability of local pattern sets and the necessity of a decoding approach, may be solved by the generation of a global model on the basis of the local pattern set(s).

To this end, we propose a novel approach called rule stacking which adapts the general meta learning method stacking (see Section 2.4.3.1) to the global modelling of rule-based local patterns. Rule stacking and the original stacking differ in three decisive aspects. The first aspect is that the meta data generation of the two approaches occurs on different levels of granularity. While stacking encodes the training instances on the basis of the individual predictions of multiple classifiers, rule stacking takes the covering information of each available pattern into account. For each instance, the information about which of these patterns cover it or not is stored. The second aspect is that rule stacking features an additional retransformation step. In this step, the generated meta model is retransformed into a compressed global model which is directly applicable to instances of the original data space (that means that instances do not have to be translated into meta instances in the prediction phase). For this reason, the obtained global model may also be more easily interpreted. The last aspect is that rule stacking may be applied to one or more arbitrary local pattern sets that do not have to be generated necessarily by an ensemble of learning algorithms. Thus, we will explain our approach in the general case that one or more local pattern sets are available for processing.

This chapter is organised as follows. We explain our motivation for our rule stacking approach and its differences to stacking in Section 5.1. Next, we introduce several notations in Section 5.2 that are necessary to encode the covering information for our meta data generation approach in Section 5.3. Thereafter, we show how rule stacking retransforms the meta model (and in this manner the utilised local pattern sets) into a global classifier in the original data format in Section 5.4, focusing on the retransformation step of our approach. In the

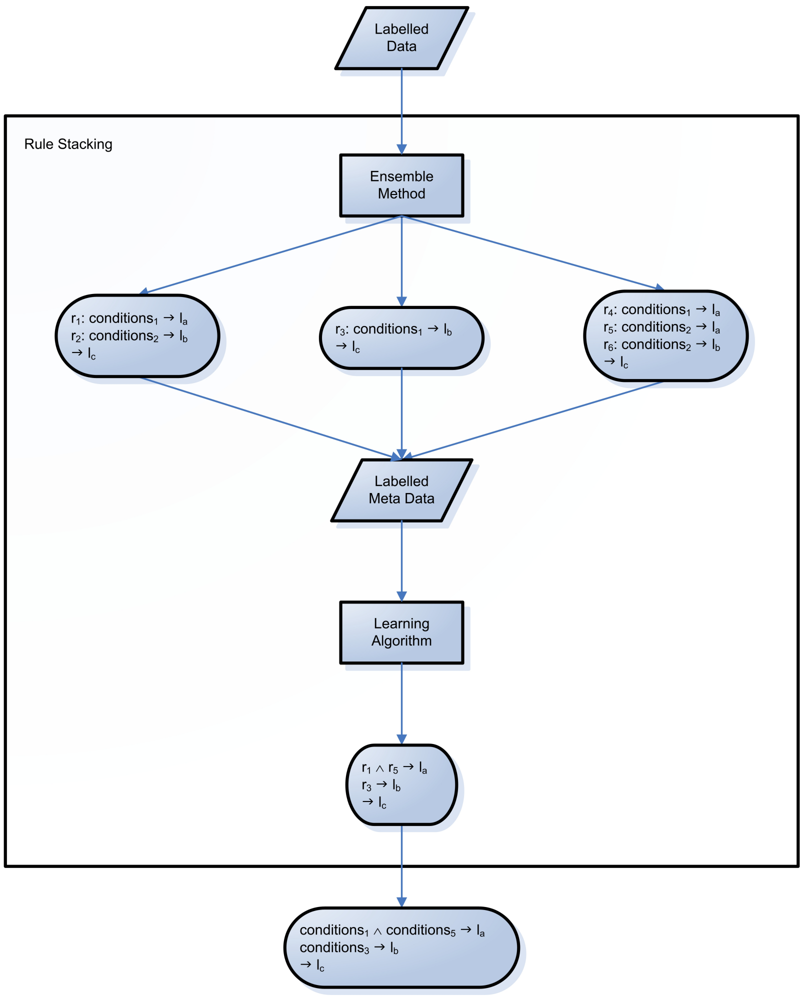

Labelled Data
Rule Stacking
Ensemble Method
r_1: conditions$_1$ → l_a
r_2: conditions$_2$ → l_b
→ l_c
r_3: conditions$_1$ → l_b
→ l_c
r_4: conditions$_1$ → l_a
r_5: conditions$_2$ → l_a
r_6: conditions$_2$ → l_b
→ l_c
Labelled Meta Data
Learning Algorithm
$r_1 \wedge r_5$ → l_a
r_3 → l_b
→ l_c
conditions$_1$ ∧ conditions$_5$ → l_a
conditions$_3$ → l_b
→ l_c

Figure 5.1.: Schematic illustration of the life cycle of a rule stacking scheme, which uses an ensemble method for the pattern generation.

subsequent Sections 5.5 and 5.6, we describe the setup and the results of our experiments, which are concluded in the summary in Section 5.7.

5.1 Motivation

Pattern discovery algorithms usually generate large sets of local patterns which are chosen for their individual quality. With the large number of patterns come some obvious disadvantages (see Section 2.5.3). Clearly, the interpretation of local pattern sets as a whole may not be feasible. Even though individual patterns may be more easily interpreted, the interesting ones have to be found in the large redundant local pattern sets. Additionally, further efforts are needed to obtain global predictions for future data. The most common solutions are voting methods (see Section 2.4.3.3), which use the prediction of each base pattern as a (weighted) vote, or the similarity to prediction vectors for the individual classes (e.g. as used in error correcting output codes [DB95]). The size of the local pattern sets is a bottleneck for the overall prediction time, since independent of the employed decoding method, several or all members of the pattern set are needed for the prediction. Finally, the resulting prediction is also harder to explain and justify, which is particularly crucial for rule-based classifiers.

The aforementioned problems may be solved if the local pattern set is transformed or better is compressed into a compact and understandable global model. A straightforward solution to this approach is the employment of the covering strategy of separate-and-conquer rule learning (see Section 2.4.2). Thus, the best patterns in the local pattern set are selected and added to a decision list one after another, afterwards the covered instances are removed from the training data. In this way, we obtain a model that is clearly more interpretable than the complete local pattern set. However, it is not guaranteed that the (potential) predictive accuracy of the set is maintained, since only a subset of the patterns is used and no new patterns are created.

Alternatively, each member of the local pattern set may be treated as an individual (but incomplete) local model. Hence, several models have to be transformed into a single one essentially. This task may be handled by the ensemble learning method stacking (see Section 2.4.3). Unfortunately, this approach only partially solves our problem, because the model at the meta level involves the predictions of the base level, which is still a problem for both efficiency and comprehensibility. Consequently, we propose rule stacking as an alternative approach (see Figure 5.1) that adapts the standard stacking method to our purposes.

This adoption differs from stacking in two major aspects. The first aspect is the different meta data generation approach. In contrast to stacking, where each meta instance encodes the prediction of each classifier of the ensemble for a respective base level instance, rule stacking determines for each base level instance by which patterns of the local pattern set(s) it is covered and stores this information in one meta attribute for each pattern, respectively. Our motivation for this approach is the assumption that the covering information yields more information than the prediction alone. Obviously, knowing which patterns cover an instance is more informative than simply knowing the predicted class label, and implicitly also captures the predicted confidence or class probability distribution, which, in a pattern-based classifier, are determined by the quality of the patterns that cover an instance.

The second aspect is the additional retransformation step. Stacking generates a meta model which needs instances in the meta level format. To obtain such meta instances, the predictions of all base level classifiers are necessary which have to be stored for later use for this reason. The main idea of the retransformation step is to avoid the transformation of instances (of the original data space) into meta instances in the prediction phase (including the utilisation and storage of the base level models). To this end, we retransform the generated meta model into a global model that may be applied to instances in the original format, since its patterns consist only of a conjunction of the original patterns. In this way, the obtained global model is more compact and interpretable than the local pattern sets or the set of models (the meta model and the base level models).

In the following sections, we will concentrate on the two above-mentioned aspects, since the other two steps: the pattern discovery and the generation of the meta model may be executed arbitrarily (for instance, analogously to the general stacking approach). To this end, we show how the meta data is generated in Section 5.3 and how the resulting meta model can be transformed into a global classifier which can be directly applied to the original instances in Section 5.4.

5.2 Encoding the Covering Information

Before we may explain our meta data generation approach, we have to introduce three notations that are needed to encode the covering information for this purpose. Given a pattern $r \in R$ and an instance $d \in D$, the first function covers(r, d) determines whether this pattern covers the considered instance or not:

$$\text{covers}(r, d) = \begin{cases} \textsf{true}, & \text{if } r \supseteq d \\ \textsf{false}, & \text{otherwise.} \end{cases} \tag{5.1}$$

If we assume that the local patterns are members of one or more decision lists $(R_1, R_2, \cdots, R_{|E|})$, two additional functions are feasible. Given a decision list R and an instance d, the function first(c, d) determines the first covering pattern in the considered decision list:

$$\text{first}(R, d) = \min_{\triangleleft}(R(d)) \in R \tag{5.2}$$

Finally, the function index(c, d) determines the index of the first covering pattern in the decision list R and a given instance d:

$$\text{index}(R, d) = \arg\min_{k}(r_k \in R(d)) \in \{1, 2, \cdots, |R(d)|\} \tag{5.3}$$

Please note that the functions first and index are used to store the first covering pattern as a nominal or numerical value, respectively. Now, we have all functions at hand that are needed for the definition of the meta data in the following section.

$r_1^{c_1}$	$\cdots$	$r_{n_1}^{c_1}$	$\cdots$	$r_{n_{\vert E\vert}}^{c_{\vert E\vert}}$	L
$\text{covers}(r_1^{c_1}, d_1)$	$\cdots$	$\text{covers}(r_{n_1}^{c_1}, d_1)$	$\cdots$	$\text{covers}(r_{n_{\vert E\vert}}^{c_{\vert E\vert}}, d_1)$	l_1
$\text{covers}(r_1^{c_1}, d_2)$	$\cdots$	$\text{covers}(r_{n_1}^{c_1}, d_2)$	$\cdots$	$\text{covers}(r_{n_{\vert E\vert}}^{c_{\vert E\vert}}, d_2)$	l_2
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
$\text{covers}(r_1^{c_1}, d_{\vert D\vert})$	$\cdots$	$\text{covers}(r_{n_1}^{c_1}, d_{\vert D\vert})$	$\cdots$	$\text{covers}(r_{n_{\vert E\vert}}^{c_{\vert E\vert}}, d_{\vert D\vert})$	$l_{\vert D\vert}$

(a) using binary features

c_1	c_2	$\cdots$	$c_{\vert E\vert}$	L
$\text{first}(c_1, d_1)$	$\text{first}(c_2, d_1)$	$\cdots$	$\text{first}(c_{\vert E\vert}, d_1)$	l_1
$\text{first}(c_1, d_2)$	$\text{first}(c_2, d_2)$	$\cdots$	$\text{first}(c_{\vert E\vert}, d_2)$	l_2
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
$\text{first}(c_1, d_{\vert D\vert})$	$\text{first}(c_2, d_{\vert D\vert})$	$\cdots$	$\text{first}(c_{\vert E\vert}, d_{\vert D\vert})$	$l_{\vert D\vert}$

(b) using nominal features

c_1	c_2	$\cdots$	$c_{\vert E\vert}$	L
$\text{index}(c_1, d_1)$	$\text{index}(c_2, d_1)$	$\cdots$	$\text{index}(c_{\vert E\vert}, d_1)$	l_1
$\text{index}(c_1, d_2)$	$\text{index}(c_2, d_2)$	$\cdots$	$\text{index}(c_{\vert E\vert}, d_2)$	l_2
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
$\text{index}(c_1, d_{\vert D\vert})$	$\text{index}(c_2, d_{\vert D\vert})$	$\cdots$	$\text{index}(c_{\vert E\vert}, d_{\vert D\vert})$	$l_{\vert D\vert}$

(c) using numerical features

Figure 5.2.: Illustration of the modified meta data variants.

5.3 Generating the Meta Data

The first major difference between the standard stacking scheme and our rule stacking approach is the generation of the meta data. As mentioned before, pattern discovery methods provide more information than just predictions, namely the information which patterns cover a given test instance and which do not. Since we assume that this additional covering information may turn out to be advantageous for the model induction, we want to exploit this information. So, the attributes of our meta data set are not the predictions of base learners but either the information if a specific pattern covers the instance or the information which is the first pattern of a given decision list that covers the instance. This information can, essentially, be encoded at the meta level in three different ways. These three variants assume that one or more local pattern sets are available for the meta data generation. While the first variant is suitable for arbitrarily generated local pattern sets in principle, the two others may only be applied to local pattern sets that are organised in a decision list. Furthermore, the last two variants are only reasonable for multiple local pattern sets.

Binary Features: For each pattern in the local pattern set(s), we create a binary attribute with boolean values. The attribute value is true if the corresponding pattern covers the instance, else it is false. Thus, a meta instance is composed of a number of boolean values and, if known, its original class label.

Meta instance format (for a single local pattern set R): (cf. Figure 5.2 (a))

$$\left(\text{covers}(r_1, d), \text{covers}(r_2, d), \cdots, \text{covers}(r_{|R|}, d), l\right)$$

Nominal Features: For each decision list R, we create a nominal attribute whose domain consists of the identifiers of its members $(r_1, r_2, \cdots, r_{|R|})$ used as nominal values. A meta instance is composed of the respective first covering pattern of each decision list R_i and if known its original class value.

Meta instance format (for multiple decision lists): (cf. Figure 5.2 (b))

$$\left(\text{first}\left(R_1, d\right), \cdots, \text{first}\left(R_{|E|}, d\right), l\right)$$

Numerical Features: For each base decision list R, we create a numerical attribute whose domain consists of its pattern indices ($\{1, \cdots, |R|\}$) used as numerical values. A meta instance is composed of the indices of the respective first covering rule of each classifier and if known its original class value.

Meta instance format (for multiple decision lists): (cf. Figure 5.2 (c))

$$\left(\text{index}\left(c_1, d\right), \cdots, \text{index}\left(c_{|E|}, d\right), l\right)$$

Please note that the default patterns of the base level decision lists are ignored in the meta data generation. Consequently, the resulting meta data set using nominal or numerical features has the same number of attributes as the number of decision lists, analogous to the standard stacking approach where the number of meta attributes is equal to the number of base models. In the case of binary features, the resulting meta data set has as many attributes as the total number of patterns in the local pattern set(s). Additionally, binary features allow two types of pattern conditions. The first one, which is also used for nominal and numerical features respectively, encodes that an instance d must be covered by the pattern r in the pattern set R (denoted as $\text{covers}(r, d) = \text{true}$) to meet the condition. The second one is analogous except that its test is negated. So, the condition is met if the instance is not covered by the pattern (denoted as covers(r, d) = false). In our experiments, we will investigate if negated conditions are a good addition to the first condition type.

Let us illustrate the generation of the meta data set with the help of a toy example. One of the data sets used in our experiments was *zoo* (see Section 5.5.2), which records the characteristics of animals divided in different classes (referred to as types in this data set), e.g. insects or invertebrates. The last classifier of the learned pairwise ensemble tries to distinguish exactly these two animal classes:

ID	Rule
r_1	(airborne = true) $\Longrightarrow$ type=insect
r_2	(predator = false) $\wedge$ (legs $\geq$ 6) $\Longrightarrow$ type=insect
r_3	$\Longrightarrow$ type=invertebrate

In the following explanations, we will use this decision list as a local pattern set for rule stacking. Assuming that we want to transform the following instances (only relevant attribute values are shown):

Name	Airborne	Predator	Legs	Type
Termite	false	false	6	insect
Lobster	false	true	6	invertebrate
Crow	true	true	2	bird

in the respective meta data format, we would get the following values for the meta attributes belonging to the given decision list (only the meta attribute values are shown, and the default pattern is ignored):

Name	Binary Features	Nominal Features	Numerical Feat.
Termite	$(\cdots$, false, true, insect)	$(\cdots, r_2$, insect)	$(\cdots$, 2, insect)
Lobster	$(\cdots$, false, false, invert.)	$(\cdots, r_3$, invert.)	$(\cdots$, 3, invert.)
Crow	$(\cdots$, true, false, bird)	$(\cdots, r_1$, bird)	$(\cdots$, 1, bird)

5.4 Retransforming the Meta Model

In this section, we are going to show how the model obtained at the meta level may be retransformed into the original data format so that it can be directly used for the classification of unknown instances. This simple idea is the key advantage of our approach which distinguishes it from previous works on compressing. Eventually, we obtain a single classifier that directly operates on the base level, as it is composed of all relevant patterns from the rule-based local pattern sets, and additionally maintains the accuracy of the meta level classifier. In contrast to the original stacking scheme, we do not need to store the original ensemble of global models (or the local pattern sets), nor do we need to transform a test instance into the meta format, sparing the use of the aforementioned base classifiers. Furthermore, the obtained retransformed model is obviously easier to understand or interpret than an ensemble of classifiers whose predictions are combined by an additional meta model.

The efficiency of this retransformation process depends on the meta feature generation approach. The proposed meta features (binary, nominal, or numerical features) are based on different assumptions and therefore necessitate different retransformation schemes. Consequently, they are differently suitable for the retransformation process. In the following, we will discuss the properties of each of the suggested meta features and explain why we choose only one of them for our experiments.

In the case of nominal features, the pattern conditions encode whether a specific pattern is the first covering one in a decision list ($\text{first}(R, d) = r$). This type of condition contains additional implicit information, since we know that all rules which are in a higher position in the decision list did not cover the instance. So, if we want to retransform this condition we would have to determine a conjunction of base conditions for each of these predecessor rules and negate each of these conjunctions. Obviously, this would greatly boost the size of our retransformed meta classifier. Otherwise, if we ignore this additional information in the retransformation process, the resulting meta classifier would be inferior to the meta classifier before its retransformation. Hence, we assume that nominal features are not a good choice for our purpose.

For numerical features, a conjunction of meta level features may denote eventually whether the first covering rule lies in a given interval of the base rules (determined by an interval of indices), i.e., the conditions can be of the form $k \leq \text{index}(R, d) \leq l$ (please note that open intervals are likewise feasible). Obviously, this approach is more expressive than the previous one and therefore the resulting global rule set could be more compact than in the nominal case. On the other hand, a back transformation of one condition of this type does not result in a simple conjunction but in a more complex DNF expression (a disjunction of conjunctions). If a meta level rule combines multiple such interval features into a conjunctive rule, the premise of the resulting base level rule would be a conjunction of DNF expressions. Thus, the direct utilisation of these features in base level rules results in much more complex rules. One may consider to compress these complex expressions into simpler DNF expressions, but we have not yet dealt with this issue because preliminary experiments showed that this complex approach does not offer any gain in predictive accuracy over the simpler nominal or binary encodings. Consequently, we refrain from using numerical features in our experiments, too.

In the case of binary features, a condition tests whether a specific pattern covers an instance or not. Hence, we get two possible meta conditions:

"covers(r, d) = true" and its negation "covers(r, d) $\neq$ true" (corresponds to "covers(r, d) = false"). In both cases, the truth value of the meta condition can be established by only testing the conditions of a single base rule, all other rules of the base local pattern set R may be ignored. As the transformation of negated meta conditions is a bit more delicate we will describe the conversion of patterns without negated conditions first and address this issue afterwards. Since we know that the local pattern sets consist of conditions which are based on rules of the base classifiers, we can distinguish two cases. In the first case, the global rule consists of only one condition, so we can directly replace the condition of the global rule with the conditions of the base rule. In the second case, the global rule consists of more than one condition hence the conditions must be merged. Each global condition corresponds to a test if a base rule covers an instance and consequently corresponds to a conjunction of the conditions of the involved base patterns. Thus, the global conditions can be merged by concatenating the conjunctions of the conditions of their corresponding base patterns. Duplicate conditions are removed. Of the three considered meta feature approaches, the binary one offers the most promising retransformation process. The retransformation of binary features is illustrated in Figure 5.1.

