Supervised Local Pattern Discovery

Diplomarbeit

Im Studiengang **Wirtschaftsinformatik**

angefertigt am Fachgebiet Knowledge Engineering

der Technischen Universität Darmstadt

von

Sven Wagner

Februar 2008

Betreuer: Prof. J. Fürnkranz, Jan-Nikolas Sulzmann

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Diplomarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, 5.2.2008

Table of Contents:

I *List of figures*

II *List of Tables*

III *List of abbreviations*

KDD = Knowledge Discovery in Databases

DNF = Disjunctive Normal Form

TP = True positives

FP = False positives

FP-Tree = Frequent Pattern Tree

TN =True negatives

FN =False negatives

SPD =Supervised Pattern Discovery

SLPD =Supervised Local Pattern Discovery

1 Introduction

1.1 Motivation

Supervised local pattern discovery is a midfield task between associations rule mining and inductive learning. It aims at finding patterns in labeled data that are descriptive. Lavrač et al. (2005) describe a pattern as being local while the global counterpart to a pattern is a model, which explains data formation.

Practical use of algorithms in this area is motivated through their successful application in a variety of contexts. For instance, supervised local pattern applications have been used in marketing. Lavrač et al. (2004a) used it to distinguish specific characteristics of customer groups with a focus on discovering actionable or operational rules. Other applications use it for identification of risk factors for coronary heart disease (Gamberger and Lavrač, 2002a,b) as well as university marketing which led to changes in student recruiting practices (Bay and Pazzani, 2001). Rather than trying to build a model which could be used to predict the behavior of a certain example, local patterns have been used to describe examples and build knowledge with the experts that work in student recruiting or marketing or other branches, helping them on a day to day basis. The task works on labeled data, which makes it applicable for research questions focused on few attributes of interest In contrast, association rule discovery works on unlabeled data and is concerned with questions like what products are typically bought together. Supervised local pattern discovery works with labeled data and can therefore help answering questions like what features do people have that have repeatedly bought a certain product or possibly never bought a certain product. In rule learning the goal is to find good rules which help to discriminate between examples of different groups. The goal is to be able to infer the class of unknown examples. In supervised local pattern discovery, patterns or rules are of interest that might not be the best discriminators, but they might still contain valuable information for a working professional. In recent years, several new algorithms have

appeared that derived new approaches to supervised local pattern discovery from rule mining. This thesis surveys those algorithms and presents a way of integrating local pattern discovery algorithms in a generic rule learning environment.

1.2 Objectives of the research

The aim of this study is to give an overview on recent works in the area of supervised local pattern discovery. Recent developments and similarities in the different algorithms for detecting local patterns in data are presented. As this thesis is mainly intended as a literature study there is no distinct research question or research hypothesis which will be followed. Instead the general aim of this thesis is to provide an overview on recent developments in the area of supervised local pattern discovery and to provide an abstract understanding of the general mechanisms which are applied by all algorithms. Therefore this thesis presents a generic local pattern discovery system which can be used to implement a majority of the algorithms presented here.

1.3 Problem formulation

To be able to understand the topic one first needs to define what is meant by patterns. Hand (2002) defines a pattern as "*a data vector serving to describe an anomalously high local density of data points*". This can be interpreted as a rule which is used to describe a set of similar examples. Informally, a pattern is defined as a local anomaly which is of special interest. Therefore it is important to note the difference of this definition compared to other usages of the word pattern, especially pattern here does not indicate repetitive regularities but areas of exceptionally dense data in a wide and normally sparsely occupied data space. An example for a sparsely occupied data space is the combination of all products in a certain store carrying thousands of different products. Only a few items will regularly be bought together. Therefore it is obvious that patterns as used here are always local since they do not describe a repetitive regularity which would have global meaning for the data of interest. Despite this definition,

since the research area is relatively new, there is so far no generally accepted definition. Especially, locality presents a problem which is described in section 2. Pattern discovery should also be contrasted to what is described as pattern matching. Hand describes the key difference as follows: pattern matching means that the patterns, the information of interest, come from outside the data. Its aim is not to look for unknown anomalies, but rather the anomaly is known beforehand and is being scanned for in the data. This thesis describes algorithms that are developed to be used for supervised pattern discovery. This means that patterns describe different data points according to an attribute of special interest. That way one can consider the data as being labeled. For this thesis most interesting are algorithms that have been discussed as problems called subgroup discovery, contrast set discovery or exception rule discovery.

1.4 Overview

This thesis is structured as follows. Chapter 2 gives a more detailed introduction into the topic. Chapter 3 defines the different problem statements that have been developed in the realm of supervised local pattern discovery. Chapters 4 to 6 describe the actual algorithms in terms of data preprocessing, search algorithms and post-processing. In chapter 7 some related algorithms are briefly discussed before chapter 8 concludes this thesis.

2 Data Mining and Pattern Discovery

After having discussed the basic properties of supervised local pattern discovery (SLPD) in the introduction, this chapter describes the tasks in more detail. It shows how SLPD is a midfield task between the two related tasks of inductive rule learning and association rule mining. Section 2.1 introduces the tasks of knowledge discovery in databases in general before section 2.2 discusses the rule learning task more deeply as the basis for supervised pattern discovery. Section 2.3 reviews the problems of finding a general definition for local pattern discovery and addresses the issue of supervised learning in the context of pattern discovery. Section 2.4 covers the task from an algorithmic point of view which is used as a guideline for the latter part of this thesis.

2.1 KDD

Fayyad et al. (1996) define Knowledge Discovery in Databases (KDD) as „*the non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data.*" Furthermore they depict a pattern as a description of a subset of the data that can be stated in a simpler form than enumerating all facts of that subset. Similarly Hand (2002) describes a pattern as "*a data vector serving to describe an anomalously high local density of data points*".

In KDD one can generally distinguish between two main goals. First there is the predictive task. Predictive algorithms deal with the prediction of unknown variables in a database based on some other known variables (Fayyad et al. 1996). Often this task is described as model building (Lavrač, Železný & Džeroski, 2005; Hand, 2002). An example for this task is rule mining algorithms (e.g. see Fürnkranz, 1999). The second goal is descriptive in nature. It aims at finding understandable patterns in the data. As Hand (2002) points out, pattern detection, as a research goal by itself, is a relatively new research area. An example of descriptive pattern detection and one of the most well known

applications is the discovery of association rules (Agrawal, Imielinski & Swarni, 1993). Association rules are often used in terms of market basket analysis, which tries to discover itemsets in a market basket which are frequently bought together.

Fayyad et al. (1996) identified classification, regression, clustering, summarization, dependency modeling, as well as change and deviation detection as primary data mining tasks. Classification and regression are both prediction tasks which aim at mapping an example according to a rule based model or a learned function to one of several predefined classes or to a real valued prediction variable. To be able to do that, both task learn a function based on examples that have been labeled according to the predefined classes in the case of classification learning or have a real value as a label in the case of regression. New examples are labeled based on the model that has been learned using the previously seen examples. Essentially in classification there is a finite number of classes, while with regression an example is assigned a real number.

Clustering, summarization, dependency modeling and change and deviation detection are descriptive tasks. Clustering aims at grouping examples in subsets which can be exclusive but not necessarily have to be depending on the context. While in the classification context, examples had been labeled, in clustering, labeling prior to cluster discovery is not assumed. Summarization highly compresses information on a subset of the data. It is descriptive and the summary might contain valuable information for a human reader though it might be too general to be used for classification purposes. An example for a summarization task is to calculate mean and standard deviation for the fields of an example table. Dependency modeling aims at discovering dependencies between variables, which are described in terms of existence and magnitude. Deviation detection is used for discovering changes which have occurred over time.

Since for supervised local pattern discovery rule learning algorithms are most relevant, the next section discusses rule learning in short before the following section discusses issues of SLPD.

2.2 Rule Learning

Rule learning means that the results of the learning algorithm lead to "if … then …" type of rules. Considering the classification above, rule learning can be used in predictive as well as descriptive settings. It is applied in classification tasks, where the rule consequent consists of one of the predefined classes, while the rule antecedent is a combination of values of attributes of the examples. Similarly rules are applied for descriptive tasks such as association rule discovery. Before going a bit more into both tasks, one should define more formally what should be understood when discussing rules. The definitions follow the work of Zimmermann and De Raedt (2005).

Definition 1 (Literal). A literal is an attribute-value-pair $A = v$ with $v \in V[A]$. An instance $\langle v_1,...,v_d \rangle$ is covered by a literal l of the form $A_i = v$ iff $v_i = v$.

Definition 2 (Rule). A rule r is of the form $b \Rightarrow h$ with $b = l_1 \wedge ... \wedge l_i$ the rule body and $h = l'_1 \vee ... \vee l'_d$ the rule head. An instance e is covered by b iff e is covered by all b's literals and e is correctly covered by the entire rule r iff it is covered by at least one literal in h as well.

In the definition $\mathrm{A} = \{A_1,..., A_d\}$ is a set of ordered attributes and $V[A] = \{V_1,...,V_p\}$ is the domain of A. An *instance* e is a tuple $\langle v_1,...,v_d \rangle$ with $v_i \in V[A_i]$. Additionally, for every instance e there is a weight which is set to 1 by default. Removing an example is equivalent to setting the example weight to 0. A multiset $E=\{e_1,...,e_n\}$ is called a *data set*.

Definition 2 allows for multiple outcomes of a rule. In the case of a binary classification the rule head typically states that covered examples are positive.

In those algorithms presented here, the rule head consists of only one literal rather than a disjunction of literals.

Since a rule should be applicable to more than one example, one is often interested in how many examples satisfy the rule body and the rule head. This is called support of a rule.

Definition 3 (Support) For a literal l the support of l is defined as $\sup(l) = |\{e \mid e \text{ is covered by l}\}|$ with $e \in E$.
Also for a rule body b the support of b is defined as $\sup(b) = |\{e \mid e \text{ is covered by b}\}|$ with $e \in E$. Often this is just described as coverage.
Finally, the support of a rule with only a single consequent $\sup(b \Rightarrow h) = |\{e \mid e \text{ is covered by b} \wedge e \text{ is covered by h}\}|$ with $e \in E$.

Often it is said that a rule r covers x examples, which means that the support of rule r is x. As mentioned above, rule learning can be classified in two conceptual problems, inductive rule learning and association rule discovery. Algorithms that had originally been developed for either of those tasks are the basis for the algorithms for SLPD therefore it is helpful to discuss both issues a little further.

2.2.1 Inductive Rule Learning

As said before the aim of the classification task is to define a function which is used to map examples to a predefined set of classes. In inductive rule learning the function is based on a set of rules. Each of the rules cover a part of the example space and the whole set of rules describes the global model which ideally describes the way the data has been generated. The rules are discovered through a sequential covering algorithm where for each discovered rule, all examples covered by that rule are removed and only then the next rule is discovered ignoring all previously covered examples. Discovery is performed by searching the hypothesis space, often done by starting from the most general rule, which covers all examples, to more specialized rules which cover less examples, though the number of examples that belong to a specific class

should increase while the number of examples covered that belong to another class should decrease. Alternatively, search can start with a single example as a first, very specific, rule, which is generalized during the search. The search stops after a stopping criterion is met or there are no more examples left, since they have all been covered by at least one rule. The discovered set of rules describes the data. There is a wealth of different algorithms that are based on this covering approach. Most of them differ in the way a new rule is generated or the way the hypothesis spaces is being searched.

This type of learning is called supervised learning since all the training data has already been labeled. Those labels make up the possible values for classification. For an unclassified example it is now possible to predict a classification by finding a rule that can be applied to cover this example. Since the final goal of inductive rule learning is the classification of unseen examples, it is necessary to make sure that discovered rules are general enough so they are likely to be applicable to new, unseen examples. This is often done by preferring shorter rules. In cases where more than one rule applies to the previously unseen example, voting schemes can be used or the order in which rules are tested can be seen the decisive factor.

Rule sets are not the only possibility to generate models. Decision trees (e.g. Mitchell 1997) can also be applied, but since rule set algorithms have been more commonly used for developing SLPD algorithms there is no need to discuss this issue further. More often, association rule learning algorithms have been found useful in a SLDP context. Therefore the problem of association rule mining is discussed next.

2.2.2 Association Rule Learning

Other than inductive rule learning, association rule learning is a descriptive task. It has originally been motivated through market basket analysis. In a shopping setting, it might be interesting to analyze buyer behavior. Therefore one can ask

what items are often bought together and possibly place different often bought items in a way that encourages buying other items as well.

Association rules are defined on binary attributes $I \in \mathrm{I}$ which are called items. On those attributes, transactions in form of a binary vector e with $e[k] = 1$ if item k was part of the transaction e are stored in a transactional database T. Agrawal (1993) defines an association rule as follows:

Definition 4 (Association rules) An association rule is an implication of the form $X \Rightarrow I_b$ with X being a set of items and $I_b \notin X$. The rule has to be above a certain confidence threshold $0 \leq c \leq 1$ which entails that at least $c\%$ of all transactions in T that satisfy X also satisfy I_b. This means that among those transactions that comprise all items in X, at least $c\%$ also contain item I_b.

Compared to inductive rules as described above one should notice that the rule head can be any item $I \in \mathrm{I}$ as long as it is not part of the antecedent. This is opposed to the idea of supervised learning in which the rule head is limited to a number of classes. In this sense, association rule learning is a type of unsupervised learning. Nevertheless there have been successful attempts to apply association rule discovery algorithms for inductive rule learning by limiting the rule heads therefore the differences between association rule discovery and inductive rule learning are rather conceptual and in terms of intended applications. Therefore algorithms for both problem types have been used as basis for the development of SLPD algorithms.

2.2.3 Hypothesis Language

After having defined what is meant by a rule, how a rule covers an example and described two related and important rule learning tasks, it is necessary to discuss how rules are applied in the algorithms and what that means for the search. The hypothesis language constraints the search space in which an algorithm can look for a problem solution by defining what a hypothesis or rule looks like. This section gives an overview on hypothesis language with regard to the algorithms that are described in this paper.

Most algorithms that have dealt with the issue of supervised local pattern discovery have done so in a propositional approach. Mostly this is done in a conjunctive approach which does not allow for internal disjunctions. As a result, rules that are built can be described as conjunctions of selectors.

$$b = l_1 \wedge ... \wedge l_m$$

with $m \leq d$ and d being the total number of attributes, there is at most one literal per attribute, since otherwise a rule would have zero coverage.