As already mentioned, the situation is somewhat more complicated if negated meta conditions are allowed, because a negated meta condition corresponds to a negated conjunction of base conditions. We currently simply add the negated conjunction directly to the retransformed rule. This has the effect that the resulting rule set is no longer in disjunctive normal form (DNF). One may argue that in this case, rule stacking may have a somewhat unfair advantage over ordinary rule learners which are confined to conjunctions in their rule bodies. For this reason, we report results on both variants, the one where negated meta conditions are allowed, and the one where they are forbidden.

5.5 Experimental Setup

The goal of our experiments is to investigate, how well our rule stacking approach solves the task to transform local patterns into a compressed understandable model. To this end, we decide to utilise an ensemble learning method in liaison with a decision list learner to obtain several classifiers. In this way, we examine whether our rule stacking approach can maintain the improved performance of a classifier ensemble, and we consider the understandability of the compressed global model. As mentioned before, understandability is a subjective measure. So, we choose to use the complexity of the model instead, as it may be regarded as an approximation of the model's understandability (less complex models should be easier to understand) and it may be used to measure the compression of the model, too. Additionally,

Table 5.1.: Data sets used in the experiments with the number of instances, number of nominal and numeric attributes, and number of classes.

		Attributes		
Data set	Instances	Nominal	Numeric	Classes
Aneal.orig	798	29	9	6
Autos	205	10	15	7
Balance-scale	625	0	4	3
Bridges Version 1	107	9	3	6
Cars	1728	6	0	4
CMC	1473	7	2	3
Dermatology	366	34	1	6
Ecoli	336	0	7	8
Glass	214	0	9	7
Hypothyroid	3772	22	7	4
Lymph	148	15	3	4
Optdigits	5620	0	64	10
Pageblocks	5473	0	10	5
Segment	2310	0	19	7
Solar-flare-c	1389	10	0	9
Soybean	683	35	0	19
Splice	3190	60	0	3
Thyroid_hyper	3772	22	7	5
Thyroid_rep	3772	22	7	5
Vehicle	846	0	18	4
Vowel	990	3	10	11
Waveform-5K	5000	0	40	3
Yeast	1484	0	8	10
Zoo	101	15	1	7

we consider the accuracy of the model, since without regard of accuracy the least complex and in most of the times inaccurate model would be a model that consists only of a single default rule.

5.5.1 Pattern Discovery & Rule Stacking

In our experiments, we decide to use the ensemble method pairwise class binarisation for the generation of a classifier ensemble and Ripper as its base learner. This decision is driven by three reasons. First, the diversity of the learning tasks in the individual ensemble members of pairwise classifiers is considerably higher than for sampling-based ensemble methods such as bagging, because each base level classifier tackles a different binary learning problem, whereas bagging tackles different training sets for the same classifier. We expect that this higher diversity makes it harder to compress the rules into a single classifier. Second, pairwise theories are obviously hard to interpret. At last, it is known that a pairwise ensemble of Ripper has a better performance than a single standard Ripper [Coh95], and we want to see whether this improved performance may be maintained.

Our experiments are performed within the WEKA framework [WFHP16]. For the pattern generation, we employ again the rule learner JRip (the Weka implementation of Ripper [Coh95]). As previously mentioned, this reimplementation does not support the unordered mode for learning rule sets, but, as we only deal with binary base level classification problems obtained by the pairwise class binarisation of multi-class classification problems [Für02], there is no practical difference between these two modes. However, we employ only the ordered mode of Ripper as it seemed more adequate for our purposes. Additionally, both pruning methods of Ripper, the incremental reduced error pruning and the pre-pruning that uses a minimum description length heuristic are applied.

For our rule stacking instantiation, the resulting decision lists are transformed into local pattern sets that consists of all their patterns except for the default rules. In the meta data generation step (see Section 5.4), we employ binary features only, since the nominal and numerical features are less suited for the subsequent retransformation step (as we explained in Section 5.3).

5.5.2 Evaluation

We evaluate the above setup on 24 multi-class data sets (see Table 5.1) of the UCI repository [Lic13]. Since the number of classes differs highly in these data sets we get a great range of different ensemble sizes. In our experiments, we always employ binary features at the meta level and either allow negated meta conditions (abbreviated as w.N., an acronym for **w**ith **n**egation) or we do not (abbreviated as wo.N., an acronym for **w**ith**o**ut **n**egation). We compare our rule stacking approach (denoted by RS) to the standard JRip and to its pairwise variant using a pairwise class binarisation (denoted by PW). For this comparison, we consider the accuracy of each classifier and the size of the generated model measured by the number of rules and the total number of conditions of all rules. The number of conditions of our retransformed meta classifier is determined by removing duplicate conditions, so each condition is only counted once. The accuracy of each classifier is estimated using a 10-fold cross-validation. For the evaluation of these comparisons, we use the Friedman test with a post hoc Nemenyi test as proposed in [Dem06] (see Section 2.4.4). The significance level is set to 5% for both tests. The results of the Nemenyi tests are depicted as critical distance charts.

Additionally, we calculate a normalised average complexity score for the number of rules and rule conditions for both configurations of the rule stacking approach, for JRip and its pairwise variant. For each data set, we divide their complexity scores by the maximum, non-redundant complexity of the data set. Afterwards, we calculate the average complexity of the four compared learning algorithms. For the number of rules, the maximum complexity is equal to the

Table 5.2.: Comparison of the performance of the standard JRip, its pairwise variant (PW) and rule stacking with (w.N.) or without negated meta conditions (wo.N.): number of classes ($|L|$) and pairwise problems (PP), and accuracy.

			Accuracy			
Data Set	$\|L\|$	PP	JRip	PW	w.N.	wo.N.
anneal	6	15	.9532	.9488	.9521	.9499
autos	6	15	.7317	.7512	.7415	.7659
balance-scale	3	3	.8080	.7872	.7904	.7376
bridges v1	6	15	.6190	.6286	.6571	.6190
car	4	6	.8646	.9034	.8750	.8791
cmc	3	3	.5241	.5485	.5336	.5255
dermatology	6	15	.8689	.9126	.9262	.9153
ecoli	8	28	.8125	.8155	.8125	.8065
glass	7	21	.6869	.6822	.6963	.6869
hypothyroid	4	6	.9934	.9939	.9939	.9939
lymph	4	6	.7770	.7973	.7973	.7973
optdigits	10	45	.9078	.9496	.9301	.9251
pageblocks	5	10	.9684	.9702	.9693	.9693
segment	7	21	.9571	.9645	.9615	.9216
solar-flare-c	8	28	.8540	.8522	.8534	.8534
soybean	19	171	.9195	.9283	.9092	.9165
splice	3	3	.9370	.9455	.9486	.9473
thyroid_hyper	5	10	.9849	.9867	.9870	.9870
thyroid_rep	5	10	.9894	.9905	.9902	.9902
vehicle	4	6	.6856	.7151	.6998	.6927
vowel	11	55	.6970	.8081	.7828	.7525
waveform-5K	3	3	.7920	.7922	.7892	.7908
yeast	10	45	.5809	.5788	.5748	.5735
zoo	7	21	.8614	.8713	.9208	.9109
Average			.8239	.8384	.8372	.8295
Average Rank			3.13	2.00	2.15	2.73

number of instances as each rule covers only one instance. Since the maximal length of a rule is equal to the number of attributes, the maximum number of rule conditions is the product of the number of instances and the number of attributes. The results of the calculations are summarised in two accuracy-complexity-charts.

5.6 Results

In our experiments, we investigate how well our rule stacking approach solves the task to transform local patterns into a compressed understandable model. For this purpose, we apply rule stacking to local patterns generated by a classifier ensemble and compare the resulting models of rule stacking to the models of the base classifier and of the classifier ensemble. In our evaluation of the experimental results, we consider the following three aspects:

- **Accuracy:** Does rule stacking maintain the (high) accuracy of the classifier ensemble?

Table 5.3.: Comparison of the performance of the standard JRip, its pairwise variant (PW) and rule stacking with (w.N.) or without negated meta conditions (wo.N.): number of classes ($|L|$) and pairwise problems (PP), and size of the model (number of rules and conditions).

			Rules				Conditions			
Data Set	$\|L\|$	PP	JRip	PW	W.N.	WO.N.	JRip	PW	W.N.	WO.N.
anneal	6	15	14	36	12	12	37	38	36	28
autos	6	15	13	41	14	14	25	42	30	29
balance-scale	3	3	12	15	9	2	39	35	24	2
bridges v1	6	15	5	33	6	7	6	21	12	14
car	4	6	49	66	37	38	195	210	143	146
cmc	3	3	5	12	4	4	14	26	15	16
dermatology	6	15	15	39	8	13	27	32	23	39
ecoli	8	28	10	55	9	8	19	35	22	18
glass	7	21	8	42	10	11	18	39	34	37
hypothyroid	4	6	5	16	6	6	11	21	19	16
lymph	4	6	6	14	6	6	8	11	9	9
optdigits	10	45	74	220	80	90	312	391	614	657
pageblocks	5	10	14	36	15	17	30	54	44	53
segment	7	21	24	63	24	25	63	72	85	78
solar-flare-c	8	28	2	33	2	2	4	11	3	3
soybean	19	171	26	355	26	27	45	199	48	56
splice	3	3	14	15	11	12	55	38	43	48
thyroid_hyper	5	10	5	20	6	6	14	29	21	20
thyroid_rep	5	10	8	14	6	6	22	18	16	16
vehicle	4	6	17	31	19	14	43	55	77	56
vowel	11	55	48	199	52	61	138	260	292	325
waveform-5K	3	3	30	46	46	46	121	163	316	284
yeast	10	45	15	127	18	16	38	153	69	59
zoo	7	21	6	43	7	7	6	23	12	15
Average		23,21	17,71	65,46	18,04	18,75	53,75	82,33	83,63	84,33
Average Rank			1,83	3,96	1,98	2,23	1,71	3,33	2,44	2,52

- **Complexity:** Does rule stacking reduce the complexity of the classifier ensemble?

- **Trade-off between accuracy and complexity:** What is the trade-off between the accuracy and complexity of the models obtained by rule stacking?

The detailed results of our experiments are shown in Tables 5.2 and 5.3. We apply a Friedman test to each considered evaluation measure: accuracy, number of rules and number of conditions. In all three cases, the test rejects the equality of the compared classifiers. So, post hoc Nemenyi tests are performed, the corresponding CD-charts are depicted in Figure 5.3. For the trade-off between accuracy and complexity, we chart the four learning algorithms according to their accuracy and complexity in Figure 5.4.

5.6.1 Accuracy

For accuracy, the Nemenyi test identifies two groups of equivalent classifiers (see Figure 5.3(a)). One can see that the pairwise variant of JRip significantly outperforms its standard variant, confirming the results of [Für02]. Moreover, both variants of rule stacking are not significantly worse than the pairwise variant, but allowing negations at the meta level clearly seems to be preferable. While this variant is somewhat worse than the original pairwise version of JRip, this difference is quite small, and not significant. However, it is still significantly better than the conventional JRip. On the other hand, the variant that does not allow negated meta conditions still seems to be preferable over conventional JRip, but this result is not statistically significant. Similar observations can be made if the average accuracies of the classifiers are compared in Table 5.2.

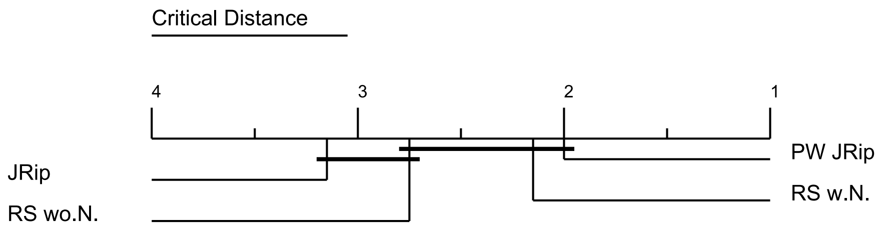
5.6.2 Complexity

Considering the number of induced rules (see Figure 5.3(b)), we discern two disjunct groups of equivalent classifiers. Pairwise JRip is the single member of the worst group, i.e., it typically induces larger rule sets than the other three classifiers. All of them belong to the best group of classifiers, hence their rule sets are of a comparable size. This finding is also reflected in the average number of induced rules for each classifier in Table 5.3.

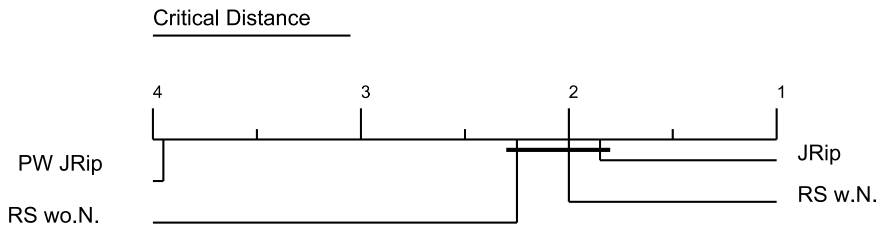
At last, we compare the number of rule conditions of the considered classifiers (see Figure 5.3(c)). Here, the results are more diverse. In essence, both versions of the rule stacking approach lie in the middle of these two classifiers, being neither significantly worse than JRip nor significantly better than pairwise JRip. The detailed results in Table 5.3 confirm this diversity, mixing results where the size of the resulting theory is even smaller than JRip's (e.g., *car* or *soybean*), with results where it is considerably higher than the simple pairwise approach (e.g. *optdigits* or *waveform500*). The latter results dominate the average values.

5.6.3 Trade-off between Accuracy and Complexity

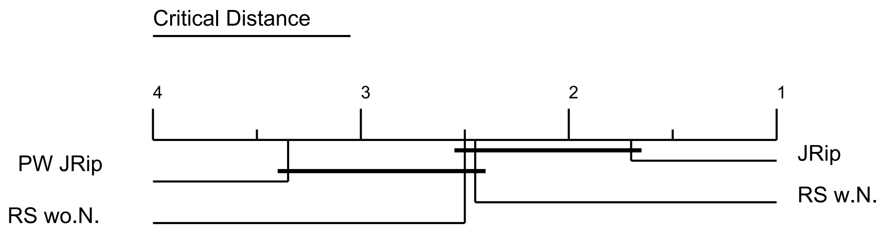
After the individual evaluation of the two performance measures (accuracy and complexity), we consider the trade-off between accuracy and complexity of the considered rule learning algorithms. As mentioned before, we use two accuracy-complexity-charts to visualise the trade-off between these two measures. In each of these charts, the standard and the pairwise version of JRip and our two rule stacking configurations are charted according to their average



(a) accuracy



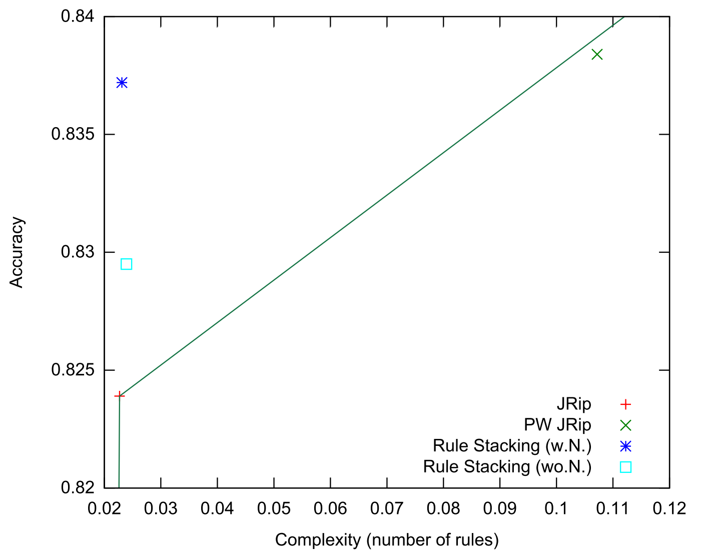
(b) number of rules



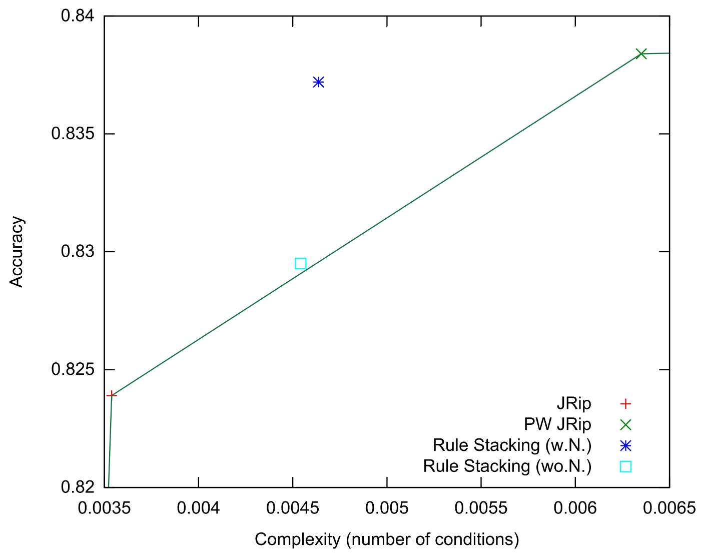
(c) number of rule conditions

Figure 5.3.: Critical distance charts.

accuracy and normalised average complexity score (based on either the number of rules or the number of rule conditions). Additionally, the empty rule set ([0, 0]), the complete rule set ([1, 1]), and the standard and pairwise JRip are utilised to plot a convex hull. For a better visibility, only partial views of these accuracy-complexity charts are shown (see Figure 5.4). In the first accuracy-complexity chart (see Figure 5.4(a)), the trade-off between accuracy and the complexity score based on the number of rules is depicted. Both configurations of rule stacking lie above the convex hull of the benchmark learner, therefore one can conclude that rule stacking is an eligible alternative to the standard and the pairwise JRip. Additionally, rule stacking maintains the low complexity score of the standard JRip and provides an increased average accuracy. Of the two configurations, the one that allows negated meta conditions exhibits a better trade-off between accuracy and complexity than the other one, as its average accuracy is comparable to that of the pairwise JRip. However, this su-



(a) number of rules



(b) number of rule conditions

Figure 5.4.: Accuracy-complexity charts: trading off accuracy against the number of rules and rule conditions, respectively.

periority may be traced back to the fact that the first one uses more expressive rules than the latter.

In the second chart (see Figure 5.4(b)), the trade-off between accuracy and the complexity score based on the number of rule conditions is shown. As the average accuracies of the considered rule learner is the same as in the first chart, we compare only the complexity scores. Expectedly, the average complexity scores of the two rule stacking configurations lie above the convex hull and midway between those of the standard and the pairwise JRip. Nevertheless, rule stacking is, according to the number of rule conditions, a sound alternative to the two variants of JRip. Summarising the results of both charts, rule stacking generates a similar number of rules as the standard JRip, but these rules are slightly more complex.

5.7 Summary

In this chapter, we introduced our rule stacking algorithm that allows to compress local pattern sets into a single compact and understandable model. In our experiments, we investigated how well rule stacking solves this task. To this end, rule stacking transformed the local patterns obtained by a classifier ensemble into a global model. Afterwards, we compared the performance of the models obtained by rule stacking (with and without negations at the meta level), by the base classifier (in our experiments Ripper was employed), and by the classifier ensemble (applying a pairwise class binarisation):

- **Accuracy:** First, we investigated the accuracy of these models. Rule stacking with or without negated conditions maintained the high performance of the classifier ensemble which performed best. The ensemble and rule stacking with negated conditions performed even significantly better than the base classifier, while rule stacking without negated conditions was only insignificantly better than the latter. Hence, the employment of negated conditions at the meta level is clearly favourable in respect of accuracy.