Some authors have suggested the use of internal disjunctions which leads to rules of the type of

$$b = l_1 \wedge (l_{2_1} \vee l_{2_2} \vee l_{2_3}) \wedge ... \wedge l_m$$

With $(l_{2_1} \vee l_{2_2} \vee l_{2_3})$ being different attribute value combinations for the second attribute. This type of hypothesis language does however lead to an exponential increase in possible rules. Atzmüller and Puppe (2006) explicitly describe a way of dealing with the problem in their algorithm, though their suggestions are merely a technical choice that does not necessarily help in reducing the search space. First they argue if internal disjunctions are only needed for few attributes they can be implemented straightforward. Though, if all possible combinations are needed, the authors suggest the use of conjunctions of negated selectors rather than disjunctions. Internal disjunctions are also part of the MESDIF (Multiobjective Evolutionary Subgroup DIscovery Fuzzy rules) algorithm (Berlanga et al. 2006). Since the algorithm performs a genetic search and is therefore a heuristic approach the algorithm should be able to handle the increased search space since only approximations of the best rules are discovered anyway.

In terms of hypothesis language the MESDIF algorithm is exceptional in another aspect as well. Different from the other algorithms, MESDIF, applies fuzzy logic to deal with continuous variables. This means that rather than discretizing

continuous variables and replacing the original value with the corresponding discretized version, fuzzy logic allows replacing the value with different corresponding discretized values but only to a certain degree. This degree of membership to one or more of the new values is dependent on a membership function. Berlanga et al. (2006) suggest the use of uniform partitions with triangular membership functions in cases where there is no available expert knowledge.

First order logic approaches are rarely used. First order logic has been applied by Lavrač et al (2002) in the RSD algorithm. Here, it is used only to generate features to form a single table from a multi-relational database on which then essentially the CN2-SD algorithm is run which makes use only of propositional logic. Wrobel (1997) also uses first-order logic in order to be able to deal with multiple relations rather than just a single table. The rules itself are stated in first order logic.

Since few of the algorithms depart from the propositional approach, the expressiveness of the discussed algorithms and search costs can be considered similar for most algorithms. This is not surprising since one of the goals of local pattern discovery is that discovered patterns should be human understandable. Using a different approach might lead to similar or even better patterns, though depending on the language in which they are stated, they might be difficult to understand. Therefore one should remember that only those algorithms that try to discover patterns in multi-table databases are employing a first order approach. All other algorithms use conjunctive rules as described above.

2.3 Local pattern discovery

Previous sections discussed the tasks of KDD generally and inductive rule learning and association rule learning problems especially. Also the distinction between predictive and descriptive tasks has been introduced. As it is possible to turn a descriptive association rule discovery algorithm into an predictive rule

learner has one can see that the lines are often dependent on application and interpretation rather than in methodology. Therefore, supervised local pattern discovery has characteristics of inductive as well as descriptive learning. Lavrač et al. (2005) describe the task as between purely descriptive and predictive induction. This view is justified by recognizing that supervised pattern discovery can be seen as a way of supervised learning, while it is also a form of descriptive induction since the goal is to discover interesting patterns as knowledge rather than building a model explaining the data generating process. Therefore, a key issue to be aware of is that when trying to discover new knowledge through patterns, it is important that the results are probable to fit new data as well. Consequently, it is important that the discovered patterns generalize well beyond the data they were discovered on. Only if this property can be assumed to hold, one can consider the discovered knowledge as applicable to other data and consider it useful. This is also a needed property of supervised learning which underlines the relations between both tasks.

It should now be made clear what is to be understood when discussing supervised local pattern discovery. Therefore let's first discuss what is meant by local patterns. Hand (2002) described the local pattern discovery problem analogously to statistical modeling as

$$data = background\_model + local\_patterns + random\_component$$

The definition given by Hand is one of the first definitions that arose on local patterns. There is still a discussion going on, on how to correctly define local patterns. Bonchi and Giannotti (2005) summarize a discussion on how to define local patterns by three main criteria:

1. *local patterns cover small parts of the data space*
2. *local patterns deviate from the distribution of the population of which they are part of*
3. *local patterns show some internal structur*e.

Most notable in the discussion on local patterns is the focus on small parts of the data space. Hand, for instance, describes local patterns as being “small” phenomena. Yet, it is not clear what exactly makes a small pattern. Bonchi and Gianotti discuss this issue using the question of whether association rules could be considered as local patterns. The problem is that all association rules taken together form a model of the data. Thus, in their opinion, the set of association rules cannot be considered a local pattern. A single rule though could be considered a local pattern unless it has very high support, representing obvious or already known knowledge. The same is true for the aforementioned idea of turning the set of learned patterns into a classifier. The key idea for describing local patterns as small phenomena of sorts is that large patterns can be discovered by humans and most likely have already been discovered, therefore there is no need to rediscover this knowledge using machine learning tools. Nevertheless, the question on how big is too big for being a local pattern is difficult to answer. Hand (2002) suggests choosing a distance measure dependent on the data and application domain. Also the necessary threshold should be chosen domain specific. An example of such a measure can be frequency as has been used in searching for frequent patterns. A frequent itemset or frequent pattern has been defined as follows (Han et. al, 2004)

Definition 5 (Frequent pattern) A pattern A is frequent if A’s support is no less than a predefined minimum support threshold ξ. The support of a pattern A is the number of transactions in T that contain the pattern.

While a very frequent pattern can be considered a global pattern, a less frequent pattern would be considered local (Bonchi and Giannotti, 2005).

While most of the points discussed so far were originally aimed at the task of unsupervised local pattern discovery, the discussion should also hold for the case of supervised local pattern discovery. Nevertheless the interpretation of smallness plays a less significant role in problem definitions that could be considered part of the SLPD problem family. As Lavrač et al. (2005) define the task of subgroup discovery as finding “*population subgroups that are statistically*

'most interesting', e.g. are as large as possible" it shows that their definition does not solely focus on small phenomena. Similarly, Fürnkranz (2005) interprets a single rule from a rule set as a local pattern. Therefore the issue of locality has rarely been tackled from the small phenomena point of view in the algorithms that have been introduced so far.

The other important property of supervised local pattern discovery is the focus on supervised data. Supervised local pattern discovery is the process of discovering properties of a population of examples that have a certain property of interest. This property of interest can be seen as a label distinguishing two or more groups in the population, making it possible to classify examples. For instance in a setting which tries to discover patterns in car accident data, one might be interested in seeing commonalities of lethal accidents, accidents with heavy injuries, light injuries and accidents without any injuries. Supervised pattern discovery only looks for patterns with regard to those categories. The alternative would be to search for patterns in non-labeled data which means that patterns can be of any kind while in supervised pattern detection pattern rules are learned with only the attributes of interest as possible rule outcomes. This is similar to rule learning in the classification task and therefore patterns could be used to predict a classification for unknown examples. This has been undertaken by Lavrač et al. (2004a) which showed a reasonable performance at this task.