- **Complexity:** The resulting model is often of comparable complexity to the one directly learned by the base learner, and considerably less complex than the original local pattern sets. In terms of the number of patterns, this advantage is consistent and significant. However, in terms of the number of conditions, there are also a few cases where the retransformed rules are considerably more complex because of base patterns that are used multiple times in various meta patterns. However, we have to note that a structured representation of these patterns (e.g., through generated meta level features) may still provide a useful compression of the pairwise theories.

- **Trade-off between accuracy and complexity:** Last, we examined the trade-off between the accuracy and complexity of the models. To this end, we displayed the four methods according to their performance in two accuracy-complexity charts: one for the number of patterns and one for the number of conditions. In both charts, one can see that the two rule stacking configurations are good alternatives to the other two methods as they may not be approximated by a linear interpolation of the base classifier and the classifier ensemble. Additionally, rule stacking obviously performs better when negated meta conditions are allowed as it may use in this case more expressive rules than rule stacking without negated meta conditions.

The key result of these experiments is that rule stacking, in particular if negated meta conditions are allowed, maintains the high improvement in accuracy of the classifier ensemble, while often providing a good compression of the classifier ensemble. As a result, we often obtain rule sets that are of comparable complexity to those learned by the base classifier but are considerably more accurate. In summary, our rule stacking approach proved to be a good solution to the question, how a set of local patterns may be transformed into a compact and understandable model.

Our work may be viewed in the context of approaches that induce rules from opaque concepts such as neural networks [ADT95], support vector machines [Die08], or ensembles [Dom99]. Our approach differs from these approaches in that it extracts patterns from interpretable concepts, and therefore does not need to consider the predictions of these models but can directly use the learned patterns. Such ideas have already been used in somewhat different context (cf., e.g., [vdB00]), but its implementation in terms of a stacking framework and, most importantly, with the goal of compressing pattern sets, is new.

6 Conclusions

In this thesis, we tried to answer the question, how local patterns may be employed for global modelling. To this end, we investigated three important aspects of this problem:

6.1 Theory Formation

In the first part of our work, we investigated the question, **how a set of local patterns may be employed to obtain optimal predictions**. As mentioned before, this problem may be solved by the generic LeGo framework which divides it into three sub steps. Subsequently, these steps - the local pattern discovery, the pattern set discovery, and the global modelling - generate an usually large set of local patterns, reduce it to a smaller one that ideally contains all necessary patterns, and utilise this reduced pattern set to generate a global model. In accordance with this division, we have split the initial problem into three sub-questions that we have investigated by an empirical comparison of a selection of methods for each of the three associated steps:

- **Local pattern discovery:** How is an optimal local pattern set discovered?

  For the local pattern discovery step, the performances of the two pattern discovery algorithms BSD and CHARM were compared. Despite differing in various aspects, the two algorithms (e.g. supervised vs. unsupervised learning) did not differ significantly in their performance. Even though the discovered patterns are the basic components of the global model, the way they are discovered seems less important (as long as the patterns feature a sufficiently high quality).

- **Pattern set discovery:** How is an optimal pattern set selected?

  For the pattern set discovery step, we compared the performance of five selection methods: the all selector which keeps all of the original patterns, two confidence-based methods that select either the k-most confident patterns or all patterns above a minimum confidence threshold, and the two forward selection methods that similarly select only k patterns. The latter four methods that generated reduced patterns set all performed at least as well or better than the all selector that returned the complete pattern set. Consequently, we conclude that the pattern discovery step is an essential component of the global modelling process that may eventually be improved by it. Furthermore, the two confidence based methods worked best. The method that employed a minimum confidence threshold even performed significantly better than the remaining methods, but we have to admit that it selected considerably more patterns than the other methods (except for the all selector). A reason for this may be that confidence does not suffer from its common deficiency as the two employed pattern discovery methods feature an integrated overfitting avoidance.

- **Global modelling:** How is a pattern set optimally employed as a global model?

  For the global modelling step, we compared six decoding methods: Bayesian decoding, best rule, inverse weighted voting, linear weighted voting, voting, and weighted voting, that all pursue an different approach to exploiting the information which patterns cover an instance. The decoding methods weighted voting, best rule and Bayesian decoding (in descending ordering according to their performance) that utilised the estimated unmodified quality values directly performed better than voting that binarised the quality values, and better than inverse weighted voting and linear weighted voting that both employed the quality values to derive a coarse ranking.

In summary, we conclude that the optimal way to utilise local patterns for prediction is to select a pattern set according to the confidence values of the patterns and simply add up these confidence values for a global prediction.

6.2 Probability Estimation

In the second part of our work, we considered the problem, **how a set of local patterns may be utilised to obtain optimal (class) probabilities**. The prediction of class probabilities, which determine the probability that a data instance belongs to a given class, may often be more useful as a simple prediction of a class label as we have seen in the first part of our work where a measure of confidence was employed for voting. The problem of the class probability estimation may be divided into two main tasks: the probability estimation, which involves a basic probability estimation and if applicable the employment of shrinkage, and the probability aggregation. In our evaluations, we considered the following four aspects of these tasks:

- **Basic probability estimation:** The first and fundamental aspect is the basic probability estimation which aims to estimate the class probability distribution of a considered data instance given a single covering local pattern. All considered methods performed better than the original benchmark learner Ripper. Among the considered methods the *m*-Estimate (indifferent of the employed *m*-parameter) ranked before the others, but this difference in performance was insignificant.

- **Shrinkage:** The second aspect is shrinkage whose base idea is to improve the basic probability estimation by combining it with further information (sources). For our considerations, we adapted a shrinkage algorithm that intends to exploit the probability estimates of selected sub patterns. This approach turned out to decrease (in some cases even significantly) the performance in comparison to the performance of the basic probability estimation used stand-alone. Thus, the employment of this specific shrinkage approach is not recommended, but other approaches may potentially lead to improvements.

- **Pattern discovery:** The third aspect is the effect of the pattern discovery approach on the probability estimation. Ripper, which was also employed for the pattern discovery, offers two options that had an impact on the performance of the probability estimation: the generation mode and the employment of pruning. The unordered generation mode performed significantly better than the ordered one as the discovered patterns of the former are more suitable for the probability estimation as the latter. Similarly, unpruned pattern sets outperformed (insignificantly) the pruned ones.

- **Probability aggregation:** The fourth and last aspect is the probability aggregation which intends to combine the probability estimates of multiple covering patterns into a single one. We observed that this approach clearly improves the probability estimation process. Amongst the considered methods, micro averaging that simply calculates the average class probability of all covering patterns outperformed the other methods though this difference in performance was insignificant.

In summary, unpruned local patterns that are generated in an unordered way should be used for class probability estimation. The basic probability estimation method *m*-estimate is preferable for this task. If possible, the probability estimates of multiple covering patterns should be combined by micro averaging.

6.3 Theory Compression

In the third part of our work, we studied the question, **how a set of local patterns may be transformed into a compact and understandable model**.

The generation of a global model with the help of local patterns solves as mentioned before two problems of local pattern sets. On the one hand, pattern discovery methods usually return large and redundant sets of local patterns that are consequently hard to handle and interpret. On the other hand, local pattern sets may only be employed for predictions by an additional decoding of the individual predictions of each pattern into a single one (e.g. by voting methods). To solve the aforementioned question, we proposed a novel approach, which we call rule stacking. This approach adapts the general meta learning method stacking to the global modelling of rule-based local pattern sets. For this purpose, rule stacking introduces an alternative meta data generation approach and an additional retransformation of the obtained meta model into a global model in the original data format. In this way, the model may be directly applied to data instances in the original data format, which do not have to be translated into meta instances any more.

In our experiments, we employed a base learner (Ripper was chosen for this task) to generate multiple local pattern sets in liaison with a pairwise class binarisation and to induce global models. Afterwards, we compared the models of rule stacking, which also employed Ripper for the model induction, to the models of the base classifier and the classifier ensemble. To this end, we considered the following aspects:

- **Accuracy:** First, we investigated the accuracy of the four abovementioned models. As desired, the performance of rule stacking did not differ significantly from the performance of the classifier ensemble. In other word, rule stacking was able to maintain the high performance of the classifier ensemble. Rule stacking was even significantly more accurate than the base classifier if it employed negated meta conditions.

- **Complexity:** Next, we compared the complexity of the models. Complexity was chosen as an approximation of the interpretability of a model which may not be measured objectively. However, as a rule of thumb less complex models are easier to interpret. To this end, we measured the complexity of a model by its number of patterns and by the total number of conditions. In terms of the number of patterns, the complexity of rule stacking was comparable to the complexity of the base classifier. Additionally, rule stacking was still comparable to the base classifier, but its complexity is only insignificantly better than the complexity of the classifier ensemble. In a nutshell, the complexity of rule stacking is comparable to the complexity of the base classifier, but the learned patterns are slightly more complex.

- **Trade-off between accuracy and complexity:** At last, we examined the trade-off between the accuracy and complexity of the models. So far, we have seen that rule stacking maintained the high accuracy of the classifier ensemble and the models obtained have a comparable number of (slightly longer) patterns as the base classifier. This observation was backed up by two accuracy-complexity charts. In these charts, it is obvious that rule stacking is a good alternative to the base classifier or classifier ensemble, respectively, as rule stacking with or without negated meta conditions lies outside of the convex hull of the base classifier and classifier ensemble. Furthermore, the employment of negated meta conditions is clearly advisable although the difference in performance of rule stacking with or without negated meta conditions is insignificant.

In summary, rule stacking provides a good solution to the question, how a set of local patterns may be transformed into a compact and understandable model. To this end, rule stacking, in particular if negated meta conditions are allowed, maintains the high accuracy of the classifier ensemble, while it offers a considerable compression of the classifier ensemble.

7 Future Work

In this thesis, we considered three aspects of the question, how local patterns may be employed for global modelling. For this purpose, we evaluated various approaches that may be utilised to solve these aspects. Our investigations opened up new issues and improvement opportunities. On the one hand, some approaches did not perform as well as we expected, but alternative setups or implementations of these approaches could still yield better results. On the other hand, the above-mentioned three aspects were investigated separately, but the results of the last two aspects, the probability estimation and theory compression, may clearly be used for an improvement of the first aspect, the theory formation.

7.1 Theory Formation

In the first part of our work, we performed empirical evaluations in the LeGo framework. To this end, we selected several methods for each of its steps and evaluated their performance aiming to identify the best choice(s) in each step:

- **Pattern set discovery:** For the pattern set discovery, we employed a greedy forward selection for the two methods joint entropy and exclusive coverage to obtain a pattern set. As both methods did not perform very well in comparison to the other simpler methods (including the all selector which essentially skips the pattern set discovery) in our experiments, we assume that the forward selection was not an optimal approach in this scenario. Hence, alternative search approaches should be considered for the greedy pattern set generation (e.g. a beam search or an exhaustive search) as they could possibly improve the performance of the above-mentioned methods.

- **Global modelling:** In the global modelling step, we employed a decoding approach using several voting methods to decode multiple predictions into a single one. This decision was driven by the consideration that in this way all the patterns selected in the pattern set discovery step are involved in the prediction phase. Nevertheless, an investigation of an integrative approach could be worthwhile. Since classifiers usually have to be modified at least slightly to be able to handle such pattern sets, we suggest to employ our introduced rule stacking algorithm as an integrative approach for the global modelling step.

7.2 Probability Estimation

In the second part of our work, we considered several methods for the probability estimation and aggregation, respectively:

- **Shrinkage:** While the basic probability estimation methods did perform well, the employed shrinkage method did not increase their performance. Nevertheless, we still think that the concept of smoothing the basic probability estimation on the basis of shorter sub rules, which may usually draw upon a higher evidence, may lead to improved probability estimates. Thus, the search for or development of alternative shrinkage approaches and their evaluation could turn out to be reasonable.

- **Probability aggregation:** For the probability aggregation, we employed four aggregation methods. While the performance of these methods differed significantly for a lower number of patterns, the performance differences decreased for a higher number. We concluded that in the latter case the method best rule performed similar to the other methods, which calculate an aggregated probability distribution, because the calculations of the latter are unfavourably influenced by lower quality patterns. To solve this problem, we assume that an additional filtering of the covering patterns (e.g. in accordance to their estimated probability) could be worthwhile.

7.3 Theory Compression

In the third part of our work, we introduced rule stacking as a novel approach to generate a compressed and understandable model:

- **Local pattern discovery:** We employed for this purpose the rule learner JRip and its pairwise variant both as learning algorithms and for the local pattern discovery. In this way, we obtained strong base rules and global models that could be used as benchmarks for the accuracy and complexity of the global models generated by rule stacking. For a more general evaluation, the next step would be the application of rule stacking to larger and more arbitrarily generated local pattern sets.

- **Pruning:** At the moment, rule stacking does not include any pruning methods, except for the pruning methods that are integrated in the employed local pattern discovery algorithm. As the latter operate only on the meta level, no base level pruning takes place. We argue that the employment of pruning - especially when applied to the base level - may be reasonable as pruning decreases the complexity of the obtained model and increases usually its interpretability and accuracy in this way. The simplest way to integrate pruning in the rule stacking process is the addition of a post-pruning method, which consequently occurs on the base level, after the retransformation of the meta model. Nevertheless, pre-pruning on the base level may also feasible. To this end, the base learner may have to be modified since the retransformation must be performed after each refinement step.

Bibliography

[ADT95] Robert Andrews, Joachim Diederich, and Alan B. Tickle. Survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge Based Systems*, 8(6):373–389, 1995.

[Agg14] Charu C. Aggarwal. Instance-based learning: A survey. In Charu C. Aggarwal, editor, *Data Classification: Algorithms and Applications*, pages 157–186. CRC Press, 2014.

[AIS93] Rakesh Agrawal, Tomasz Imielinski, and Arun N. Swami. Mining association rules between sets of items in large databases. In Peter Buneman and Sushil Jajodia, editors, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, D.C., USA*, pages 207–216. ACM Press, 1993.

[AL09] Martin Atzmüller and Florian Lemmerich. Fast subgroup discovery for continuous target concepts. In Jan Rauch, Zbigniew W. Ras, Petr Berka, and Tapio Elomaa, editors, *Foundations of Intelligent Systems, 18th International Symposium, ISMIS 2009, Prague, Czech Republic, Proceedings*, volume 5722 of *Lecture Notes in Computer Science*, pages 35–44. Springer, 2009.

[AL12] Martin Atzmüller and Florian Lemmerich. VIKAMINE - Open-source subgroup discovery, pattern mining, and analytics. In Peter A. Flach, Tijl De Bie, and Nello Cristianini, editors, *Machine Learning and Knowledge Discovery in Databases, European Conference, ECML PKDD 2012, Bristol, UK, Proceedings, Part II*, volume 7524 of *Lecture Notes in Computer Science*, pages 842–845. Springer, 2012.

[AS94] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *Proceedings of 20th International Conference on Very Large Data Bases (VLDB'94), Santiago de Chile, Chile*, pages 487–499. Morgan Kaufmann, 1994.

[Bre96] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[CB91] Peter Clark and Robin Boswell. Rule induction with CN2: Some recent improvements. In Yves Kodratoff, editor, *Machine Learning - EWSL-91, European Working Session on Learning, Porto, Portugal,* *Proceedings*, volume 482 of *Lecture Notes in Artificial Intelligence*, pages 151–163. Springer, 1991.

[Ces90] Bojan Cestnik. Estimating probabilities: A crucial task in Machine Learning. In Luigia Carlucci Aiello, editor, *ECAI 90, Proceedings of the 9th European Conference on Artificial Intelligence, Stockholm*, pages 147–150. Pitman, 1990.

[CG99] Stanley F. Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. *Computer Speech & Language*, 13(4):359–393, 1999.

[Coh95] William W. Cohen. Fast effective rule induction. In Prieditis and Russell [PR95], pages 115–123.

[DB95] Thomas G. Dietterich and Ghulum Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286, 1995.

[Dem06] Janez Demsar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

[Die00] Thomas G. Dietterich. Ensemble methods in machine learning. In Josef Kittler and Fabio Roli, editors, *Multiple Classifier Systems, First International Workshop, MCS 2000, Cagliari, Italy, Proceedings*, volume 1857 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2000.

[Die08] Joachim Diederich, editor. *Rule Extraction from Support Vector Machines*, volume 80 of *Studies in Computational Intelligence*. Springer, 2008.

[Dom99] Pedro M. Domingos. Metacost: A general method for making classifiers cost-sensitive. In Usama M. Fayyad, Surajit Chaudhuri, and David Madigan, editors, *KDD '99, Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA*, pages 155–164. ACM, 1999.

[Faw06] Tom Fawcett. An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8):861–874, 2006.

[FFH03] César Ferri, Peter A. Flach, and José Hernández-Orallo. Improving the AUC of probabilistic estimation trees. In Nada Lavrač, Dragan Gamberger, Ljupco Todorovski, and Hendrik Blockeel, editors, *Machine Learning: ECML 2003, 14th European Conference on Machine*

Learning, Cavtat-Dubrovnik, Croatia, Proceedings, volume 2837 of *Lecture Notes in Computer Science*, pages 121–132. Springer, 2003.

[FGL12] Johannes Fürnkranz, Dragan Gamberger, and Nada Lavrač. *Foundations of Rule Learning*. Cognitive Technologies. Springer, 2012.

[FI93] Usama M. Fayyad and Keki B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In Ruzena Bajcsy, editor, *IJCAI-93: Proceedings of the 13th International Joint Conference on Artificial Intelligence, Chambéry, France*, volume 2, pages 1022–1029. Morgan Kaufmann, 1993.

[FPS96a] Usama M. Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. From data mining to knowledge discovery: An overview. In *Advances in Knowledge Discovery and Data Mining*, pages 1–34. AAAI Press, 1996.

[FPS96b] Usama M. Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. From data mining to knowledge discovery in databases. *AI Magazine*, 17(3):37–54, 1996.

[Fri37] Milton Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32(200):675–701, 1937.

[Fri40] Milton Friedman. A comparison of alternative tests of significance for the problem of m rankings. *The Annals of Mathematical Statistics*, 11(1):86–92, 1940.

[Für99] Johannes Fürnkranz. Separate-and-conquer rule learning. *Artificial Intelligence Review*, 13(1):3–54, February 1999.

[Für02] Johannes Fürnkranz. Round robin classification. *Journal of Machine Learning Research*, 2:721–747, 2002.

[Für04] Johannes Fürnkranz. From local to global patterns: Evaluation issues in rule learning algorithms. In Morik et al. [MBS04], pages 20–38.

[FW94] Johannes Fürnkranz and Gerhard Widmer. Incremental reduced error pruning. In William W. Cohen and Haym Hirsh, editors, *Machine Learning, Proceedings of the Eleventh International Conference, New Brunswick, NJ, USA*, pages 70–77. Morgan Kaufmann, 1994.

[GRW08] Henrik Grosskreutz, Stefan Rüping, and Stefan Wrobel. Tight optimistic estimates for fast subgroup discovery. In Walter Daelemans, Bart Goethals, and Katharina Morik, editors, *Machine Learning and Knowledge Discovery in Databases, European Conference, ECML PKDD 2008, Antwerp, Belgium, Proceedings, Part I*, volume 5211 of *Lecture Notes in Computer Science*, pages 440–456. Springer, 2008.

[HV09] Eyke Hüllermeier and Stijn Vanderlooy. Why fuzzy decision trees are good rankers. *IEEE Transactions on Fuzzy Systems*, 17(6):1233–1244, 2009.