Local pattern discovery can be considered somewhat similar to the task of clustering (Höppner, 2005). There are important differences though. First, the focus on labeled data distinguishes supervised local pattern discovery most from the related task of clustering. Clustering typically is applied for unlabelled data, therefore automatically creating class labels for different clusters. Considering the problem of supervised local pattern discovery, there is a second issue that needs to be considered. Clustering should result in different, exclusive clusters where every example does belong to exactly one cluster. Höppner argues that if this constraint is relieved then clustering and pattern

discovery are practically the same. This point was made for the task of subgroup discovery which is a specific task within the general concept of local pattern discovery.

Before discussing how local pattern discovery can be performed from an algorithmic point of view and what kind of challenges need to be addressed, one possible element of confusion should be addressed. As pointed out by Fayyad et al (1996), it is important to distinguish pattern discovery from pattern recognition. While pattern discovery is a descriptive task of identifying a pattern and presenting it in an understandable way, pattern recognition is a predictive task in which an application has to label a new example according to previously seen examples. Additionally, pattern recognition needs to identify the example despite differences in appearance, e.g. in speech recognition. Hand (2002) elaborates by stating that in pattern discovery the information comes mostly from inside the data itself, while in pattern recognition outside patterns such as words have to be recognized in a wealth of examples. This is again meant for unsupervised pattern discovery. Though in supervised pattern discovery the only type of information provided from outside the database is the feature of interest. Therefore in pattern detection, one needs not only to find patterns in the database, but should also establish that the found pattern is indeed a pattern that has occurred due to the underling model responsible for data generation and has not just occurred by chance. He points out that for this, the algorithm can rely only on the information in the database and no other information.

To summarize this section we can state that the supervised local pattern discovery task needs to satisfy the following criteria: patterns discovered need to be local in that sense that they do not describe a complete model of the evaluated data, though there is no generally accepted definition of locality yet. Second, supervised local pattern discovery is a task that is performed on labeled data and therefore is similar to classification rule learning systems. Therefore the discussion on problems in discovery of supervised local patterns

often refers to problems that have also been discussed in the context of supervised learning.

2.4 Supervised Local Pattern discovery techniques

This section now deals generally with the algorithmic solution to the problem of supervised local pattern discovery and discusses the problems that have to be addressed. In doing so it provides the structure of the later part of this paper.

A variety of approaches have been developed, though mostly the issue of locality is somewhat ignored. The most important approaches are described in detail in chapter 3. This section aims at describing the most important aspects of supervised local pattern discovery techniques. This provides the background for the structure of the later chapters of this thesis. Due to the wealth of different vocabulary, the terms rule and pattern are used interchangeably in this thesis. This is possible since supervised patterns discovery consists of a pattern description and a corresponding classification which can be considered a rule. In Kralj et al. (2007) one can find a description of synonyms that have arisen in the contexts of subgroup discovery, rule learning and contrast set mining.

As has been discussed above, single rules derived from a rule learning algorithm or association rules can be considered local patterns. Hence most algorithms draw from the literature on rule learning or association rule mining. Additionally using rule based systems for pattern discovery has the advantage of being understandable for humans. When considering association rule mining based approaches though, criticism arises on the wealth of rules that are found (Bay & Pazzani, 2001; Höppner, 2005). Mined rules are often so vast that they are hardly manageable. Nevertheless, the principles of mining for association rules are commonly used in supervised pattern mining. Opposed to that, other algorithms are based on rule learning algorithms like CN2 (Clark and Niblett, 1989, Clark and Boswell, 1991). These have the advantage that using only the best rules significantly reduces the total number of rules presented to the user, though at the cost of potentially missing important yet slightly less well

performing rules. Furthermore, the use of a heuristic search might even lead to missing the actual best patterns. So, in supervised local pattern discovery the goal is to find human understandable, interesting rules without loosing too much information. Some solutions that have been suggested deal with avoiding finding highly correlated rules. For instance, if one rule is stating that young drivers are more likely to be involved in an accident, a second one stating that drivers that only recently acquired their driver's license are more likely to be involved in an accident is not interesting once the first one is found since both rules cover nearly the same examples (Scholz, 2005). Generally a simple solution to the problem of finding too many rules is to offer the user the ability to limit the number of rules that are found. In such a scenario it is important to make sure that those first best rules are as diverse as possible. Another possibility is to post-process discovered rules in such a way that only those rules that are deemed interesting are shown to the user or only those rules that are quite different from those already presented are presented next. As can be seen, discovery and presentation of rules are interconnected and important in local pattern discovery. Therefore this discussion leads to two important aspects of an algorithm for this task. First one needs to determine which rules are better or more interesting through the use of a quality measuring function and secondly, a possible post-processing step is often required in which rules that have been found are further analyzed in order to avoid excessive amounts of rules.

Another problem posed by the task of supervised local pattern discovery is the problem of the vastness of the search space. In order to reduce this problem, it is possible to try either a heuristic approach to rule discovery which may lead to good, though possibly suboptimal solutions or to try exhaustive search which employs constraints to limit the search space. This leads to search algorithms that find all rules considering constraints such as minimum frequency. This constraint could be chosen rather strict to make search feasible. This latter approach is used mostly in frequent itemset approaches trying to find

association rules. The search space problem is also influenced by the hypothesis language which should hence be a topic for discussion as well.

Last but not least there is the data quality problem. A quality problem could be either missing values or simply wrong data. While solutions have been suggested in the literature to deal with the first problem through a pre-processing step, wrong data cannot be addressed automatically. In this context Hand (2002) reminds us to be careful when using results from data mining as he states Twyman's Law that "*any figure that looks interesting or different is usually wrong*". Furthermore there is the question of representability. As most rule based algorithms are based on a symbolic representation of attribute-value pairs, continuous variables pose some problems. These can also be addressed during a pre-processing phase.

Considering the discussion above there have been four key steps identified that have been addressed in the presented literature. First one needs to perform pre-processing of the data in order to improve data quality and adapt the data to language constraints, second there is the search has to be organised efficiently, whether by employing heuristic searches or exhaustive search and how those are implemented. Third, there are quality functions used in order to find the best rules and fourth post-processing issues that try to avoid presenting the user too many rules.

Figure 1 shows an overview on the supervised local pattern discovery task based on the tasks discussed above. The processes *PreprocessData*,

```
    Procedure LOCALPATTERNDISCOVERY(Examples)
1   LocalRules={ }
2   FinalRules={ }
3   PREPROCESSDATA(Examples)
4   LocalRules=FINDLOCALPATTERNS(Examples)
5   FinalRules=POSTPROCESSPATTERNS(Examples,LocalRules)
6   return(FinalRules)
```

Figure 1.Overview on the processes of local pattern discovery algorithms

FindLocalPatterns, and *PostProcessPatterns* are described in their respective sections. Note that the key process here is the *FindLocalPatterns* and that most of the preprocessing task consists of issues that are not unique to the supervised local pattern discovery task and that cannot easily be generalized. They are important to mention though, since often these tasks deal with aspects that do necessarily pose a problem for the original application of an underlying algorithm. For instance, the missing value problem does not occur in an association rule discovery scenario for market based data, but it does pose a problem for pattern discovery on other data.

Before discussing the different processes, the following chapter presents the most important problem definitions that have been developed in the area of supervised local pattern discovery. After that, chapters' four to six discuss the different algorithms in detail according to the order formulated above. Chapter seven will then show some other algorithms that deviate significantly from those discussed in chapters four to six.

3 Problem Definitions in Supervised Local Pattern Discovery

SLPD is being discussed from a variety of viewpoints. All approaches try to discover interesting subsets in a dataset or even within a subset of the dataset. This chapter presents different problem definitions that have arisen in the literature which can be attributed to the field of supervised local pattern discovery. Most of these fields have been discussed independently of the other literature, though some attempts of incorporating different research streams have been performed (i.e. Zimmerman and de Raedt, 2005, Kralj et al., 2007). The key research streams have been named subgroup discovery, contrast sets, correlated pattern mining, exception rules and cluster grouping, which itself is a superset of subgroup discovery and correlated pattern mining. Those are described in more detail in the following sections. The goal of this section is to provide an overview on the research streams that have been developed by the different research communities. Furthermore, it is a goal of this section to underline the research questions that where asked which lead to the development of this type of problem. These research streams are important since different algorithms have been developed within the different communities. As we will see, most research goals are achievable by applying adapted algorithms that had originally been developed with a different research question in mind.

3.1 Cluster Grouping

Cluster grouping has been developed by Zimmerman and de Raedt, (2005) to unify several data mining concepts that are among the group of supervised local pattern discovery. There is no explicit research question that was at the beginning of the development of this problem type. Rather than that, the question was to unify different seemingly different problem definitions. The authors also have devised an algorithm which can solve the problem of cluster

grouping, the CG-algorithm. For the current analysis, the definition of the cluster grouping problem is most interesting. Zimmerman and de Raedt, define the problem of cluster grouping as

Given:

- A set of literals L
- A data set E
- An interesting measure σ
- A set of target Literals T

Find:

The set of k rules expressible in L having the highest value of σ on E w.r.t. the given target literals T .

Using this definition the authors define subgroup discovery as well as correlated pattern mining which is shown next.

3.1.1 Subgroup Discovery

The task of subgroup discovery has been described by Lavrac et al. (2004a) as follows:

> *"Given a population of individuals and a property of those individuals we are interested in, find population subgroups that are statistically 'most interesting', e.g., are as large as possible and have the most unusual statistical (distributional) characteristics with respect to the property of interest."*

Furthermore, Gamberger & Lavrač (2002b) add that a subgroup description of the data should meet the following properties

- the subgroup's coverage is sufficiently large
- the subgroup has a bias towards target class coverage
- the subgroups are sufficiently diverse
- the subgroups are understandable, simple and actionable.

A more formal definition of the task is given by Zimmerman and de Raedt (2005) using their notation as shown above:

Given:

- $L = \{A = v \mid A \in \mathrm{A} \setminus \{\mathrm{A_t}\}, \mathrm{v} \in \mathrm{V[A]}\}$
- A data set E
- An interestingness measure σ
- $T = \{A_t = v \mid v \in V[A_t]\}$

Find:

The set of k rules expressible in L having the highest value of σ on E w.r.t. the given target literals T.

One has to be aware that in the original definition of Zimmermann and de Raedt (2005) the interestingness measure σ has been identified as WRAcc. This is due to the authors referring to the CN2-SD algorithm by Lavrac et al. (2004a). As one of this paper's aims is to present the literature on supervised pattern discovery, possible other instantiations of σ are presented in section 5.2. The definition of the target literals is only true for CN2-SD since it is possible to find rules for all possible values of an attribute. Using a different algorithm (i.e. Gamberger and Lavrač, 2002a), only a single literal from an attribute is chosen to be the property of interest in which case the target definition is the same as with correlated pattern mining. Concerning the goal of the search, it needs to be stated that rather than finding k rules, some algorithms presented use exhaustive search and do not limit the number of discovered patterns during the search but rather suggest choosing the best k pattern in a post-processing step. Wrobel (1997) has extended the problem of subgroup discovery to usage of multi-relational databases.

Most algorithms that will be described in this thesis have dealt with the issue of subgroup discovery. Those algorithms are CN2-SD, which is a variation of the CN2 algorithm aimed at subgroup discovery, algorithm SD, which is a simple beam search algorithm that uses a quality function that allows for user guided

search through a parameterizable quality function. Also the MESDIF (Multiobjective Evolutionary Subgroup DIscovery Fuzzy rules) algorithm is explicitly designed with subgroup discovery in mind. As opposed to the other algorithms, MESDIF employs a genetic search and multiple quality functions. Furthermore Apriori-SD, a subgroup discovery algorithm based on the Apriori algorithm and SD-Map are two algorithms that employ exhaustive search. Even though SD-Map was developed for subgroup discovery, the algorithm is capable of dealing with single value properties of interest only, rather than a range of values for a single attribute. A more detailed discussion of those algorithms is the subject of the next three chapters. The following section describes the related task of contrast set discovery as described by Bay and Pazzani (2004).

3.1.2 Contrast sets

Contrast set mining aims at finding differences between groups and has been introduced by Bay and Pazzani (2004). This is motivated largely by research questions derived from social research based on census data. The aim of contrast set discovery is to distinguish one or more groups against each other or contrasting the development of a group over different points of time. As opposed to other statistical techniques like time series analysis, contrast set mining deals with a multitude of observations at different points of time rather than observations throughout a time period. While in subgroup discovery the discussion evolves around finding subgroup descriptions which characterize a specific subset of examples, in contrast set discovery the aim is finding contrast sets, which help to differentiate groups against each other. In order to be able to do that, the examples are divided into groups which correspond to the classification of examples in the subgroup case.

The problem of contrast set mining is defined as follows.

A contrast set is a conjunction of attribute value pairs $(A_i,V(A_i))$ defined on groups $G_1,G_2,\ldots,G_n$ with no attribute occurring more than once. The groups are mutually exclusive meaning that $G_i \cap G_j = \{\} \forall i \neq j$.

Bay and Pazzani (2004) introduce also a notion of support for contrast sets with respect to the groups. "Support of a contrast set with respect to a group G is the percentage of examples in G where the contrast set is true."

Considering the formal definition given above one can describe contrast set mining as

Given:

- $L = \{A = v \mid A \in \mathrm{A} \setminus \{\mathrm{A_t}\}, \mathrm{v} \in \mathrm{V[A]}\}$
- A data set E
- An interestingness measure χ^2
- $T = \{A_t = v \mid v \in V[A_t]\}$

Find:

The set of k rules expressible in L having the highest value of χ^2 on E w.r.t. the given target literals T and different groups.

Note that here the interestingness measure is set to χ^2. The key difference in contrast set mining is when trying to evaluate the different contrast sets, one need to be aware of the different groups. That means rather than comparing the number of examples that are covered by a contrast set rule to the whole population, in contrast set mining the number of covered positive examples in one group is compared to the number of covered positive examples in all other groups independently. Using the χ^2 measure one can find significant deviations between the groups and therefore find interesting contrast sets.

As shown by Kralj et al. (2007) contrast set mining is so similar to the task of subgroup discovery that it is possible to use subgroup discovery algorithms for contrast set mining. This is done by sequentially applying a subgroup discovery algorithm to all groups that should be contrasted individually. Webb et at (2003) also found that contrast mining can be tackled using the so called Opus_AR rule learning algorithm which discovers association rules though it does not rely on the frequent itemset paradigm. He therefore concludes that contrast set mining is a special case of rule learning.