[ID80] Ronald L. Iman and James M. Davenport. Approximations of the critical region of the Friedman statistic. *Communications in Statistics, A: Theory and Methods*, 9:571–595, 1980.

[KCFS08] Arno J. Knobbe, Bruno Crémilleux, Johannes Fürnkranz, and Martin Scholz. From local patterns to global models: The LeGo approach to data mining. In Arno J. Knobbe, editor, *From Local Patterns to Global Models: Proceedings of the ECML PKDD-08 Workshop (LeGo-08)*, pages 1–16, Antwerp, Belgium, 2008.

[KH06a] Arno J. Knobbe and Eric K. Y. Ho. Maximally informative k-itemsets and their efficient discovery. In Tina Eliassi-Rad, Lyle H. Ungar, Mark Craven, and Dimitrios Gunopulos, editors, *KDD '06, Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Philadelphia, PA, USA*, pages 237–244. ACM, 2006.

[KH06b] Arno J. Knobbe and Eric K. Y. Ho. Pattern teams. In Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou, editors, *Knowledge Discovery in Databases: PKDD 2006, 10th European Conference on Principles and Practice of Knowledge Discovery in Databases, Berlin, Germany, Proceedings*, volume 4213 of *Lecture Notes in Computer Science*, pages 577–584. Springer, 2006.

[Klö96] Willi Klösgen. Explora: A multipattern and multistrategy discovery assistant. In *Advances in Knowledge Discovery and Data Mining*, pages 249–271. AAAI Press, 1996.

[LHM98] Bing Liu, Wynne Hsu, and Yiming Ma. Integrating classification and association rule mining. In Rakesh Agrawal, Paul E. Stolorz, and Gregory Piatetsky-Shapiro, editors, *KDD '98, Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining, New York City, NY, USA*, pages 80–86. AAAI Press, 1998.

[Lic13] Moshe Lichman. UCI machine learning repository, 2013.

[LRA10] Florian Lemmerich, Mathias Rohlfs, and Martin Atzmüller. Fast discovery of relevant subgroup patterns. In Hans W. Guesgen and R. Charles Murray, editors, *Proceedings of the Twenty-Third International Florida Artificial Intelligence Research Society Conference, Daytona Beach, Florida*. AAAI Press, 2010.

[MBS04] Katharina Morik, Jean-François Boulicaut, and Arno Siebes, editors. *Local Pattern Detection, International Seminar, Dagstuhl Castle, Germany, Revised Selected Papers*, volume 3539 of *Lecture Notes in Computer Science*. Springer, 2004.

[MS99] Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. The MIT Press, 1999.

[Mut04] Stefan Mutter. Classification using association rules. Master's thesis, Department of Computer Science, University of Freiburg, Germany, March 2004.

[Nem63] Peter Björn Nemenyi. *Distribution-free multiple comparisons*. PhD thesis, Princeton University, 1963.

[NLW09] Petra Kralj Novak, Nada Lavrač, and Geoffrey I. Webb. Supervised descriptive rule discovery: A unifying survey of contrast set, emerging pattern and subgroup mining. *Journal of Machine Learning Research*, 10:377–403, 2009.

[PD03] Foster J. Provost and Pedro M. Domingos. Tree induction for probability-based ranking. *Machine Learning*, 52(3):199–215, 2003.

[PR95] Armand Prieditis and Stuart J. Russell, editors. *Machine Learning: Proceedings of the Twelfth International Conference on Machine Learning, Tahoe City, CA, USA*. Morgan Kaufmann, 1995.

[PSF91] Gregory Piatetsky-Shapiro and William J. Frawley, editors. *Knowledge Discovery in Databases*. AAAI Press, 1991.

[Pyl99] Dorian Pyle. *Data Preparation for Data Mining*. Morgan Kaufmann, 1999.

[QC93] J. Ross Quinlan and R. Mike Cameron-Jones. FOIL: A midterm report. In Pavel Brazdil, editor, *Machine Learning: ECML-93, European Conference on Machine Learning, Vienna, Austria, Proceedings*, volume 667 of *Lecture Notes in Computer Science*, pages 3–20. Springer, 1993.

[Qui90] J. Ross Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990.

[Qui95] J. Ross Quinlan. MDL and categorial theories (continued). In Prieditis and Russell [PR95], pages 464–470.

[Ris78] Jorma Rissanen. Minimax codes for finite alphabets (Corresp.). *IEEE Transactions on Information Theory*, 24(3):389–392, 1978.

[RM14] Lior Rokach and Oded Maimon. *Data Mining With Decision Trees: Theory and Applications*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 2nd edition, 2014.

[SC08] Ingo Steinwart and Andreas Christmann. *Support Vector Machines*. Springer, 1st edition, 2008.

[See02] Alexander K. Seewald. How to make stacking better and faster while also taking care of an unknown weakness. In Claude Sammut and Achim G. Hoffmann, editors, *ICML '02, Proceedings of the Nineteenth International Conference on Machine Learning, University of New South Wales, Sydney, Australia*, pages 554–561. Morgan Kaufmann, 2002.

[SF08] Jan-Nikolas Sulzmann and Johannes Fürnkranz. A comparison of techniques for selecting and combining class association rules. In Arno J. Knobbe, editor, *From Local Patterns to Global Models: Proceedings of the ECML/PKDD-08 Workshop (LeGo-08)*, pages 154–168, Antwerp, Belgium, 2008.

[SF09] Jan-Nikolas Sulzmann and Johannes Fürnkranz. An empirical comparison of probability estimation techniques for probabilistic rules. Carl Smith award for the best student paper. In João Gama, Vítor Santos Costa, Alípio Mário Jorge, and Pavel Brazdil, editors, *Discovery Science, 12th International Conference, DS 2009, Porto, Portugal*, volume 5808 of *Lecture Notes in Computer Science*, pages 317–331. Springer, 2009.

[SF11] Jan-Nikolas Sulzmann and Johannes Fürnkranz. Rule stacking: An approach for compressing an ensemble of rule sets into a single classifier. In Tapio Elomaa, Jaakko Hollmén, and Heikki Mannila, editors, *Discovery Science - 14th International Conference, DS 2011, Espoo, Finland, Proceedings*, volume 6926 of *Lecture Notes in Computer Science*, pages 323–334. Springer, 2011.

[TW99] Kai Ming Ting and Ian H. Witten. Issues in stacked generalization. *Journal of Artificial Intelligence Research*, 10:271–289, 1999.

[vdB00] Antal van den Bosch. Using induced rules as complex features in memory-based language learning. In Walter Daelemans and Rémi Zajac, editors, *Fourth Conference on Computational Natural Language Learning, CoNLL 2000, and the Second Learning Language in Logic Workshop, LLL 2000, Held in cooperation with ICGI-2000, Lisbon, Portugal*, pages 73–78. ACL, 2000.

[WFHP16] Ian H. Witten, Eibe Frank, Mark A. Hall, and Christopher J. Pal. *Data Mining, Fourth Edition: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 4th edition, 2016.

[Wol92] David H. Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–259, 1992.

[Wro97] Stefan Wrobel. An algorithm for multi-relational discovery of subgroups. In Henryk Jan Komorowski and Jan M. Zytkow, editors, *Principles of Data Mining and Knowledge Discovery, First European Symposium, PKDD '97, Trondheim, Norway, Proceedings*, volume 1263 of *Lecture Notes in Computer Science*, pages 78–87. Springer, 1997.

[WZ06] Bin Wang and Harry Zhang. Improving the ranking performance of decision trees. In Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou, editors, *Machine Learning: ECML 2006, 17th European Conference on Machine Learning, Berlin, Germany, Proceedings*, volume 4212 of *Lecture Notes in Computer Science*, pages 461–472. Springer, 2006.

[ZH02] Mohammed Javeed Zaki and Ching-Jiu Hsiao. CHARM: an efficient algorithm for closed itemset mining. In Robert L. Grossman, Jiawei Han, Vipin Kumar, Heikki Mannila, and Rajeev Motwani, editors, *Proceedings of the Second SIAM International Conference on Data Mining, Arlington, VA, USA*, pages 457–473. SIAM, 2002.

Own Publications

Conference Papers

Jan-Nikolas Sulzmann and Johannes Fürnkranz. Rule stacking: An approach for compressing an ensemble of rule sets into a single classifier. In Tapio Elomaa, Jaakko Hollmén, and Heikki Mannila, editors, *Discovery Science - 14th International Conference, DS 2011, Espoo, Finland, Proceedings*, volume 6926 of *Lecture Notes in Computer Science*, pages 323–334. Springer, 2011.

Jan-Nikolas Sulzmann and Johannes Fürnkranz. An empirical comparison of probability estimation techniques for probabilistic rules. Carl Smith award for the best student paper. In João Gama, Vítor Santos Costa, Alípio Mário Jorge, and Pavel Brazdil, editors, *Discovery Science, 12th International Conference, DS 2009, Porto, Portugal*, volume 5808 of *Lecture Notes in Computer Science*, pages 317–331. Springer, 2009.

Jan-Nikolas Sulzmann, Johannes Fürnkranz, and Eyke Hüllermeier. On pairwise naive Bayes classifiers. In Joost N. Kok, Jacek Koronacki, Ramon López de Mántaras, Stan Matwin, Dunja Mladenic, and Andrzej Skowron, editors, *ECML*, volume 4701 of *Lecture Notes in Computer Science*, pages 371–381. Springer, 2007.

Workshop Papers

Jan-Nikolas Sulzmann and Johannes Fürnkranz. Probability estimation and aggregation for rule learning. In Martin Atzmüller, Dominik Benz, Andreas Hotho, and Gerd Stumme, editors, *Proceedings of LWA2010 - Workshop-Woche: Lernen, Wissen & Adaptivität*, Kassel, Germany, 2010.

Jan-Nikolas Sulzmann and Johannes Fürnkranz. A study of probability estimation techniques for rule learning. In Arno J. Knobbe and Johannes Fürnkranz, editors, *From Local Patterns to Global Models: Proceedings of the ECML/PKDD-09 Workshop (LeGo-09)*, pages 123–138, Bled, Slovenia, 2009.

Jan-Nikolas Sulzmann and Johannes Fürnkranz. A comparison of techniques for selecting and combining class association rules. In Arno J. Knobbe, editor, *From Local Patterns to Global Models: Proceedings of the ECML/PKDD-08 Workshop (LeGo-08)*, pages 154–168, Antwerp, Belgium, 2008.

Jan-Nikolas Sulzmann and Johannes Fürnkranz. A comparison of techniques for selecting and combining class association rules. In Joachim Baumeister and Martin Atzmüller, editors, *LWA*, volume 448 of *Technical Report*, pages 87–93. Department of Computer Science, University of Würzburg, Germany, 2008.

Jan-Nikolas Sulzmann. Pairwise naive Bayes classifier. In Klaus-Dieter Althoff and Martin Schaaf, editors, *LWA*, volume 1/2006 of *Hildesheimer Informatik-Berichte*, pages 356–363. University of Hildesheim, Institute of Computer Science, 2006.

A Theory Formation: Detailed Experimental Results

Table A.1.: Accuracy using all local patterns generated by CHARM and the decoding methods Bayesian decoding (BD), best rule (BR), inverse weighted voting (IV), linear weighted voting (LV), voting (V), and weighted voting (WV).

Data Set	BD	BR	IV	LV	V	WV
Anneal.orig	.8173	.7817	.7428	.7617	.7617	.7617
Autos	.4826	.6298	.2576	.4140	.4729	.5069
Balance-scale	.7059	.7425	.0784	.0784	.6899	.7410
Breast-cancer	.7030	.7101	.2970	.2970	.6892	.7382
Breast-w	.5700	.5509	.1827	.4445	.5245	.5891
Bridges Version 1	.7002	.7002	.0417	.3779	.8171	.7112
Cars	.7326	.8216	.2049	.5253	.7987	.8346
CMC	.4433	.4806	.2295	.3389	.4595	.4609
Diabetes	.5895	.6493	.2145	.2943	.6299	.7798
Ecoli	.6645	.5563	.1357	.3833	.5147	.6130
Glass	.7519	.7852	.1889	.4074	.8185	.8370
Heart-c	.7946	.7996	.3229	.7933	.7938	.7938
Heart-h	.6395	.8136	.1799	.3191	.7990	.8271
Heart-statlog	.5333	.8800	.5600	.7200	.8667	.9200
Hepatitis	.7767	.7233	.4567	.6633	.7000	.7167
Iris	.6414	.8033	.2776	.6819	.8038	.7905
Labor	.7187	.7344	.3033	.4335	.7370	.7344
Lymph	.7111	.7111	.2667	.4000	.7000	.7111
Postoperative-patient-data	.3230	.6883	.3230	.3230	.3230	.7815
Solar-flare-c	.8493	.8516	.0280	.4288	.8347	.8534
Tic-tac-toe	.7307	.9624	.3466	.3507	.7943	.8007
Titanic	.5141	.5667	.0838	.1788	.4172	.5929
Vowel	.6552	.8927	.4009	.9542	.9685	.9699
Yeast	.4461	.4879	.0203	.1638	.4354	.4542
Zoo	.8918	.9118	.9018	.9218	.9418	.9318

Table A.2.: Accuracy using the 25 local patterns generated by CHARM and selected by exclusive coverage, and the decoding methods Bayesian decoding (BD), best rule (BR), inverse weighted voting (IV), linear weighted voting (LV), voting (V), and weighted voting (WV).

Data Set	BD	BR	IV	LV	V	WV
Anneal.orig	.8475	.7795	.8475	.8475	.8385	.8374
Autos	.4086	.4086	.1898	.2140	.3893	.4086
Balance-scale	.6212	.6291	.0784	.1039	.4068	.6595
Breast-cancer	.7202	.7236	.2762	.2762	.2936	.7202
Breast-w	.4482	.4473	.4000	.4191	.4291	.4291
Bridges Version 1	.6968	.7002	.2598	.2604	.3843	.6997
Cars	.6042	.5974	.4555	.4555	.4555	.6042
CMC	.4270	.4284	.3021	.3062	.3082	.4257
Diabetes	.4969	.2621	.1278	.1575	.1666	.2621
Ecoli	.3931	.3314	.1452	.1452	.1545	.3364
Glass	.6370	.6370	.4481	.4444	.4444	.6333
Heart-c	.8133	.8196	.8133	.8133	.8196	.8196
Heart-h	.6492	.6395	.6079	.6079	.6079	.6254
Heart-statlog	.5400	.8133	.5067	.6267	.7600	.8533
Hepatitis	.7033	.6867	.6333	.6700	.6867	.6867
Iris	.5405	.5271	.4257	.4324	.4324	.5338
Labor	.7253	.7122	.3749	.3749	.3763	.7253
Lymph	.6889	.7111	.3889	.4000	.4000	.7111
Postoperative-patient-data	.3230	.6883	.3230	.3230	.3230	.7815
Solar-flare-c	.8505	.8505	.0403	.0403	.0993	.8487
Tic-tac-toe	.6993	.6993	.3236	.3194	.6722	.6993
Titanic	.1434	.1343	.1000	.1111	.1313	.1343
Vowel	.8240	.8240	.2990	.2990	.2990	.8240
Yeast	.3302	.3147	.2116	.2197	.2682	.3167
Zoo	.8527	.8118	.7818	.8118	.8218	.8218

Table A.3.: Accuracy using the 25 most confident local patterns generated by CHARM and the decoding methods Bayesian decoding (BD), best rule (BR), inverse weighted voting (IV), linear weighted voting (LV), voting (V), and weighted voting (WV).

Data Set	BD	BR	IV	LV	V	WV
Anneal.orig	.7617	.7617	.7617	.7617	.7617	.7617
Autos	.4402	.4402	.4402	.4402	.4402	.4402
Balance-scale	.7426	.7409	.1788	.3324	.6962	.7442
Breast-cancer	.6994	.6994	.6994	.6994	.6994	.6994
Breast-w	.5509	.5509	.5509	.5509	.5509	.5509
Bridges Version 1	.7002	.7002	.7002	.7002	.7002	.7002
Cars	.7459	.7426	.7491	.7459	.7426	.7426
CMC	.4270	.4270	.4270	.4270	.4270	.4270
Diabetes	.7412	.7411	.7023	.7412	.7441	.7441
Ecoli	.5420	.5420	.5281	.5420	.5420	.5420
Glass	.6741	.6741	.6778	.6778	.6778	.6778
Heart-c	.7938	.7938	.7938	.7938	.7938	.7938
Heart-h	.7752	.7752	.7752	.7752	.7752	.7752
Heart-statlog	.5667	.8800	.6267	.7867	.9000	.9067
Hepatitis	.6833	.6833	.6833	.7367	.6833	.6833
Iris	.7152	.7152	.7152	.7152	.7152	.7152
Labor	.6615	.6615	.6615	.6615	.6615	.6615
Lymph	.7111	.7111	.7111	.7111	.7111	.7111
Postoperative-patient-data	.3230	.6883	.3230	.3230	.3230	.7815
Solar-flare-c	.8534	.8534	.8534	.8534	.8534	.8534
Tic-tac-toe	.7651	.7651	.7630	.7630	.7630	.7651
Titanic	.2020	.1798	.1798	.1798	.1798	.1798
Vowel	.6552	.6552	.6552	.6552	.6552	.6552
Yeast	.4239	.4239	.4056	.4164	.4245	.4239
Zoo	.4064	.4064	.4064	.4064	.4064	.4064

Table A.4.: Accuracy using the 25 local patterns generated by CHARM and selected by joint entropy, and the decoding methods Bayesian decoding (BD), best rule (BR), inverse weighted voting (IV), linear weighted voting (LV), voting (V), and weighted voting (WV).

Data Set	BD	BR	IV	LV	V	WV
Anneal.orig	.7517	.7617	.7617	.7617	.7617	.7617
Autos	.3805	.3652	.2588	.3069	.3517	.3705
Balance-scale	.6450	.6373	.0768	.1278	.4674	.6515
Breast-cancer	.7030	.7030	.5041	.5181	.5424	.6823
Breast-w	.5127	.5227	.2409	.4300	.4391	.4482
Bridges Version 1	.7002	.7025	.2875	.4594	.6498	.6997
Cars	.6005	.6401	.3761	.4354	.4981	.6467
CMC	.4230	.4202	.3360	.3598	.3904	.4263
Diabetes	.6078	.6492	.2494	.2652	.4667	.6701
Ecoli	.5385	.5426	.2242	.3229	.5011	.5147
Glass	.6074	.7000	.3222	.3444	.4852	.7111
Heart-c	.7742	.8058	.5688	.6992	.7996	.8058
Heart-h	.7287	.7860	.2616	.3197	.4083	.7826
Heart-statlog	.7067	.9067	.5267	.6067	.7600	.9067
Hepatitis	.5600	.5400	.6300	.6133	.6133	.5567
Iris	.5348	.4871	.3938	.4071	.4138	.4938
Labor	.6719	.6563	.3815	.3854	.4599	.6563
Lymph	.7111	.7333	.5778	.6556	.7222	.7333
Postoperative-patient-data	.3230	.6883	.3230	.3230	.3230	.7815
Solar-flare-c	.8516	.8516	.5077	.5089	.8271	.8516
Tic-tac-toe	.6576	.6576	.3674	.3957	.6545	.6534
Titanic	.1677	.1576	.0929	.0980	.1222	.1434
Vowel	.6724	.6695	.6138	.6181	.6366	.6681
Yeast	.3322	.3524	.0526	.1059	.3160	.3288
Zoo	.4855	.4855	.4855	.4855	.4855	.4855

Table A.5.: Accuracy using all confident local patterns generated by CHARM and the decoding methods Bayesian decoding (BD), best rule (BR), inverse weighted voting (IV), linear weighted voting (LV), voting (V), and weighted voting (WV).

Data Set	BD	BR	IV	LV	V	WV
Anneal.orig	.8207	.7817	.7962	.7617	.7617	.7617
Autos	.5117	.6298	.6305	.5612	.5712	.5910
Balance-scale	.7379	.7425	.7330	.7634	.7538	.7442
Breast-cancer	.7030	.7101	.7268	.7484	.7518	.7518
Breast-w	.6273	.5509	.6000	.6182	.6082	.6082
Bridges Version 1	.7002	.7002	.7118	.7002	.7002	.7002
Cars	.7326	.8216	.7490	.8346	.8247	.8182
CMC	.4888	.4806	.4596	.4691	.4860	.4867
Diabetes	.6760	.6493	.7113	.7767	.7799	.7799
Ecoli	.6786	.5563	.6695	.6693	.6924	.6690
Glass	.7519	.7852	.7704	.8370	.8370	.8370
Heart-c	.8008	.7996	.8000	.7938	.7938	.7938
Heart-h	.6395	.8136	.7684	.8237	.8305	.8305
Heart-statlog	.6000	.8667	.8267	.8800	.8933	.8933
Hepatitis	.7933	.7233	.6800	.6467	.6633	.7333
Iris	.6814	.8033	.7705	.7900	.8038	.7900
Labor	.7174	.7344	.6797	.6836	.7201	.7201
Lymph	.7111	.7111	.7111	.7111	.7111	.7111
Postoperative-patient-data	.3230	.6883	.3294	.7828	.7524	.7524
Solar-flare-c	.8493	.8516	.8522	.8528	.8516	.8516
Tic-tac-toe	.7411	.9624	.6690	.7161	.7338	.7370
Titanic	.6485	.5707	.7232	.7222	.6869	.6768
Vowel	.6552	.8927	.9470	.9657	.9699	.9699
Yeast	.4610	.4778	.5014	.4987	.4846	.4805
Zoo	.9318	.9118	.9218	.9318	.9318	.9318

Table A.6.: Accuracy using all local patterns generated by BSD and the decoding methods Bayesian decoding (BD), best rule (BR), inverse weighted voting (IV), linear weighted voting (LV), voting (V), and weighted voting (WV).

Data Set	BD	BR	IV	LV	V	WV
Anneal.orig	.6603	.7695	.2561	.4197	.5823	.8742
Autos	.5655	.6098	.4676	.5898	.6298	.6100
Balance-scale	.6947	.7313	.0944	.2721	.6339	.7265
Breast-cancer	.6888	.7169	.6395	.6990	.7132	.7273
Breast-w	.3909	.5709	.1809	.3536	.5800	.6455
Bridges Version 1	.7002	.7002	.1522	.3559	.7078	.8432
Cars	.8144	.8180	.6897	.8114	.8212	.8311
CMC	.4690	.4820	.3197	.3734	.4453	.4799
Diabetes	.4373	.6191	.2448	.5561	.7563	.7651
Ecoli	.6128	.4868	.2381	.4297	.4859	.6407
Glass	.8074	.8185	.7185	.7778	.8259	.8185
Heart-c	.6517	.8129	.6058	.7108	.8133	.8321
Heart-h	.8269	.8000	.7654	.8199	.8236	.8136
Heart-statlog	.3333	.9067	.9333	.9200	.9133	.9200
Hepatitis	.7067	.7567	.6000	.6900	.7067	.7067
Iris	.3500	.7829	.6629	.7638	.8171	.8238
Labor	.6614	.7201	.5429	.6589	.7044	.7240
Lymph	.7222	.6889	.1889	.3000	.4222	.4889
Postoperative-patient-data	.3230	.7751	.3230	.4676	.7129	.7733
Solar-flare-c	.8516	.8131	.2494	.4072	.5690	.7711
Tic-tac-toe	.7432	.9843	.5387	.6368	.7015	.7411
Titanic	.4515	.4869	.1434	.3273	.4747	.5020
Vowel	.8311	.9055	.9671	.9628	.9570	.9528
Yeast	.4501	.5061	.0546	.3066	.5552	.5552
Zoo	.9318	.9318	.5036	.8436	.9618	.9418

Table A.7.: Accuracy using the 25 local patterns generated by BSD and selected by exclusive coverage, and the decoding methods Bayesian decoding (BD), best rule (BR), inverse weighted voting (IV), linear weighted voting (LV), voting (V), and weighted voting (WV).

Data Set	BD	BR	IV	LV	V	WV
Anneal.orig	.8319	.8386	.7173	.7173	.7751	.8375
Autos	.5314	.5419	.5169	.5169	.5169	.5562
Balance-scale	.6339	.6753	.2256	.2847	.4400	.6674
Breast-cancer	.7065	.7097	.7099	.7064	.7065	.7099
Breast-w	.5045	.4309	.4409	.4409	.4409	.4409
Bridges Version 1	.7123	.5347	.3704	.3744	.3761	.5306
Cars	.7624	.7624	.7591	.7624	.7624	.7624
CMC	.4535	.4439	.4148	.4148	.4148	.4439
Diabetes	.6010	.4307	.2767	.2767	.3213	.4249
Ecoli	.5056	.3457	.3171	.3171	.3266	.3504
Glass	.7704	.7741	.7741	.7741	.7741	.7741
Heart-c	.7288	.7475	.7038	.7038	.7104	.7104
Heart-h	.8169	.8238	.8169	.8169	.8169	.8169
Heart-statlog	.4667	.4867	.5467	.5733	.5933	.5467
Hepatitis	.5133	.5333	.5633	.5467	.5467	.5133
Iris	.6895	.7029	.6895	.6895	.6895	.6895
Labor	.7253	.6656	.6642	.6642	.6642	.6695
Lymph	.6889	.5556	.4889	.5000	.5000	.5222
Postoperative-patient-data	.7733	.7733	.7097	.7097	.7210	.7210
Solar-flare-c	.8511	.7827	.7652	.7652	.7658	.7827
Tic-tac-toe	.8871	.9007	.7337	.7452	.7807	.8756
Titanic	.2404	.2495	.2131	.2131	.2364	.2495
Vowel	.9213	.9242	.9184	.9184	.9227	.9242
Yeast	.4016	.3423	.2655	.2662	.3309	.3444
Zoo	.8227	.7536	.7227	.7227	.7327	.7527

Table A.8.: Accuracy using the 25 most confident local patterns generated by BSD and the decoding methods Bayesian decoding (BD), best rule (BR), inverse weighted voting (IV), linear weighted voting (LV), voting (V), and weighted voting (WV).

Data Set	BD	BR	IV	LV	V	WV
Anneal.orig	.7617	.7617	.7617	.7617	.7617	.7617
Autos	.3410	.3410	.3410	.3410	.3410	.3410
Balance-scale	.7186	.7265	.6867	.7330	.7282	.7282
Breast-cancer	.7239	.7239	.7273	.7273	.7273	.7239
Breast-w	.5700	.5700	.5700	.5700	.5700	.5700
Bridges Version 1	.7002	.7002	.7002	.7002	.7002	.7002
Cars	.7751	.7751	.7784	.7784	.7784	.7784
CMC	.4270	.4270	.4270	.4270	.4270	.4270
Diabetes	.6190	.6190	.6190	.6190	.6190	.6190
Ecoli	.4861	.4816	.4816	.4861	.4861	.4861
Glass	.7667	.7667	.7704	.7667	.7667	.7667
Heart-c	.7738	.7871	.7871	.7871	.7871	.7871
Heart-h	.7795	.7795	.7795	.7795	.7795	.7795
Heart-statlog	.7333	.7467	.7600	.7800	.7800	.7800
Hepatitis	.6633	.6633	.6633	.6633	.6633	.6633
Iris	.5476	.5543	.5543	.5543	.5543	.5543
Labor	.6511	.6511	.6511	.6511	.6511	.6511
Lymph	.7111	.7111	.7111	.7111	.7111	.7111
Postoperative-patient-data	.7733	.7751	.7792	.7860	.7837	.7833
Solar-flare-c	.8511	.8511	.8511	.8511	.8511	.8511
Tic-tac-toe	.7662	.7662	.7651	.7651	.7651	.7662
Titanic	.1374	.1030	.1030	.1030	.1030	.1030
Vowel	.6552	.6552	.6552	.6552	.6552	.6552
Yeast	.3335	.3335	.3335	.3335	.3335	.3335
Zoo	.4064	.4064	.4064	.4064	.4064	.4064

Table A.9.: Accuracy using the 25 local patterns generated by BSD and selected by joint entropy, and the decoding methods: Bayesian decoding (BD), best rule (BR), inverse weighted voting (IV), linear weighted voting (LV), voting (V), and weighted voting (WV).

Data Set	BD	BR	IV	LV	V	WV
Anneal.orig	.6648	.2483	.2294	.2294	.2305	.2316
Autos	.3410	.3267	.3410	.3410	.3410	.3410
Balance-scale	.5699	.6548	.1667	.4001	.5878	.6595
Breast-cancer	.7206	.7276	.6780	.6814	.6884	.7169
Breast-w	.4645	.4273	.4473	.4473	.4464	.4364
Bridges Version 1	.7031	.6823	.3321	.4501	.5532	.6915
Cars	.5838	.5838	.5838	.5838	.5838	.5838
CMC	.4745	.4644	.4488	.4521	.4562	.4610
Diabetes	.4879	.4758	.3487	.4406	.4643	.4758
Ecoli	.4920	.4024	.2948	.3978	.3976	.3976
Glass	.6185	.6185	.6185	.6185	.6185	.6185
Heart-c	.6563	.6958	.5717	.5917	.6371	.6767
Heart-h	.6395	.6395	.6395	.6395	.6395	.6395
Heart-statlog	.6067	.6133	.5933	.6267	.6533	.6133
Hepatitis	.4233	.4567	.4400	.4233	.4233	.4233
Iris	.5695	.5695	.5695	.5695	.5695	.5695
Labor	.6576	.6642	.6485	.6485	.6538	.6603
Lymph	.7111	.6222	.5444	.5556	.5778	.6222
Postoperative-patient-data	.3230	.7769	.5880	.7324	.7833	.7833
Solar-flare-c	.8516	.7027	.5684	.5970	.6332	.7027
Tic-tac-toe	.7234	.7077	.5804	.5930	.7046	.7078
Titanic	.2141	.1899	.1394	.1606	.1838	.1899
Vowel	.6552	.6552	.6552	.6552	.6552	.6552
Yeast	.3302	.3180	.1915	.2548	.3194	.3160
Zoo	.4064	.4064	.4064	.4064	.4064	.4064

Table A.10.: Accuracy using all confident local patterns generated by BSD and the decoding methods: Bayesian decoding (BD), best rule (BR), inverse weighted voting (IV), linear weighted voting (LV), voting (V), and weighted voting (WV).

Data Set	BD	BR	IV	LV	V	WV
Anneal.orig	.8653	.7817	.8151	.8608	.8708	.8719
Autos	.5755	.6098	.5314	.5705	.6195	.6048
Balance-scale	.7203	.7329	.7490	.7730	.7425	.7249
Breast-cancer	.6851	.7169	.6712	.7167	.7204	.7203
Breast-w	.5027	.5618	.5700	.5909	.6182	.6273
Bridges Version 1	.7002	.7002	.8172	.8090	.7755	.7355
Cars	.8144	.8180	.6897	.8114	.8212	.8311
CMC	.4868	.4725	.4392	.4643	.4738	.4833
Diabetes	.4463	.6191	.7410	.8004	.7680	.7590
Ecoli	.6268	.4868	.5136	.6223	.6359	.6361
Glass	.8074	.8185	.7185	.7778	.8259	.8185
Heart-c	.6904	.8129	.7029	.8008	.8321	.8129
Heart-h	.8269	.8000	.7654	.8199	.8236	.8136
Heart-statlog	.3333	.9067	.9067	.9133	.9200	.9200
Hepatitis	.7067	.7567	.6000	.6900	.7067	.7067
Iris	.3500	.7829	.6762	.7705	.8171	.8238
Labor	.6666	.7201	.6224	.7097	.7136	.7292
Lymph	.6778	.7000	.4778	.5333	.5889	.6333
Postoperative-patient-data	.3230	.7751	.3230	.7515	.7833	.7833
Solar-flare-c	.8516	.8511	.8511	.8511	.8511	.8511
Tic-tac-toe	.7870	.9843	.6827	.6868	.7390	.7494
Titanic	.4465	.4283	.3929	.4202	.4293	.4293
Vowel	.8311	.9055	.9671	.9628	.9570	.9528
Yeast	.5209	.5135	.5330	.5162	.5249	.5229
Zoo	.9218	.9318	.7936	.9136	.9618	.9418

B Probability Estimation: Experiments 1 - Detailed Experimental Results

Table B.1.: Basic probability estimation using patterns generated by ordered, unpruned JRip.

Name	Jrip	P	L	M=2	M=5	M=10
Anneal.orig	.9210	.9172	.9196	.9199	.9199	.9189
Audiology	.8630	.8449	.8323	.8396	.8388	.8310
Autos	.9042	.9072	.8998	.9071	.9041	.9030
Balance-scale	.8233	.8013	.8205	.8203	.8209	.8209
Breast-cancer	.5908	.5768	.5781	.5780	.5778	.5768
Breast-w	.9280	.9298	.9351	.9351	.9351	.9351
Colic	.7357	.7387	.7474	.7475	.7475	.7475
Credit-a	.8419	.8487	.8608	.8607	.8607	.8607
Credit-g	.5847	.5874	.5868	.5868	.5868	.5868
Diabetes	.6416	.6538	.6554	.6554	.6554	.6554
Glass	.8059	.8034	.7895	.7941	.7929	.7919
Heart-c	.7621	.7647	.7963	.7963	.7963	.7963
Heart-h	.7276	.7365	.7580	.7580	.7580	.7580
Heart-statlog	.7626	.7592	.8065	.8065	.8065	.8065
Hepatitis	.6791	.6608	.6599	.6599	.6599	.6599
Hypothyroid	.9713	.9734	.9743	.9744	.9743	.9734
Ionosphere	.8842	.8853	.9026	.9026	.9026	.9026
Iris	.9565	.8888	.8894	.8895	.8893	.8887
Kr-vs-kp	.9931	.9941	.9948	.9948	.9948	.9948
Labor	.8122	.8000	.7939	.7926	.7926	.7926
Lymph	.7500	.7388	.7480	.7459	.7437	.7490
Primary-tumor	.6490	.6358	.6147	.6448	.6409	.6422
Segment	.9827	.9637	.9665	.9660	.9666	.9662
Sick	.9221	.9276	.9294	.9293	.9293	.9293
Sonar	.7737	.7712	.7836	.7832	.7832	.7832
Soybean	.9624	.9713	.9656	.9735	.9675	.9671
Vehicle	.7723	.7995	.8113	.8117	.8109	.8118
Vote	.9518	.9535	.9552	.9552	.9547	.9530
Vowel	.8841	.9057	.9091	.9095	.9110	.9095
Zoo	.9155	.8989	.9023	.9081	.9068	.8994

Table B.2.: Probability estimation by shrinkage using patterns generated by ordered, unpruned JRip.

Name	Jrip	P	L	M=2	M=5	M=10
Anneal.orig	.9210	.9197	.9187	.9191	.9203	.9192
Audiology	.8630	.8433	.8364	.8437	.8414	.8324
Autos	.9042	.9005	.8908	.9016	.9022	.8982
Balance-scale	.8233	.8116	.8116	.8111	.8123	.8152
Breast-cancer	.5908	.5805	.5803	.5803	.5794	.5788
Breast-w	.9280	.9294	.9302	.9310	.9323	.9332
Colic	.7357	.7414	.7457	.7458	.7449	.7459
Credit-a	.8419	.8571	.8592	.8594	.8622	.8638
Credit-g	.5847	.5866	.5866	.5866	.5869	.5870
Diabetes	.6416	.6559	.6560	.6559	.6559	.6555
Glass	.8059	.7947	.7866	.7966	.7986	.7947
Heart-c	.7621	.7747	.7774	.7773	.7798	.7890
Heart-h	.7276	.7554	.7570	.7554	.7565	.7573
Heart-statlog	.7626	.7807	.7824	.7827	.7899	.7908
Hepatitis	.6791	.6615	.6627	.6645	.6635	.6626
Hypothyroid	.9713	.9738	.9738	.9739	.9741	.9736
Ionosphere	.8842	.8972	.8996	.8993	.9004	.9017
Iris	.9565	.8760	.8783	.8783	.8782	.8777
Kr-vs-kp	.9931	.9943	.9943	.9943	.9943	.9943
Labor	.8122	.8101	.8101	.8061	.7953	.7831
Lymph	.7500	.7482	.7450	.7482	.7457	.7458
Primary-tumor	.6490	.6523	.6383	.6563	.6531	.6623
Segment	.9827	.9439	.9431	.9440	.9431	.9430
Sick	.9221	.9286	.9286	.9287	.9290	.9290
Sonar	.7737	.7786	.7780	.7778	.7794	.7808
Soybean	.9624	.9724	.9709	.9721	.9726	.9714
Vehicle	.7723	.8110	.8158	.8134	.8157	.8190
Vote	.9518	.9502	.9494	.9491	.9525	.9559
Vowel	.8841	.9086	.9061	.9103	.9104	.9074
Zoo	.9155	.9156	.8969	.8999	.8953	.8901

Table B.3.: Basic probability estimation using patterns generated by ordered, pruned JRip.

Name	Jrip	P	L	M=2	M=5	M=10
Anneal.orig	.9422	.9383	.9364	.9370	.9360	.9352
Audiology	.9071	.8650	.8097	.8517	.8391	.8341
Autos	.8500	.8330	.8212	.8286	.8229	.8213
Balance-scale	.8523	.8119	.8150	.8146	.8160	.8160
Breast-cancer	.5976	.5960	.5960	.5963	.5980	.5982
Breast-w	.9730	.9649	.9645	.9643	.9636	.9612
Colic	.8231	.8014	.8044	.8090	.8131	.8163
Credit-a	.8738	.8720	.8733	.8736	.8745	.8753
Credit-g	.5927	.6129	.6128	.6131	.6133	.6132
Diabetes	.7389	.7337	.7338	.7339	.7339	.7338
Glass	.8027	.8137	.8217	.8200	.8199	.8203
Heart-c	.8310	.8372	.8428	.8420	.8453	.8473
Heart-h	.7582	.7392	.7396	.7397	.7412	.7415
Heart-statlog	.7805	.7916	.7904	.7903	.7909	.7899
Hepatitis	.6637	.5996	.5996	.5995	.5987	.5972
Hypothyroid	.9877	.9895	.9899	.9900	.9900	.9898
Ionosphere	.8995	.9041	.9068	.9083	.9097	.9103
Iris	.9735	.8884	.8904	.8904	.8904	.8904
Kr-vs-kp	.9949	.9943	.9943	.9943	.9942	.9941
Labor	.7791	.7824	.7824	.7811	.7676	.7459
Lymph	.7947	.7953	.7880	.7900	.7793	.7766
Primary-tumor	.6420	.6261	.6215	.6297	.6271	.6294
Segment	.9878	.9530	.9534	.9535	.9532	.9531
Sick	.9481	.9493	.9496	.9498	.9500	.9500
Sonar	.7591	.7404	.7420	.7430	.7463	.7445
Soybean	.9813	.9798	.9676	.9777	.9714	.9688
Vehicle	.8549	.8426	.8436	.8440	.8433	.8423
Vote	.9417	.9485	.9485	.9485	.9485	.9485
Vowel	.9095	.9000	.8977	.9036	.9046	.8982
Zoo	.9247	.8888	.8871	.8948	.8947	.8891

Table B.4.: Probability estimation by shrinkage using patterns generated by ordered, pruned JRip.