In this paper, STUCCO (Bay and Pazzani, 2003) is presented since it is the only algorithm that has been developed especially for this problem formulation. In the final chapter, GroupSAX is presented to show how contrast set mining can be done on sequential data. The next section discusses correlated pattern mining.

3.1.3 Correlated Pattern Mining

Correlated pattern mining has been established by Morishita and Sese (2002). It is motivated through the lack of interestingness of some association rules despite high confidence. Therefore the authors use a correlation measure in order to determine which items are actually correlated and thus are deemed interesting. Zimmermann and de Raedt (2005) formulate the problem of correlated pattern mining in terms of cluster grouping as follows:

Let $\mathrm{I} = \{I_1,...,I_z\}, \forall I \in \mathrm{I} : V[I] = \{false, true\}$ be the set of items,

Given:

- $L = \{I = true \mid I \in \mathrm{I}\} \setminus$
- E a transaction database
- σ is a correlation measure such as χ^2
- T is a single literal $l \in L$

As Zimmermann and de Raedt point out, the approach suggested by Morishita and Sese is restricted to itemsets that appear together and are therefore

positively correlated. Nevertheless by also accepting literals set to false, it would be possible to mine for negative correlation, therefore finding rules such as e.g. item x is rarely bought together with item y. A similar though more general approach to this kind of itemset mining without the definition of a target literal has been suggested by Silverstein et al. (1998)

A specific algorithm for correlated pattern mining is not discussed in this paper since the CU-algorithm (Zimmerman and de Raedt, 2005) for general cluster grouping problems is presented. It is based on algorithms for correlated pattern mining developed by Morishita and Sese (2002). The next section discusses the rather different approach of exception rule mining which aims at finding local patterns within all examples covered by a strong rule.

3.2 Exception Rules

The idea of exception rule discovery (Suzuki, 2004) is that it is possible to increase the accuracy of a rule r by finding a significant exception population for r, which is described by an exception rule. Hereby, the examples covered by rule r are partitioned into true positive (TP) and false positive (FP) examples. Among the FP examples exception rule discovery tries to find large groups that can be considered a regular exception to the rule. The authors cite the example rule that “fastening your seatbelt is safe” is common knowledge, though the exception rule is “using a seatbelt is risky for a child”. As can be seen from this example, discovery of exception rules can lead to better knowledge discovery results. In terms of the discussion of supervised local patterns, the total example space is partitioned twice, first by discovering a strong rule and second by searching for a meaningful exception. This means we are looking for local patterns within patterns that could be described as local as well.

The authors define a conjunction rule as a rule represented by a conjunction of attribute value-pairs as the rule body and single attribute-value pair as the rule head. The aim of exception rule discovery is to find pairs of rules which include the strong rule with its associated exception rule. Therefore the aim is to find a

strong rule $b_\mu = v_1 \wedge v_2 \wedge v_3 \wedge ... v_\mu \Rightarrow h$ with h being a single attribute-value pair, with an exception rule $b'_v = v'_1 \wedge v'_2 \wedge v'_3 \wedge ... v'_v \rightarrow h'$ with h' being an attribute-value pair of the same attribute as h but of a different value. The exception rule is fully stated as: "if b_μ then h and if V_μ and V'_v then h'". The discovered pattern is represented as a rule pair.

$$r(h,h',b_\mu,b_v') \equiv \begin{cases} b_\mu & \rightarrow h \\ b_\mu \wedge b_v' & \rightarrow h' \end{cases}$$
$$\mu,\nu \leq M$$

With M being the user specified maximum number of attribute-value pairs in the rule body.

Considering the definition of pattern detection given in section 2.3 as a relatively small phenomenon, exception rules appear to fit rather strict. While the strong rule could be considered a global pattern, the corresponding exception rule defines a local pattern which deviates significantly from the distribution created by the strong rule and can still be considered a small phenomenon with regard to the base distribution.

Describing the task using the definition scheme above, one can define the task of exception rule mining as follows:

Given:

- $L = \{A = v \mid A \in \mathrm{A} \setminus \{\mathrm{A_t}\}, \mathrm{v} \in \mathrm{V[A]}\}$
  with $\left|\{A = v \mid A \in \mathrm{A} \setminus \{\mathrm{A_t}\}, \mathrm{v} \in \mathrm{V[A]}\}\right| \leq M$ defining the maximum length for each possible rule.
- A data set E
- $\sigma = ACEP(h,b_\mu,h',b_v')$
- $T = \{A_t = v \mid v \in V[A_t]\}$

Find:

The set of k rule pairs (b_μ, b_v') expressible in L having the highest value of σ on E w.r.t. the given target literals T.

3.3 Conclusion

The aim of this chapter was to present the different problem formulations that have been used in order to motivate supervised local pattern discovery. From the problem formulations one can see that the general goal is to discover interesting knowledge which describes that data.

The research questions posed by the different problem definitions can be described as finding the most interesting subsets of examples in the data for the case of subgroup discovery. This can be used to answer questions like, what common properties do buyers of a certain product share. In the case of contrast set mining, the question is what properties distinguish different groups of examples. Here the question could be described as, what common properties do buyers of a certain product distinguish from those who do not buy that particular product and what properties do those examples have. Correlated pattern mining asks the question of when a discovered pattern has to be considered interesting Research questions that can be answered using correlated pattern mining are similar to that of subgroup discovery, but discovered patterns are validated to be correlated. Exception rule mining was motivated through improving predictive accuracy of rules covering a large part of the example space. It can be considered as local pattern mining since the exception takes into account locality explicitly. As can be seen, the research questions that were at the at the focus of discussion when developing the problems all look at the problem of local pattern discovery from different angles or simply put a different focus on similar problems like subgroup discovery and correlated pattern mining.

Nevertheless, the problem definitions can be considered similar and the algorithms that are applied can typically be used for either task. This is most obvious in the case of cluster grouping algorithms. As discussed by Kralj et al.

(2007) the differences between subgroup discovery and contrast set mining are terms and interpretation of results rather than being actual different tasks. The Cluster Grouping problem formulation works as a superset of those different problem statements and the CU algorithm is able to complete all the related tasks. Nevertheless, also the other algorithms that have been developed for the different subtasks are able to perform the tasks equivalently. Exception Rule discovery is another issue. Since the goal is to find rule pairs, it is not directly compatible with the other tasks. Yet it is possible to change the way the algorithms which have been developed for the other tasks work by first letting them discover large group descriptions and then letting them search for subgroups within those group descriptions. Therefore these tasks can all be considered similar.

4 Data preprocessing and data quality

This chapter is the first chapter to discuss the different algorithms for SLPD. The algorithms that are described are CN2-SD, Algo-SD, MESDIF, Apriori-SD, SD-Map, CG-Algorithm, STUCCO as well as MEPRO. This chapter is concerned with the task of data preprocessing and data quality issues. Data preprocessing addresses things like conversion of the data structure to fit a specific algorithm and feature selection. This is not the focus of the algorithms, but rather a necessity in order to be able to deal with different problems that can lead to problems such as a too large search space with continuous variables or missing data. The goal of this chapter is to present obstacles that have to be faced when applying any algorithm for supervised local pattern discovery. Also elective procedures which may make searching easier have been discussed in the literature and should therefore be mentioned in this chapter.