Name	Jrip	P	L	M=2	M=5	M=10
Anneal.orig	.9422	.9366	.9356	.9365	.9366	.9357
Audiology	.9071	.8539	.7762	.8399	.8260	.8012
Autos	.8500	.8364	.8292	.8303	.8302	.8187
Balance-scale	.8523	.8095	.8101	.8101	.8107	.8112
Breast-cancer	.5976	.5969	.5970	.5973	.5990	.6024
Breast-w	.9730	.9558	.9557	.9558	.9566	.9569
Colic	.8231	.8083	.8154	.8154	.8154	.8156
Credit-a	.8738	.8742	.8735	.8739	.8733	.8745
Credit-g	.5927	.6119	.6120	.6121	.6120	.6117
Diabetes	.7389	.7357	.7360	.7360	.7358	.7360
Glass	.8027	.8095	.8250	.8178	.8169	.8121
Heart-c	.8310	.8176	.8183	.8184	.8231	.8250
Heart-h	.7582	.7421	.7403	.7421	.7422	.7412
Heart-statlog	.7805	.7764	.7760	.7756	.7754	.7733
Hepatitis	.6637	.5962	.5962	.5957	.5949	.5864
Hypothyroid	.9877	.9900	.9900	.9902	.9902	.9900
Ionosphere	.8995	.9088	.9086	.9092	.9099	.9093
Iris	.9735	.8886	.8907	.8905	.8907	.8907
Kr-vs-kp	.9949	.9931	.9932	.9932	.9937	.9938
Labor	.7791	.7554	.7608	.7635	.7595	.7446
Lymph	.7947	.7671	.7722	.7732	.7734	.7741
Primary-tumor	.6420	.6237	.6270	.6224	.6223	.6280
Segment	.9878	.9322	.9325	.9322	.9325	.9331
Sick	.9481	.9491	.9492	.9492	.9496	.9499
Sonar	.7591	.7341	.7370	.7374	.7400	.7437
Soybean	.9813	.9699	.9647	.9695	.9669	.9665
Vehicle	.8549	.8389	.8428	.8418	.8427	.8440
Vote	.9417	.9472	.9472	.9472	.9472	.9472
Vowel	.9095	.8914	.8914	.8916	.8932	.8918
Zoo	.9247	.9086	.8947	.9024	.9012	.8931

Table B.5.: Basic probability estimation using patterns generated by unordered, unpruned JRip.

Name	Jrip	P	L	M=2	M=5	M=10
Anneal.orig	.9210	.9869	.9898	.9932	.9930	.9930
Audiology	.8630	.9105	.8773	.9087	.9034	.8922
Autos	.9042	.9162	.9264	.9270	.9286	.9302
Balance-scale	.8233	.8740	.9084	.9082	.9086	.9082
Breast-cancer	.5908	.6077	.6329	.6327	.6325	.6320
Breast-w	.9280	.9588	.9532	.9530	.9530	.9530
Colic	.7357	.8348	.8546	.8553	.8553	.8586
Credit-a	.8419	.8899	.9129	.9129	.9129	.9129
Credit-g	.5847	.6953	.7160	.7160	.7160	.7160
Diabetes	.6416	.7600	.7834	.7834	.7834	.7834
Glass	.8059	.8096	.8083	.8083	.8084	.8086
Heart-c	.7621	.7901	.8610	.8606	.8606	.8606
Heart-h	.7276	.7891	.8510	.8530	.8492	.8523
Heart-statlog	.7626	.7880	.8446	.8409	.8409	.8409
Hepatitis	.6791	.7743	.7990	.8022	.8022	.8022
Hypothyroid	.9713	.9912	.9941	.9940	.9940	.9940
Ionosphere	.8842	.9176	.9379	.9379	.9379	.9386
Iris	.9565	.9681	.9779	.9779	.9779	.9779
Kr-vs-kp	.9931	.9978	.9991	.9991	.9991	.9991
Labor	.8122	.8182	.7770	.7777	.7777	.7777
Lymph	.7500	.8431	.8914	.8875	.8808	.8838
Primary-tumor	.6490	.6821	.6714	.6929	.6942	.6914
Segment	.9827	.9911	.9973	.9973	.9973	.9973
Sick	.9221	.9577	.9814	.9817	.9817	.9817
Sonar	.7737	.8227	.8409	.8407	.8407	.8407
Soybean	.9624	.9791	.9820	.9847	.9843	.9846
Vehicle	.7723	.8512	.8883	.8882	.8882	.8882
Vote	.9518	.9726	.9822	.9826	.9827	.9827
Vowel	.8841	.9168	.9223	.9223	.9223	.9222
Zoo	.9155	.9643	.9648	.9843	.9843	.9869

Table B.6.: Probability estimation by shrinkage using patterns generated by unordered, unpruned JRip.

Name	Jrip	P	L	M=2	M=5	M=10
Anneal.orig	.9210	.9835	.9826	.9838	.9839	.9839
Audiology	.8630	.8875	.8739	.8951	.8935	.8887
Autos	.9042	.9146	.9142	.9142	.9183	.9262
Balance-scale	.8233	.8655	.8729	.8664	.8705	.8819
Breast-cancer	.5908	.5873	.6053	.5893	.6061	.6169
Breast-w	.9280	.9659	.9660	.9668	.9690	.9691
Colic	.7357	.8399	.8509	.8486	.8492	.8494
Credit-a	.8419	.9088	.9107	.9108	.9143	.9169
Credit-g	.5847	.7168	.7161	.7163	.7161	.7178
Diabetes	.6416	.7776	.7795	.7787	.7806	.7831
Glass	.8059	.8261	.8326	.8249	.8273	.8303
Heart-c	.7621	.8134	.8268	.8232	.8308	.8442
Heart-h	.7276	.8030	.8388	.8193	.8349	.8372
Heart-statlog	.7626	.8113	.8045	.8046	.8204	.8287
Hepatitis	.6791	.8175	.8190	.8205	.8172	.8159
Hypothyroid	.9713	.9936	.9928	.9937	.9932	.9932
Ionosphere	.8842	.9317	.9309	.9308	.9312	.9346
Iris	.9565	.9729	.9799	.9757	.9797	.9802
Kr-vs-kp	.9931	.9973	.9972	.9972	.9972	.9974
Labor	.8122	.8063	.8034	.8034	.7899	.7750
Lymph	.7500	.8522	.8572	.8481	.8524	.8784
Primary-tumor	.6490	.7066	.6897	.7119	.7107	.7115
Segment	.9827	.9892	.9897	.9895	.9896	.9897
Sick	.9221	.9790	.9840	.9790	.9795	.9802
Sonar	.7737	.8264	.8263	.8262	.8283	.8357
Soybean	.9624	.9809	.9795	.9808	.9810	.9807
Vehicle	.7723	.8787	.8807	.8795	.8811	.8838
Vote	.9518	.9675	.9680	.9679	.9748	.9783
Vowel	.8841	.9194	.9202	.9205	.9203	.9202
Zoo	.9155	.9651	.9702	.9816	.9823	.9883

Table B.7.: Basic probability estimation using patterns generated by unordered, pruned JRip.

Name	Jrip	P	L	M=2	M=5	M=10
Anneal.orig	.9422	.9898	.9849	.9889	.9875	.9842
Audiology	.9071	.9120	.8913	.8955	.8892	.8825
Autos	.8500	.8889	.8910	.8936	.8920	.8913
Balance-scale	.8523	.8880	.8986	.8948	.8997	.9013
Breast-cancer	.5976	.5620	.5565	.5565	.5565	.5603
Breast-w	.9730	.9624	.9626	.9625	.9626	.9611
Colic	.8231	.7815	.7990	.7933	.8012	.8125
Credit-a	.8738	.8763	.8773	.8771	.8789	.8811
Credit-g	.5927	.7023	.7027	.7030	.7034	.7051
Diabetes	.7389	.7401	.7418	.7421	.7413	.7387
Glass	.8027	.8189	.8213	.8191	.8243	.8275
Heart-c	.8310	.8267	.8266	.8292	.8278	.8295
Heart-h	.7582	.7390	.7353	.7368	.7360	.7348
Heart-statlog	.7805	.8057	.8155	.8164	.8232	.8237
Hepatitis	.6637	.7658	.7693	.7708	.7640	.7682
Hypothyroid	.9877	.9835	.9916	.9866	.9921	.9921
Ionosphere	.8995	.9184	.9213	.9224	.9259	.9264
Iris	.9735	.9746	.9752	.9750	.9752	.9752
Kr-vs-kp	.9949	.9989	.9990	.9989	.9989	.9981
Labor	.7791	.8372	.8149	.8122	.8122	.8095
Lymph	.7947	.8580	.8493	.8533	.8510	.8514
Primary-tumor	.6420	.7033	.6789	.7092	.7101	.7081
Segment	.9878	.9910	.9948	.9947	.9948	.9948
Sick	.9481	.9488	.9477	.9483	.9484	.9479
Sonar	.7591	.8269	.8270	.8272	.8240	.8236
Soybean	.9813	.9892	.9883	.9897	.9892	.9890
Vehicle	.8549	.8921	.8932	.8927	.8930	.8930
Vote	.9417	.9475	.9605	.9517	.9604	.9608
Vowel	.9095	.9211	.9238	.9255	.9248	.9240
Zoo	.9247	.9733	.9601	.9873	.9873	.9873

Table B.8.: Probability estimation by shrinkage using patterns generated by unordered, pruned JRip.

Name	Jrip	P	L	M=2	M=5	M=10
Anneal.orig	.9422	.9834	.9795	.9833	.9822	.9821
Audiology	.9071	.8889	.8782	.8926	.8853	.8807
Autos	.8500	.8823	.8893	.8877	.8889	.8892
Balance-scale	.8523	.8607	.8640	.8604	.8615	.8643
Breast-cancer	.5976	.5546	.5546	.5546	.5546	.5582
Breast-w	.9730	.9721	.9725	.9726	.9730	.9737
Colic	.8231	.8306	.8296	.8361	.8370	.8374
Credit-a	.8738	.8776	.8773	.8775	.8789	.8794
Credit-g	.5927	.7113	.7108	.7110	.7110	.7111
Diabetes	.7389	.7287	.7293	.7288	.7298	.7315
Glass	.8027	.8208	.8260	.8206	.8244	.8246
Heart-c	.8310	.8159	.8039	.8155	.8101	.8072
Heart-h	.7582	.7401	.7362	.7379	.7371	.7359
Heart-statlog	.7805	.8145	.8125	.8122	.8187	.8266
Hepatitis	.6637	.7899	.7929	.7901	.7948	.7891
Hypothyroid	.9877	.9935	.9927	.9937	.9930	.9930
Ionosphere	.8995	.9152	.9174	.9184	.9231	.9234
Iris	.9735	.9686	.9686	.9686	.9701	.9726
Kr-vs-kp	.9949	.9952	.9952	.9952	.9957	.9970
Labor	.7791	.8203	.8108	.8176	.8122	.8027
Lymph	.7947	.8321	.8331	.8358	.8419	.8558
Primary-tumor	.6420	.7006	.6945	.7039	.7062	.7075
Segment	.9878	.9892	.9900	.9897	.9899	.9901
Sick	.9481	.9338	.9376	.9353	.9369	.9374
Sonar	.7591	.8148	.8143	.8146	.8131	.8177
Soybean	.9813	.9815	.9811	.9809	.9811	.9808
Vehicle	.8549	.8914	.8900	.8904	.8901	.8900
Vote	.9417	.9557	.9565	.9565	.9562	.9583
Vowel	.9095	.9147	.9151	.9153	.9157	.9152
Zoo	.9247	.9892	.9692	.9892	.9890	.9886

C Probability Estimation: Experiments 1 - Averaged Experimental Results

Table C.1.: Average weighted AUC results of precision and the Laplace-estimate used as a basic probability estimation method or in liaison with shrinkage.

	Precision		Laplace	
Data Set	Basic	Shrinkage	Basic	Shrinkage
Anneal.orig	.9581	.9558	.9577	.9541
Audiology	.8831	.8684	.8526	.8412
Autos	.8863	.8835	.8846	.8809
Balance-scale	.8438	.8368	.8606	.8397
Breast-cancer	.5856	.5798	.5909	.5843
Breast-w	.9540	.9558	.9538	.9561
Colic	.7891	.8050	.8013	.8104
Credit-a	.8717	.8794	.8811	.8802
Credit-g	.6495	.6567	.6546	.6564
Diabetes	.7219	.7245	.7286	.7252
Glass	.8114	.8128	.8102	.8175
Heart-c	.8047	.8054	.8317	.8066
Heart-h	.7509	.7602	.7710	.7681
Heart-statlog	.7861	.7957	.8142	.7939
Hepatitis	.7001	.7162	.7070	.7177
Hypothyroid	.9844	.9878	.9875	.9873
Ionosphere	.9063	.9132	.9172	.9141
Iris	.9300	.9265	.9332	.9294
Kr-vs-kp	.9963	.9950	.9968	.9950
Labor	.8095	.7981	.7921	.7963
Lymph	.8088	.7999	.8192	.8019
Primary-tumor	.6618	.6708	.6466	.6624
Segment	.9747	.9636	.9780	.9638
Sick	.9458	.9476	.9520	.9499
Sonar	.7903	.7885	.7984	.7889
Soybean	.9799	.9762	.9759	.9740
Vehicle	.8463	.8550	.8591	.8573
Vote	.9555	.9551	.9616	.9553
Vowel	.9109	.9085	.9132	.9082
Zoo	.9313	.9446	.9286	.9328

Table C.2.: Average weighted AUC results of the *m*-Estimate (*m*=2, 5, or 10) used as a basic probability estimation method or in liaison with shrinkage.

	M=2		M=5		M=10	
Data Set	Basic	Shrink.	Basic	Shrink.	Basic	Shrink.
Anneal.orig	.9597	.9557	.9591	.9558	.9578	.9552
Audiology	.8739	.8678	.8676	.8615	.8600	.8508
Autos	.8891	.8834	.8869	.8849	.8864	.8831
Balance-scale	.8595	.8370	.8613	.8388	.8616	.8432
Breast-cancer	.5909	.5804	.5912	.5848	.5918	.5891
Breast-w	.9537	.9565	.9535	.9577	.9526	.9582
Colic	.8013	.8115	.8043	.8116	.8087	.8121
Credit-a	.8811	.8804	.8818	.8821	.8825	.8836
Credit-g	.6547	.6565	.6549	.6565	.6553	.6569
Diabetes	.7287	.7248	.7285	.7255	.7278	.7265
Glass	.8104	.8150	.8114	.8168	.8121	.8154
Heart-c	.8320	.8086	.8325	.8109	.8335	.8164
Heart-h	.7719	.7637	.7711	.7677	.7716	.7679
Heart-statlog	.8135	.7938	.8154	.8011	.8153	.8049
Hepatitis	.7081	.7177	.7062	.7176	.7069	.7135
Hypothyroid	.9863	.9879	.9876	.9876	.9873	.9874
Ionosphere	.9178	.9144	.9190	.9161	.9195	.9172
Iris	.9332	.9283	.9332	.9297	.9331	.9303
Kr-vs-kp	.9968	.9950	.9967	.9952	.9965	.9956
Labor	.7909	.7976	.7875	.7892	.7814	.7764
Lymph	.8192	.8013	.8137	.8033	.8152	.8135
Primary-tumor	.6692	.6736	.6681	.6731	.6678	.6773
Segment	.9779	.9638	.9780	.9638	.9778	.9640
Sick	.9523	.9481	.9524	.9488	.9522	.9491
Sonar	.7985	.7890	.7985	.7902	.7980	.7945
Soybean	.9814	.9758	.9781	.9754	.9774	.9748
Vehicle	.8591	.8563	.8588	.8574	.8588	.8592
Vote	.9595	.9552	.9616	.9577	.9613	.9599
Vowel	.9152	.9094	.9157	.9099	.9135	.9087
Zoo	.9436	.9433	.9433	.9420	.9407	.9400

Table C.3.: Average weighted AUC results of precision and the Laplace-estimate applied to unpruned or pruned pattern sets.

	Precision		Laplace	
Data Set	Unpruned	Pruned	Unpruned	Pruned
Anneal.orig	.9869	.9898	.9898	.9849
Audiology	.9105	.9120	.8773	.8913
Autos	.9162	.8889	.9264	.8910
Balance-scale	.8740	.8880	.9084	.8986
Breast-cancer	.6077	.5620	.6329	.5565
Breast-w	.9588	.9624	.9532	.9626
Colic	.8348	.7815	.8546	.7990
Credit-a	.8899	.8763	.9129	.8773
Credit-g	.6953	.7023	.7160	.7027
Diabetes	.7600	.7401	.7834	.7418
Glass	.8096	.8189	.8083	.8213
Heart-c	.7901	.8267	.8610	.8266
Heart-h	.7891	.7390	.8510	.7353
Heart-statlog	.7880	.8057	.8446	.8155
Hepatitis	.7743	.7658	.7990	.7693
Hypothyroid	.9912	.9835	.9941	.9916
Ionosphere	.9176	.9184	.9379	.9213
Iris	.9681	.9746	.9779	.9752
Kr-vs-kp	.9978	.9989	.9991	.9990
Labor	.8182	.8372	.7770	.8149
Lymph	.8431	.8580	.8914	.8493
Primary-tumor	.6821	.7033	.6714	.6789
Segment	.9911	.9910	.9973	.9948
Sick	.9577	.9488	.9814	.9477
Sonar	.8227	.8269	.8409	.8270
Soybean	.9791	.9892	.9820	.9883
Vehicle	.8512	.8921	.8883	.8932
Vote	.9726	.9475	.9822	.9605
Vowel	.9168	.9211	.9223	.9238
Zoo	.9643	.9733	.9648	.9601

Table C.4.: Average weighted AUC results of the m-Estimate (m=2, 5, or 10) applied to unpruned or pruned pattern sets.

	M=2		M=5		M=10	
Data Set	Unprun.	Pruned	Unprun.	Pruned	Unprun.	Pruned
Anneal.orig	.9932	.9889	.9930	.9875	.9930	.9842
Audiology	.9087	.8955	.9034	.8892	.8922	.8825
Autos	.9270	.8936	.9286	.8920	.9302	.8913
Balance-scale	.9082	.8948	.9086	.8997	.9082	.9013
Breast-cancer	.6327	.5565	.6325	.5565	.6320	.5603
Breast-w	.9530	.9625	.9530	.9626	.9530	.9611
Colic	.8553	.7933	.8553	.8012	.8586	.8125
Credit-a	.9129	.8771	.9129	.8789	.9129	.8811
Credit-g	.7160	.7030	.7160	.7034	.7160	.7051
Diabetes	.7834	.7421	.7834	.7413	.7834	.7387
Glass	.8083	.8191	.8084	.8243	.8086	.8275
Heart-c	.8606	.8292	.8606	.8278	.8606	.8295
Heart-h	.8530	.7368	.8492	.7360	.8523	.7348
Heart-statlog	.8409	.8164	.8409	.8232	.8409	.8237
Hepatitis	.8022	.7708	.8022	.7640	.8022	.7682
Hypothyroid	.9940	.9866	.9940	.9921	.9940	.9921
Ionosphere	.9379	.9224	.9379	.9259	.9386	.9264
Iris	.9779	.9750	.9779	.9752	.9779	.9752
Kr-vs-kp	.9991	.9989	.9991	.9989	.9991	.9981
Labor	.7777	.8122	.7777	.8122	.7777	.8095
Lymph	.8875	.8533	.8808	.8510	.8838	.8514
Primary-tumor	.6929	.7092	.6942	.7101	.6914	.7081
Segment	.9973	.9947	.9973	.9948	.9973	.9948
Sick	.9817	.9483	.9817	.9484	.9817	.9479
Sonar	.8407	.8272	.8407	.8240	.8407	.8236
Soybean	.9847	.9897	.9843	.9892	.9846	.9890
Vehicle	.8882	.8927	.8882	.8930	.8882	.8930
Vote	.9826	.9517	.9827	.9604	.9827	.9608
Vowel	.9223	.9255	.9223	.9248	.9222	.9240
Zoo	.9843	.9873	.9843	.9873	.9869	.9873

D Probability Estimation: Experiments 2 - Detailed Experimental Results

Table D.1.: Average weighted AUC of the bagged JRip and the employed probability estimation techniques: Bayesian decoding, 10 samples, unpruned rule sets (except for precision).

Name	Bagged Jrip	Precison	Laplace	M=2	M=5	M=10
Anneal.orig	.9826	.9947	.9952	.9952	.9948	.9948
Audiology	.9310	.9454	.9221	.9260	.9246	.9202
Autos	.9529	.9368	.9470	.9473	.9460	.9448
Balance-Scale	.9291	.9633	.9584	.9591	.9616	.9634
Breast-cancer	.6339	.6501	.6589	.6605	.6613	.6599
Breast-w	.9872	.9909	.9918	.9918	.9918	.9917
Colic	.8613	.8935	.8931	.8932	.8933	.8930
Credit-a	.9224	.9338	.9333	.9332	.9335	.9336
Credit-g	.7307	.7736	.7797	.7793	.7804	.7814
Diabetes	.7812	.8180	.8142	.8137	.8151	.8166
Glass	.8912	.8984	.8942	.8990	.8952	.8905
Heart-c	.8913	.9026	.9043	.9031	.9045	.9053
Heart-h	.8463	.8846	.8734	.8730	.8749	.8769
Heart-statlog	.8646	.8983	.9023	.9020	.9030	.9031
Hepatitis	.7622	.8481	.8417	.8420	.8425	.8402
Hypothyroid	.9929	.9957	.9981	.9977	.9977	.9977
Ionosphere	.9635	.9570	.9737	.9736	.9733	.9722
Iris	.9811	.9923	.9915	.9913	.9909	.9908
Kr-vs-kp	.9979	.9996	.9997	.9997	.9999	.9998
Labor	.8838	.8959	.9122	.9081	.9027	.8973
Lymph	.8945	.9273	.9229	.9258	.9247	.9235
Primary-tumor	.7316	.8034	.7490	.7884	.7849	.7776
Segment	.9993	.9986	.9988	.9989	.9989	.9988
Sick	.9831	.9964	.9965	.9963	.9963	.9963
Sonar	.9055	.9057	.9161	.9155	.9153	.9133
Soybean	.9937	.9955	.9925	.9948	.9947	.9944
Vehicle	.9289	.9301	.9302	.9300	.9301	.9302
Vote	.9646	.9925	.9928	.9928	.9928	.9926
Vowel	.9887	.9907	.9899	.9915	.9909	.9900
Zoo	.9495	.9910	.9623	.9876	.9819	.9753

Table D.2.: Average weighted AUC of the bagged JRip and the employed probability estimation techniques: best rule, 10 samples, unpruned rule sets (except for precision).

Name	Bagged Jrip	Precison	Laplace	M=2	M=5	M=10
Anneal.orig	.9826	.9946	.9925	.9937	.9937	.9937
Audiology	.9310	.9487	.9295	.9283	.9243	.9210
Autos	.9529	.9367	.9469	.9486	.9471	.9449
Balance-Scale	.9291	.9548	.9546	.9547	.9567	.9590
Breast-cancer	.6339	.6284	.6662	.6676	.6643	.6609
Breast-w	.9872	.9876	.9859	.9852	.9850	.9850
Colic	.8613	.8920	.8849	.8872	.8881	.8876
Credit-a	.9224	.9295	.9300	.9300	.9305	.9308
Credit-g	.7307	.7643	.7726	.7695	.7708	.7733
Diabetes	.7812	.8052	.8062	.8049	.8067	.8085
Glass	.8912	.8963	.8884	.8879	.8876	.8873
Heart-c	.8913	.8909	.9088	.9068	.9072	.9073
Heart-h	.8463	.8840	.8745	.8744	.8750	.8766
Heart-statlog	.8646	.8922	.8959	.8956	.8960	.8961
Hepatitis	.7622	.8471	.8434	.8487	.8505	.8512
Hypothyroid	.9929	.9937	.9954	.9957	.9957	.9957
Ionosphere	.9635	.9534	.9702	.9702	.9702	.9706
Iris	.9811	.9930	.9931	.9929	.9933	.9937
Kr-vs-kp	.9979	.9997	.9998	.9998	.9998	.9998
Labor	.8838	.8932	.9216	.9189	.9149	.9054
Lymph	.8945	.9245	.9242	.9256	.9258	.9247
Primary-tumor	.7316	.8045	.7529	.7786	.7755	.7738
Segment	.9993	.9987	.9988	.9988	.9988	.9988
Sick	.9831	.9932	.9943	.9949	.9949	.9947
Sonar	.9055	.9008	.9098	.9084	.9083	.9075
Soybean	.9937	.9955	.9914	.9941	.9940	.9937
Vehicle	.9289	.9248	.9266	.9262	.9269	.9276
Vote	.9646	.9821	.9870	.9871	.9871	.9869
Vowel	.9887	.9905	.9886	.9903	.9895	.9887
Zoo	.9495	.9936	.9703	.9928	.9936	.9919

Table D.3.: Average weighted AUC of the bagged JRip and the employed probability estimation techniques: macro averaging, 10 samples, unpruned rule sets (except for precision).

Name	Bagged Jrip	Precison	Laplace	M=2	M=5	M=10
Anneal.orig	.9826	.9974	.9960	.9972	.9972	.9972
Audiology	.9310	.9561	.9328	.9367	.9337	.9300
Autos	.9529	.9395	.9510	.9514	.9498	.9486
Balance-Scale	.9291	.9582	.9558	.9562	.9587	.9605
Breast-cancer	.6339	.6469	.6604	.6611	.6618	.6617
Breast-w	.9872	.9864	.9870	.9869	.9871	.9870
Colic	.8613	.8928	.8911	.8916	.8920	.8923
Credit-a	.9224	.9310	.9326	.9323	.9329	.9335
Credit-g	.7307	.7690	.7729	.7727	.7742	.7757
Diabetes	.7812	.8089	.8088	.8084	.8102	.8126
Glass	.8912	.8991	.8956	.9003	.8971	.8936
Heart-c	.8913	.8983	.9060	.9057	.9064	.9066
Heart-h	.8463	.8829	.8764	.8762	.8785	.8794
Heart-statlog	.8646	.8934	.8999	.8999	.9006	.9001
Hepatitis	.7622	.8417	.8498	.8509	.8521	.8504
Hypothyroid	.9929	.9939	.9960	.9963	.9962	.9962
Ionosphere	.9635	.9552	.9721	.9722	.9723	.9720
Iris	.9811	.9930	.9929	.9927	.9931	.9933
Kr-vs-kp	.9979	.9996	.9997	.9997	.9997	.9997
Labor	.8838	.8959	.9081	.9068	.9000	.8932
Lymph	.8945	.9268	.9277	.9278	.9290	.9275
Primary-tumor	.7316	.8108	.7615	.7925	.7905	.7863
Segment	.9993	.9989	.9990	.9990	.9990	.9990
Sick	.9831	.9947	.9940	.9943	.9942	.9940
Sonar	.9055	.9057	.9172	.9168	.9154	.9140
Soybean	.9937	.9957	.9929	.9949	.9948	.9946
Vehicle	.9289	.9294	.9291	.9288	.9291	.9293
Vote	.9646	.9829	.9861	.9862	.9858	.9857
Vowel	.9887	.9911	.9907	.9922	.9915	.9908
Zoo	.9495	.9944	.9723	.9943	.9941	.9913

Table D.4.: Average weighted AUC of the bagged JRip and the employed probability estimation techniques: micro averaging, 10 samples, unpruned rule sets (except for precision).

Name	Bagged Jrip	Precison	Laplace	M=2	M=5	M=10
Anneal.orig	.9826	.9970	.9954	.9966	.9968	.9969
Audiology	.9310	.9563	.9369	.9361	.9324	.9292
Autos	.9529	.9387	.9481	.9491	.9471	.9450
Balance-Scale	.9291	.9596	.9565	.9572	.9598	.9618
Breast-cancer	.6339	.6602	.6601	.6607	.6614	.6598
Breast-w	.9872	.9867	.9876	.9876	.9875	.9874
Colic	.8613	.8959	.8933	.8938	.8940	.8937
Credit-a	.9224	.9306	.9329	.9326	.9335	.9341
Credit-g	.7307	.7691	.7765	.7759	.7780	.7795
Diabetes	.7812	.8129	.8121	.8115	.8136	.8151
Glass	.8912	.9013	.8989	.9036	.9014	.8976
Heart-c	.8913	.8987	.9053	.9050	.9054	.9054
Heart-h	.8463	.8825	.8758	.8760	.8778	.8779
Heart-statlog	.8646	.8960	.9023	.9022	.9029	.9032
Hepatitis	.7622	.8422	.8504	.8529	.8552	.8526
Hypothyroid	.9929	.9938	.9962	.9963	.9963	.9962
Ionosphere	.9635	.9552	.9718	.9719	.9719	.9717
Iris	.9811	.9927	.9922	.9921	.9925	.9925
Kr-vs-kp	.9979	.9996	.9997	.9997	.9997	.9997
Labor	.8838	.9095	.9135	.9135	.9068	.9054
Lymph	.8945	.9286	.9283	.9293	.9301	.9283
Primary-tumor	.7316	.8104	.7653	.7963	.7942	.7895
Segment	.9993	.9988	.9990	.9990	.9990	.9989
Sick	.9831	.9951	.9937	.9941	.9940	.9938
Sonar	.9055	.9095	.9174	.9172	.9159	.9145
Soybean	.9937	.9952	.9928	.9945	.9946	.9945
Vehicle	.9289	.9304	.9304	.9300	.9304	.9305
Vote	.9646	.9832	.9862	.9863	.9860	.9854
Vowel	.9887	.9905	.9899	.9914	.9907	.9900
Zoo	.9495	.9939	.9720	.9943	.9938	.9913

Table D.5.: Average weighted AUC of the bagged JRip and the employed probability estimation techniques: Bayesian decoding, 20 samples, unpruned rule sets (except for precision).

Name	Bagged Jrip	Precison	Laplace	M=2	M=5	M=10
Anneal.orig	.9821	.9957	.9959	.9961	.9963	.9964
Audiology	.9415	.9428	.9276	.9262	.9244	.9219
Autos	.9573	.9418	.9475	.9497	.9478	.9450
Balance-Scale	.9344	.9630	.9598	.9607	.9633	.9654
Breast-cancer	.6213	.6651	.6619	.6604	.6637	.6666
Breast-w	.9888	.9919	.9920	.9919	.9919	.9919
Colic	.8664	.8942	.8945	.8945	.8940	.8928
Credit-a	.9265	.9384	.9371	.9370	.9373	.9373
Credit-g	.7438	.7832	.7842	.7840	.7843	.7847
Diabetes	.7935	.8233	.8208	.8207	.8221	.8230
Glass	.8975	.9084	.9029	.9093	.9072	.9032
Heart-c	.8952	.9070	.9079	.9072	.9081	.9095
Heart-h	.8536	.8892	.8755	.8742	.8761	.8767
Heart-statlog	.8769	.8993	.9028	.9029	.9031	.9028
Hepatitis	.7671	.8478	.8572	.8572	.8582	.8580
Hypothyroid	.9922	.9955	.9983	.9979	.9979	.9978
Ionosphere	.9621	.9582	.9726	.9726	.9717	.9708
Iris	.9910	.9931	.9922	.9922	.9923	.9923
Kr-vs-kp	.9982	.9997	.9998	.9998	.9998	.9998
Labor	.9135	.9108	.9243	.9257	.9189	.9122
Lymph	.9048	.9276	.9335	.9345	.9328	.9292
Primary-tumor	.7479	.8044	.7560	.7929	.7894	.7829
Segment	.9994	.9989	.9990	.9991	.9991	.9990
Sick	.9883	.9973	.9967	.9965	.9962	.9957
Sonar	.9223	.9111	.9200	.9199	.9180	.9161
Soybean	.9934	.9958	.9948	.9953	.9952	.9951
Vehicle	.9344	.9331	.9345	.9345	.9345	.9344
Vote	.9668	.9927	.9923	.9924	.9921	.9919
Vowel	.9955	.9931	.9928	.9940	.9935	.9929
Zoo	.9536	.9940	.9664	.9930	.9895	.9838

Table D.6.: Average weighted AUC of the bagged JRip and the employed probability estimation techniques: best rule, 20 samples, unpruned rule sets (except for precision).

Name	Bagged Jrip	Precison	Laplace	M=2	M=5	M=10
Anneal.orig	.9821	.9950	.9929	.9937	.9938	.9938
Audiology	.9415	.9490	.9381	.9362	.9326	.9295
Autos	.9573	.9418	.9488	.9513	.9496	.9466
Balance-Scale	.9344	.9581	.9572	.9576	.9607	.9630
Breast-cancer	.6213	.6596	.6699	.6637	.6652	.6664
Breast-w	.9888	.9888	.9889	.9890	.9890	.9890
Colic	.8664	.9045	.8888	.8886	.8892	.8874
Credit-a	.9265	.9329	.9338	.9338	.9341	.9341
Credit-g	.7438	.7766	.7760	.7730	.7749	.7768
Diabetes	.7935	.8138	.8185	.8166	.8172	.8186
Glass	.8975	.9101	.8943	.8963	.8949	.8946
Heart-c	.8952	.9027	.9091	.9095	.9101	.9096
Heart-h	.8536	.8846	.8758	.8758	.8764	.8773
Heart-statlog	.8769	.8942	.8992	.8996	.8998	.8988
Hepatitis	.7671	.8438	.8562	.8577	.8598	.8615
Hypothyroid	.9922	.9940	.9956	.9957	.9957	.9957
Ionosphere	.9621	.9543	.9713	.9714	.9711	.9704
Iris	.9910	.9937	.9931	.9931	.9932	.9933
Kr-vs-kp	.9982	.9997	.9998	.9998	.9998	.9998
Labor	.9135	.8973	.9243	.9257	.9257	.9243
Lymph	.9048	.9257	.9332	.9348	.9331	.9300
Primary-tumor	.7479	.8062	.7694	.7949	.7907	.7857
Segment	.9994	.9991	.9991	.9992	.9992	.9991
Sick	.9883	.9951	.9958	.9964	.9963	.9962
Sonar	.9223	.9172	.9136	.9127	.9112	.9102
Soybean	.9934	.9957	.9936	.9946	.9945	.9943
Vehicle	.9344	.9300	.9320	.9319	.9322	.9325
Vote	.9668	.9812	.9903	.9903	.9902	.9900
Vowel	.9955	.9932	.9917	.9933	.9927	.9919
Zoo	.9536	.9939	.9715	.9947	.9943	.9934

Table D.7.: Average weighted AUC of the bagged JRip and the employed probability estimation techniques: macro averaging, 20 samples, unpruned rule sets (except for precision).

Name	Bagged Jrip	Precison	Laplace	M=2	M=5	M=10
Anneal.orig	.9821	.9974	.9964	.9972	.9973	.9975
Audiology	.9415	.9534	.9395	.9368	.9334	.9296
Autos	.9573	.9431	.9517	.9540	.9518	.9495
Balance-Scale	.9344	.9603	.9578	.9592	.9615	.9635
Breast-cancer	.6213	.6675	.6643	.6630	.6668	.6692
Breast-w	.9888	.9901	.9882	.9882	.9881	.9880
Colic	.8664	.8983	.8948	.8944	.8943	.8939
Credit-a	.9265	.9360	.9357	.9357	.9362	.9361
Credit-g	.7438	.7808	.7797	.7791	.7802	.7812
Diabetes	.7935	.8184	.8175	.8172	.8187	.8202
Glass	.8975	.9104	.9048	.9100	.9079	.9055
Heart-c	.8952	.9051	.9076	.9070	.9084	.9089
Heart-h	.8536	.8837	.8770	.8761	.8768	.8779
Heart-statlog	.8769	.8970	.9022	.9019	.9011	.9007
Hepatitis	.7671	.8509	.8565	.8608	.8600	.8585
Hypothyroid	.9922	.9940	.9960	.9960	.9960	.9959
Ionosphere	.9621	.9563	.9716	.9716	.9713	.9705
Iris	.9910	.9935	.9927	.9927	.9928	.9931
Kr-vs-kp	.9982	.9996	.9996	.9996	.9996	.9996
Labor	.9135	.8959	.9176	.9189	.9135	.9081
Lymph	.9048	.9271	.9364	.9365	.9359	.9314
Primary-tumor	.7479	.8122	.7701	.8000	.7972	.7936
Segment	.9994	.9991	.9992	.9992	.9992	.9992
Sick	.9883	.9975	.9973	.9976	.9975	.9973
Sonar	.9223	.9136	.9170	.9172	.9159	.9140
Soybean	.9934	.9959	.9949	.9955	.9954	.9952
Vehicle	.9344	.9329	.9345	.9342	.9344	.9342
Vote	.9668	.9825	.9854	.9855	.9850	.9845
Vowel	.9955	.9935	.9931	.9944	.9938	.9933
Zoo	.9536	.9948	.9719	.9948	.9944	.9931

Table D.8.: Average weighted AUC of the bagged JRip and the employed probability estimation techniques: micro averaging, 20 samples, unpruned rule sets (except for precision).

Name	Bagged Jrip	Precison	Laplace	M=2	M=5	M=10
Anneal.orig	.9821	.9969	.9958	.9968	.9968	.9971
Audiology	.9415	.9542	.9418	.9364	.9329	.9294
Autos	.9573	.9441	.9498	.9517	.9492	.9464
Balance-Scale	.9344	.9607	.9586	.9602	.9627	.9647
Breast-cancer	.6213	.6735	.6628	.6595	.6634	.6657
Breast-w	.9888	.9907	.9886	.9886	.9884	.9881
Colic	.8664	.9013	.8963	.8965	.8963	.8945
Credit-a	.9265	.9371	.9357	.9356	.9366	.9369
Credit-g	.7438	.7827	.7839	.7833	.7845	.7851
Diabetes	.7935	.8216	.8205	.8202	.8222	.8234
Glass	.8975	.9125	.9075	.9134	.9124	.9098
Heart-c	.8952	.9052	.9068	.9061	.9074	.9076
Heart-h	.8536	.8863	.8764	.8746	.8755	.8770
Heart-statlog	.8769	.8975	.9026	.9026	.9032	.9035
Hepatitis	.7671	.8519	.8598	.8610	.8626	.8626
Hypothyroid	.9922	.9940	.9961	.9961	.9961	.9960
Ionosphere	.9621	.9571	.9718	.9718	.9713	.9705
Iris	.9910	.9927	.9919	.9919	.9920	.9922
Kr-vs-kp	.9982	.9995	.9996	.9996	.9996	.9996
Labor	.9135	.9122	.9243	.9230	.9203	.9135
Lymph	.9048	.9276	.9360	.9365	.9361	.9341
Primary-tumor	.7479	.8112	.7734	.8033	.8000	.7955
Segment	.9994	.9991	.9992	.9992	.9992	.9992
Sick	.9883	.9974	.9968	.9975	.9974	.9971
Sonar	.9223	.9127	.9211	.9208	.9191	.9176
Soybean	.9934	.9956	.9946	.9950	.9950	.9948
Vehicle	.9344	.9338	.9349	.9347	.9348	.9344
Vote	.9668	.9826	.9856	.9857	.9850	.9839
Vowel	.9955	.9929	.9925	.9937	.9932	.9926
Zoo	.9536	.9944	.9717	.9949	.9943	.9929

Table D.9.: Average weighted AUC of the bagged JRip and the employed probability estimation techniques: Bayesian decoding, 50 samples, unpruned rule sets (except for precision).