The algorithms studied here are mostly using symbolic representation of the data and are therefore unable to cope with continuous variables. Furthermore there are several other issues discussed in this section that are being tackled during the preprocessing phase. There is the topic of missing values for which solutions have been suggested in some algorithms individually while others suggest more general possibilities. Furthermore the data structure has to be adapted to the individual algorithms. So far we assume that all examples are stored in a database using a symbolic representation for all attribute values. Some algorithms require special data formats to work efficiently, consequently it is necessary to preprocess the data in a way that it can be used in accord with a specific algorithm. Finally it can also be useful to apply feature subset selection as suggested by Kavŝec, et al. (2003). As can be seen from the description, there are challenges that are algorithm specific, while others are algorithm independent such as feature subset selection. All these questions are discussed in more detail next.

```
procedure PREPROCESSDATA(Examples)

1        For all {A | V[A] = ∞} do
2            DISCRETIZE( A )
3        CHANGEDATASTRUCTURE(EXAMPLES)
4        INCORPORATEMISSINGVALUES(EXAMPLES)
5        PERFORMFEATURESUBSETSELECTION
```

Figure 2.PreprocessData pseudo code

Considering the program shown in Figure 1, this chapter implements the PREPROCESSDATA(Examples) procedure. In that regard Figure 2 presents the pseudo code that describes the actions that can be undertaken during the preprocessing phase. Note that again some of the measures, such as discretization are nearly universally needed, while others such as changes in the data structure would be needed only for SD-Map as well as MESDIF and Apriori-SD.

Lines 1-2 describe the discretization of continuous attributes which is discussed in section 4.1. This section describes the different discretization algorithms which can instantiate the *DISCRETIZE(A)* procedure though a more detailed implementation is not shown. Line 3 and 4 discuss the procedures for changing the data structure and dealing with missing values which are described in section 4.2 and 4.3. Since there are several possibilities to implement the issues discussed in lines 2-4 which do not necessarily share many common traits a detailed pseudo code implementation is omitted. Lines 5 present a function for feature selection which is discussed in section 4.4.

4.1 Discretization of continuous variables

As has been mentioned already this discretization has to be performed for nearly all algorithms presented here. The only exception of sorts is the MESDIF

algorithm presented by Berlanga et al. (2006). The authors use fuzzy rules in order to cope with continuous variables. This is still a kind of discretization though it is performed directly during the runtime of the search algorithm rather than in a preprocessing step. Discretization has been a challenge already for rule learning algorithms and association rule learners even before trying to detect local patterns. Therefore possible solutions can be found in the literature for rule learning. Since here we are concerned with labeled data and therefore with supervised learning, it is possible to employ methods developed for labeled data. Two possible solutions are chi-merge algorithm by Kerber (1992) or Entropy Split by Fayyad and Irani (1993).

Chi-merge is a bottom-up algorithm that first creates an interval for every value that can be found in at least one example and sorts them either ascending or descending. Adjacent intervals are merged if the distribution of positive and negative examples is independent. To decide whether the distribution is independent, the chi value for those two intervals is calculated.

$$\chi^2 = \sum_{i=1}^{2}\sum_{j=1}^{c}\frac{(O_{ij}-E_{ij})^2}{E_{ij}}$$

O_{ij} is the number of examples in the i-th interval that are of class j and the E_{ij} is the expected frequency. This is the number of examples in the intervals times the a priori frequency for the whole distribution for positive and negative examples. Those two intervals which have the smallest χ^2 value are merged. This is done as long as there are more intervals than desired or as long as there is still at least one pair of two adjacent intervals that have a χ^2 value less than a predefined threshold.

Entropy Split uses a top down approach. First, all values that are present in any example are added to a single interval and ordered according to the value. Next compute a split point for all values which represent a class change in the ordered list. This is done by computing the entropy value for a specific split point. This is calculated as

$$E\text{-}Score(A,B,E)=\frac{|E_{A<B}|}{|E|}Entropy(E_{A<B})+\frac{|E_{A\geq B}|}{|E|}Entropy(E_{A\geq B})$$

E is the data set, A is the attribute in question and B is the boundary. Furthermore, $E_{A<B}$ describes the set of examples where the value is less than the boundary B, $E_{A\geq B}$ describes the set of examples where the value is equal or larger than the boundary and entropy is calculated as:

$$Entropy(E)=-\frac{|E_+|}{|E|}\log_2\frac{|E_+|}{|E|}-\frac{|E_-|}{E}\log_2\frac{|E_-|}{|E|}$$

with E_+ representing all positive examples of the set and E_- representing all negative examples. The E-score is calculated for all possible split points and the split is performed where the score is minimal. This procedure is then recursively continued for the different partitions until the number of partitions that was

Alternatively it is of course possible to use known domain specific discretizations such as discretizing age (0-18) as baby (0-3), child (4-6), school child(7-10) and teenager (11-18) or using equal width intervals or equal frequency kind of discretizing algorithms which are always applicable, even in cases when working with unlabeled data.

4.2 Data Structure

We assume that all algorithms are capable of working with the standard symbolic representation of the data as depicted in Table 1. Nevertheless some algorithms rely on a specific data format in order to be efficient. Therefore this section presents alternative data formats as used by the algorithm presented by

	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_0
Domain(A)	{a,s}	{f,k}	{c,o}	{m,h,e}	{p,l,j}	{b,d}	{x}	{y}	{z}	{true, false}
1	a	f	c	m	p	d	x	y	z	True
2	a	f	c	m	l	b	x	y	z	True
3	s	f	o	h	j	b	x	y	z	False
4	s	k	c	e	p	b	x	y	z	False
5	a	f	c	m	p	d	x	y	z	true

Table 1. Example Standard Data Representation

Atzmüller and Puppe (2006) in their SD-Map algorithm as well as the dataformat used by Apriori-SD (Kavŝec, Lavrač, & Krastačić. 2003). Changing the data structure according to the needs of a specific algorithm could be performed in an additional step when performing the search which relies on the data structure. Despite this change, the data structure is described as an option during preprocessing since it makes it possible to implement the solution of dealing with missing values (e.g. 4.3) according to Atzmüller and Puppe (2006) even if the complete algorithm should not be used.

Table 1 shows the standard data representation. Row 2 shows the domain of the attributes and rows 3 to 7 show example data. Attribute A_0 describes the attribute of interest.