Name	Bagged Jrip	Precison	Laplace	M=2	M=5	M=10
Anneal.orig	.9898	.9956	.9957	.9958	.9959	.9957
Audiology	.9454	.9468	.9366	.9383	.9374	.9342
Autos	.9592	.9487	.9506	.9527	.9504	.9468
Balance-Scale	.9383	.9648	.9614	.9623	.9653	.9678
Breast-cancer	.6206	.6736	.6712	.6694	.6737	.6750
Breast-w	.9916	.9917	.9920	.9920	.9919	.9919
Colic	.8789	.8972	.8950	.8954	.8949	.8938
Credit-a	.9285	.9374	.9364	.9363	.9365	.9365
Credit-g	.7526	.7873	.7875	.7874	.7876	.7873
Diabetes	.8066	.8258	.8206	.8202	.8217	.8228
Glass	.8994	.9147	.9114	.9170	.9146	.9100
Heart-c	.8967	.9119	.9105	.9091	.9101	.9107
Heart-h	.8676	.8907	.8794	.8773	.8789	.8802
Heart-statlog	.8831	.9022	.8998	.8998	.9005	.9008
Hepatitis	.7887	.8585	.8544	.8549	.8554	.8526
Hypothyroid	.9927	.9973	.9984	.9979	.9979	.9979
Ionosphere	.9700	.9625	.9715	.9715	.9709	.9695
Iris	.9941	.9919	.9927	.9926	.9927	.9927
Kr-vs-kp	.9984	.9996	.9998	.9998	.9998	.9998
Labor	.9135	.9230	.9311	.9324	.9257	.9216
Lymph	.8994	.9377	.9382	.9392	.9368	.9352
Primary-tumor	.7555	.8120	.7550	.7926	.7898	.7846
Segment	.9996	.9993	.9991	.9991	.9991	.9991
Sick	.9913	.9973	.9969	.9963	.9961	.9959
Sonar	.9316	.9233	.9274	.9272	.9261	.9239
Soybean	.9947	.9959	.9949	.9953	.9952	.9948
Vehicle	.9370	.9351	.9361	.9362	.9360	.9357
Vote	.9739	.9933	.9931	.9932	.9930	.9926
Vowel	.9959	.9940	.9942	.9952	.9947	.9942
Zoo	.9479	.9942	.9676	.9942	.9924	.9881

Table D.10.: Average weighted AUC of the bagged JRip and the employed probability estimation techniques: best rule, 50 samples, unpruned rule sets (except for precision).

Name	Bagged Jrip	Precison	Laplace	M=2	M=5	M=10
Anneal.orig	.9898	.9953	.9933	.9936	.9938	.9940
Audiology	.9454	.9570	.9464	.9453	.9440	.9422
Autos	.9592	.9500	.9500	.9524	.9492	.9466
Balance-Scale	.9383	.9581	.9593	.9602	.9634	.9661
Breast-cancer	.6206	.6646	.6778	.6753	.6784	.6789
Breast-w	.9916	.9914	.9917	.9918	.9918	.9917
Colic	.8789	.9015	.8929	.8931	.8920	.8910
Credit-a	.9285	.9348	.9337	.9336	.9337	.9336
Credit-g	.7526	.7843	.7823	.7822	.7833	.7839
Diabetes	.8066	.8192	.8205	.8186	.8188	.8197
Glass	.8994	.9132	.9049	.9101	.9067	.9021
Heart-c	.8967	.9076	.9123	.9123	.9127	.9119
Heart-h	.8676	.8876	.8790	.8775	.8782	.8791
Heart-statlog	.8831	.9012	.9007	.8997	.9002	.8996
Hepatitis	.7887	.8450	.8496	.8514	.8549	.8565
Hypothyroid	.9927	.9940	.9958	.9960	.9960	.9960
Ionosphere	.9700	.9642	.9768	.9768	.9764	.9755
Iris	.9941	.9930	.9938	.9936	.9938	.9939
Kr-vs-kp	.9984	.9997	.9997	.9997	.9997	.9997
Labor	.9135	.9203	.9270	.9270	.9270	.9243
Lymph	.8994	.9407	.9382	.9397	.9392	.9380
Primary-tumor	.7555	.8161	.7707	.7982	.7932	.7885
Segment	.9996	.9993	.9992	.9992	.9992	.9992
Sick	.9913	.9977	.9961	.9966	.9965	.9963
Sonar	.9316	.9284	.9224	.9222	.9206	.9201
Soybean	.9947	.9957	.9940	.9948	.9947	.9944
Vehicle	.9370	.9331	.9353	.9352	.9351	.9349
Vote	.9739	.9873	.9923	.9925	.9922	.9919
Vowel	.9959	.9944	.9936	.9948	.9942	.9936
Zoo	.9479	.9948	.9739	.9952	.9950	.9942

Table D.11.: Average weighted AUC of the bagged JRip and the employed probability estimation techniques: macro averaging, 50 samples, unpruned rule sets (except for precision).

Name	Bagged Jrip	Precison	Laplace	M=2	M=5	M=10
Anneal.orig	.9898	.9975	.9964	.9971	.9973	.9974
Audiology	.9454	.9565	.9468	.9456	.9443	.9426
Autos	.9592	.9508	.9533	.9551	.9532	.9497
Balance-Scale	.9383	.9621	.9593	.9607	.9633	.9656
Breast-cancer	.6206	.6749	.6773	.6753	.6805	.6817
Breast-w	.9916	.9911	.9916	.9916	.9915	.9915
Colic	.8789	.8993	.8981	.8975	.8975	.8969
Credit-a	.9285	.9366	.9351	.9350	.9353	.9355
Credit-g	.7526	.7856	.7837	.7836	.7841	.7839
Diabetes	.8066	.8210	.8196	.8193	.8204	.8216
Glass	.8994	.9169	.9134	.9179	.9169	.9135
Heart-c	.8967	.9103	.9112	.9102	.9114	.9118
Heart-h	.8676	.8873	.8787	.8769	.8778	.8792
Heart-statlog	.8831	.9006	.8988	.8987	.8991	.8993
Hepatitis	.7887	.8537	.8544	.8554	.8549	.8506
Hypothyroid	.9927	.9967	.9960	.9960	.9960	.9959
Ionosphere	.9700	.9623	.9741	.9742	.9732	.9720
Iris	.9941	.9927	.9936	.9935	.9938	.9941
Kr-vs-kp	.9984	.9995	.9995	.9995	.9995	.9995
Labor	.9135	.9162	.9230	.9216	.9176	.9162
Lymph	.8994	.9389	.9393	.9400	.9383	.9368
Primary-tumor	.7555	.8201	.7688	.7999	.7976	.7934
Segment	.9996	.9993	.9993	.9994	.9993	.9993
Sick	.9913	.9975	.9976	.9977	.9976	.9975
Sonar	.9316	.9233	.9283	.9281	.9263	.9241
Soybean	.9947	.9960	.9948	.9955	.9952	.9948
Vehicle	.9370	.9352	.9361	.9362	.9361	.9359
Vote	.9739	.9877	.9928	.9929	.9924	.9916
Vowel	.9959	.9944	.9945	.9955	.9951	.9946
Zoo	.9479	.9948	.9741	.9953	.9949	.9942

Table D.12.: Average weighted AUC of the bagged JRip and the employed probability estimation techniques: micro averaging, 50 samples, unpruned rule sets (except for precision).

Name	Bagged Jrip	Precison	Laplace	M=2	M=5	M=10
Anneal.orig	.9898	.9970	.9958	.9967	.9969	.9970
Audiology	.9454	.9569	.9475	.9448	.9430	.9414
Autos	.9592	.9503	.9503	.9531	.9510	.9483
Balance-Scale	.9383	.9623	.9591	.9609	.9638	.9662
Breast-cancer	.6206	.6784	.6719	.6703	.6751	.6771
Breast-w	.9916	.9909	.9914	.9914	.9913	.9912
Colic	.8789	.9041	.8967	.8972	.8965	.8955
Credit-a	.9285	.9383	.9350	.9349	.9354	.9357
Credit-g	.7526	.7872	.7873	.7865	.7873	.7875
Diabetes	.8066	.8237	.8206	.8204	.8215	.8227
Glass	.8994	.9157	.9148	.9193	.9170	.9136
Heart-c	.8967	.9132	.9103	.9097	.9103	.9112
Heart-h	.8676	.8884	.8817	.8793	.8805	.8820
Heart-statlog	.8831	.9014	.8992	.8991	.9001	.9010
Hepatitis	.7887	.8577	.8526	.8542	.8534	.8532
Hypothyroid	.9927	.9967	.9961	.9961	.9961	.9960
Ionosphere	.9700	.9645	.9753	.9753	.9744	.9735
Iris	.9941	.9922	.9921	.9921	.9929	.9930
Kr-vs-kp	.9984	.9995	.9995	.9995	.9995	.9995
Labor	.9135	.9203	.9270	.9270	.9243	.9243
Lymph	.8994	.9403	.9407	.9420	.9400	.9378
Primary-tumor	.7555	.8173	.7705	.8010	.7988	.7956
Segment	.9996	.9993	.9993	.9994	.9993	.9993
Sick	.9913	.9973	.9975	.9976	.9975	.9974
Sonar	.9316	.9237	.9276	.9277	.9262	.9244
Soybean	.9947	.9957	.9949	.9951	.9950	.9948
Vehicle	.9370	.9355	.9364	.9365	.9363	.9360
Vote	.9739	.9882	.9931	.9931	.9925	.9915
Vowel	.9959	.9938	.9939	.9950	.9945	.9940
Zoo	.9479	.9942	.9739	.9950	.9948	.9941

Table D.13.: Average weighted AUC of the bagged JRip and the employed probability estimation techniques: Bayesian decoding, 100 samples, unpruned rule sets (except for precision).

Name	Bagged Jrip	Precison	Laplace	M=2	M=5	M=10
Anneal.orig	.9903	.9956	.9958	.9958	.9959	.9958
Audiology	.9444	.9465	.9335	.9384	.9372	.9329
Autos	.9595	.9498	.9509	.9532	.9505	.9483
Balance-Scale	.9386	.9652	.9621	.9630	.9661	.9684
Breast-cancer	.6349	.6700	.6708	.6683	.6711	.6722
Breast-w	.9920	.9913	.9920	.9920	.9919	.9919
Colic	.8802	.8985	.8938	.8934	.8933	.8922
Credit-a	.9298	.9373	.9365	.9364	.9367	.9365
Credit-g	.7559	.7883	.7912	.7913	.7915	.7912
Diabetes	.8098	.8257	.8203	.8201	.8216	.8226
Glass	.9022	.9203	.9141	.9198	.9172	.9134
Heart-c	.8951	.9115	.9108	.9105	.9113	.9120
Heart-h	.8630	.8911	.8817	.8792	.8809	.8831
Heart-statlog	.8821	.9029	.9003	.9004	.9016	.9014
Hepatitis	.8161	.8572	.8514	.8537	.8521	.8501
Hypothyroid	.9953	.9986	.9983	.9979	.9979	.9979
Ionosphere	.9724	.9639	.9732	.9732	.9722	.9711
Iris	.9937	.9929	.9929	.9929	.9932	.9933
Kr-vs-kp	.9984	.9997	.9996	.9996	.9998	.9998
Labor	.9115	.9257	.9365	.9365	.9324	.9257
Lymph	.9076	.9375	.9378	.9391	.9388	.9364
Primary-tumor	.7547	.8138	.7553	.7943	.7909	.7857
Segment	.9996	.9993	.9994	.9994	.9994	.9994
Sick	.9947	.9973	.9977	.9977	.9976	.9974
Sonar	.9283	.9200	.9303	.9304	.9288	.9261
Soybean	.9953	.9959	.9950	.9953	.9952	.9949
Vehicle	.9368	.9354	.9367	.9368	.9366	.9363
Vote	.9764	.9938	.9933	.9933	.9930	.9928
Vowel	.9969	.9945	.9946	.9955	.9951	.9947
Zoo	.9452	.9944	.9708	.9944	.9937	.9905

Table D.14.: Average weighted AUC of the bagged JRip and the employed probability estimation techniques: best rule, 100 samples, unpruned rule sets (except for precision).

Name	Bagged Jrip	Precison	Laplace	M=2	M=5	M=10
Anneal.orig	.9903	.9953	.9936	.9945	.9944	.9945
Audiology	.9444	.9561	.9434	.9458	.9437	.9405
Autos	.9595	.9507	.9501	.9528	.9500	.9478
Balance-Scale	.9386	.9595	.9616	.9626	.9659	.9685
Breast-cancer	.6349	.6588	.6736	.6702	.6725	.6740
Breast-w	.9920	.9913	.9919	.9919	.9918	.9918
Colic	.8802	.9014	.8894	.8901	.8889	.8881
Credit-a	.9298	.9358	.9344	.9344	.9348	.9349
Credit-g	.7559	.7865	.7890	.7894	.7901	.7899
Diabetes	.8098	.8216	.8216	.8206	.8208	.8211
Glass	.9022	.9213	.9059	.9133	.9103	.9048
Heart-c	.8951	.9091	.9127	.9129	.9136	.9134
Heart-h	.8630	.8912	.8806	.8787	.8802	.8804
Heart-statlog	.8821	.9008	.9028	.9021	.9023	.9016
Hepatitis	.8161	.8458	.8488	.8486	.8516	.8544
Hypothyroid	.9953	.9940	.9958	.9962	.9963	.9962
Ionosphere	.9724	.9683	.9782	.9783	.9776	.9769
Iris	.9937	.9936	.9935	.9935	.9936	.9937
Kr-vs-kp	.9984	.9997	.9996	.9996	.9996	.9996
Labor	.9115	.9203	.9311	.9311	.9257	.9270
Lymph	.9076	.9416	.9382	.9381	.9373	.9359
Primary-tumor	.7547	.8164	.7678	.7984	.7941	.7892
Segment	.9996	.9993	.9992	.9993	.9992	.9992
Sick	.9947	.9978	.9961	.9966	.9965	.9963
Sonar	.9283	.9268	.9299	.9301	.9278	.9258
Soybean	.9953	.9957	.9941	.9949	.9948	.9947
Vehicle	.9368	.9343	.9361	.9363	.9360	.9356
Vote	.9764	.9878	.9922	.9923	.9920	.9918
Vowel	.9969	.9950	.9945	.9955	.9951	.9946
Zoo	.9452	.9948	.9750	.9952	.9950	.9946

Table D.15.: Average weighted AUC of the bagged JRip and the employed probability estimation techniques: macro averaging, 100 samples, unpruned rule sets (except for precision).

Name	Bagged Jrip	Precison	Laplace	M=2	M=5	M=10
Anneal.orig	.9903	.9975	.9964	.9972	.9973	.9974
Audiology	.9444	.9555	.9447	.9464	.9445	.9421
Autos	.9595	.9518	.9538	.9561	.9538	.9513
Balance-Scale	.9386	.9626	.9603	.9617	.9646	.9669
Breast-cancer	.6349	.6705	.6738	.6730	.6747	.6760
Breast-w	.9920	.9909	.9917	.9916	.9917	.9914
Colic	.8802	.9023	.8962	.8964	.8958	.8945
Credit-a	.9298	.9368	.9353	.9351	.9355	.9354
Credit-g	.7559	.7862	.7887	.7888	.7890	.7886
Diabetes	.8098	.8220	.8191	.8190	.8201	.8215
Glass	.9022	.9232	.9171	.9216	.9198	.9166
Heart-c	.8951	.9110	.9114	.9110	.9122	.9123
Heart-h	.8630	.8898	.8813	.8793	.8812	.8825
Heart-statlog	.8821	.9021	.8993	.8993	.8996	.9002
Hepatitis	.8161	.8554	.8557	.8542	.8552	.8516
Hypothyroid	.9953	.9966	.9960	.9960	.9960	.9959
Ionosphere	.9724	.9673	.9760	.9760	.9750	.9737
Iris	.9937	.9936	.9933	.9932	.9935	.9935
Kr-vs-kp	.9984	.9995	.9995	.9995	.9995	.9995
Labor	.9115	.9176	.9243	.9230	.9189	.9162
Lymph	.9076	.9400	.9398	.9412	.9402	.9377
Primary-tumor	.7547	.8208	.7678	.8002	.7984	.7944
Segment	.9996	.9993	.9994	.9994	.9994	.9994
Sick	.9947	.9975	.9977	.9979	.9978	.9976
Sonar	.9283	.9213	.9316	.9316	.9290	.9266
Soybean	.9953	.9959	.9948	.9954	.9951	.9949
Vehicle	.9368	.9353	.9368	.9369	.9366	.9363
Vote	.9764	.9881	.9924	.9924	.9918	.9911
Vowel	.9969	.9950	.9950	.9959	.9955	.9951
Zoo	.9452	.9948	.9751	.9953	.9952	.9944

Table D.16.: Average weighted AUC of the bagged JRip and the employed probability estimation techniques: micro averaging, 100 samples, unpruned rule sets (except for precision).

Name	Bagged Jrip	Precison	Laplace	M=2	M=5	M=10
Anneal.orig	.9903	.9970	.9959	.9967	.9969	.9970
Audiology	.9444	.9565	.9458	.9450	.9429	.9404
Autos	.9595	.9502	.9511	.9537	.9513	.9487
Balance-Scale	.9386	.9627	.9601	.9618	.9649	.9674
Breast-cancer	.6349	.6717	.6757	.6737	.6760	.6773
Breast-w	.9920	.9906	.9914	.9914	.9913	.9911
Colic	.8802	.9050	.8963	.8962	.8961	.8945
Credit-a	.9298	.9385	.9355	.9353	.9358	.9359
Credit-g	.7559	.7871	.7901	.7901	.7903	.7899
Diabetes	.8098	.8232	.8203	.8202	.8215	.8226
Glass	.9022	.9214	.9171	.9215	.9194	.9163
Heart-c	.8951	.9133	.9114	.9106	.9118	.9122
Heart-h	.8630	.8893	.8837	.8808	.8833	.8850
Heart-statlog	.8821	.9031	.9000	.8997	.8998	.8997
Hepatitis	.8161	.8565	.8554	.8554	.8562	.8526
Hypothyroid	.9953	.9967	.9962	.9961	.9961	.9961
Ionosphere	.9724	.9681	.9770	.9770	.9762	.9752
Iris	.9937	.9928	.9923	.9925	.9925	.9927
Kr-vs-kp	.9984	.9995	.9995	.9995	.9995	.9995
Labor	.9115	.9230	.9284	.9297	.9257	.9243
Lymph	.9076	.9393	.9408	.9422	.9409	.9388
Primary-tumor	.7547	.8194	.7713	.8018	.8001	.7973
Segment	.9996	.9993	.9994	.9994	.9994	.9994
Sick	.9947	.9973	.9976	.9978	.9977	.9975
Sonar	.9283	.9184	.9296	.9297	.9278	.9264
Soybean	.9953	.9956	.9948	.9951	.9949	.9946
Vehicle	.9368	.9356	.9372	.9373	.9370	.9366
Vote	.9764	.9885	.9927	.9927	.9921	.9910
Vowel	.9969	.9943	.9943	.9953	.9949	.9944
Zoo	.9452	.9947	.9750	.9951	.9948	.9943