4.2.1 FP-Tree

The data structure chosen by Atzmüller and Puppe (2006) for the SD-Map algorithm is the Frequent Pattern tree (Han et al., 2004) that has been developed in the context of association rule and frequent pattern mining. The FP-growth algorithm uses only two passes through all examples e_i in the database in order to find all frequent itemsets. In the first pass, it counts how often each attribute A_j is found in all the examples and sorts the attributes in descending order in a sorted list L. Attributes that occur less than the minimal user specified support are ignored, hence the elements of L are a subset of all possible attribute values $V[A]$. In a second pass through the data, the FP-tree is build. The FP-tree consists of nodes for each attribute counting how often it has

ID	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	Attribute of interest	Ordered frequent items
1	a	f	c	m	p	d	x	y	z	True	x,y,z,f,c,a,m,p
2	a	f	c	m	l	b	x	y	z	True	x,y,z,f,c,a,b,m
3	s	f	o	h	j	b	x	y	z	False	x,y,z ,f,b
4	s	k	c	e	p	b	x	y	z	False	x,y,z, c,b,p
5	a	f	c	m	p	d	x	y	z	True	x,y,z, f,c,a,m,p

Table 2. Input Data FP-growth

been found in the data. For the SD-Map algorithm, the nodes contain the number of times an attribute has been found in the data, though the count is split into positive and negative examples. Attributes that are found in different parts of the tree are linked through a node-link for discovery of frequent patterns and initial node-links are stored in a header table. The creation of the data structure used in the FP-growth algorithm is illustrated based on an example of

Han et al. (2004) though it has been extended to incorporate the changes made by Atzmüller and Puppe. The example contains building the tree according to seen examples which is presented next and is continued in section 5.1.3.2 where discovery of frequent itemsets is presented. Notice that the minimum support for a frequent itemset is set to 3.

→ Frequent itemsets: x:5,y:5,z:5, f:4,c:4,a:3,b:3,m:3:p:3

To build the FP-tree, the ordered frequent items are used as seen in column four in Table 2. Column three describes whether an example has the property of interest or whether it does not. Taking the first example leads to a single-path tree xyzfcamp with count 1|0 for each node with the one representing one positive example and the zero representing that so far no negative examples have been found here. The next example (200) increases the counts to 2|0 for xyzfca and creates a new branch bm with counts 1|0 for each element. The third example increases the counts for xyzf and branches out to b:1|0 and so

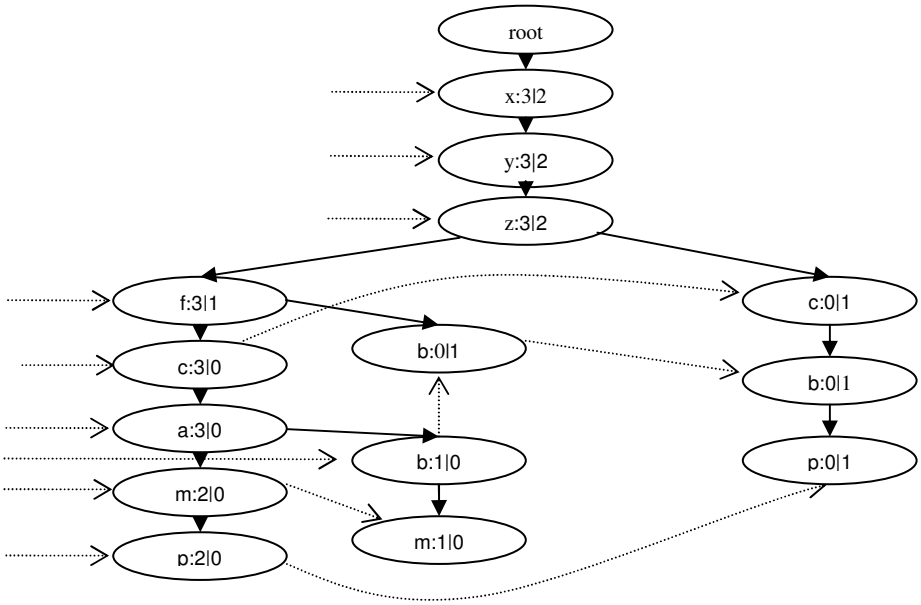


Figure 3.FP-growth Tree

on. Finding those examples and creating the FP-tree will result in the tree as seen in Figure 3.

This section has shown how the data structure of the FP-Tree is build. It is useful to do this already in the preprocessing stage as this offers an alternative opportunity to deal with missing values and thus provides more flexibility. More information on how to use this data structure in order to find frequent patterns and therefore local patterns is provided in section 5.1.3.2.

4.2.2 Binary Vectors

In the Apriori-SD (Kavŝec, Lavrač, & Krastačić. 2003) case the authors specifically state that the algorithm runs on a binarized database in which every attribute is represented as a binary vector where for every example exactly one element is set to 1 representing the value of the attribute. In cases of missing values the vector will consist only of 0s. In the case of MESDIF (Berlanga et al., 2006) the authors do not explicitly demand such a data structure, yet, since

rules are internally represented as binary vectors it can be useful to work with the same data representation as in the Apriori-SD case.

4.3 Missing variables

Another important problem are missing values. Incomplete data can be problematic since too many incomplete examples could misguide a learner or pattern discovery agent to believe there is a certain pattern when actually there is not. The easiest way to deal with incomplete data is to simply ignore incomplete examples completely. Unfortunately this can often lead to a situation in which rather few examples are left for the data mining process. Second, Lavrač et al. (2004b) suggest training a rule learner such as CN2 (Clark & Niblett, 1989) to predict the missing values rather than ignoring all or parts of the examples

Third, it is possible to ignore incomplete data in a case by case approach for each rule. While creating a new rule only those examples are counted that fulfill the rule body and therefore have no missing values concerning the attributes in the rule. The problem is that when trying to find statistically relevant subgroups the distribution of elements concerning the rule body is usually compared to the total distribution in the total set of elements. Due to the missing attributes one cannot simply use the total numbers of the base distribution since the value for an attribute is unknown and therefore it could satisfy the rule body or not. For that reason one needs to subtract all those examples that have missing values on attributes in the rule body. This problem is addressed by Atzmüller and Puppe (2006). They use a special data structure derived from their approach to develop an efficient way of counting all those examples that need to be taken into account as the base distribution. Hereby, they create a FP-tree which counts missing values for each attribute according to whether this example is a positive or a negative one. This FP-tree is used in order to calculate the number of complete examples for each rule using attributes that have missing values. Since the data structure of the FP-tree has already been discussed the next

ID	A1	A2	A3	A4	A5	Frequent Items
1	a	c	e	missing	true	ace
2	a	c	e	h	true	ace
3	missing	d	e	g	true	de
4	a	c	missing	g	true	ac
5	missing	d	f	h	true	d
6	b	d	f	i	true	d
7	b	C	f	missing	false	c
8	b	C	e	g	false	ce
9	missing	D	missing	h	false	d
10	a	C	e	i	false	ace

Table 3. Input data for SD-map example

paragraphs discuss an example of how to use the structure to count missing values.

Next, an example will describe how to create the missing-FP-tree. Consider the data in 0. Attributes A_1 to A_5 are the attributes that describe the examples, attribute A_5 is the target attribute and the target value is true. The possible values are {a,b,c,d,e,f,g,i} with dom(A_1)={a,b}, dom(A_2)={c,d}, dom(A_3)={e,f}, dom(A_4)={g,h,i}, dom(A_5)={a,b}. The total number of positives is six and the total number of negative examples is 4. A total of 6 examples have missing values and there are in total 7 values missing.

Now the missing-FP tree is created. Since only values from the attributes A_1 to A_3 have been used in building the tree only those attributes have to be considered. The first example only contains a missing value for attribute A_4 therefore there is no addition to the tree. The third example has a missing value for attribute A_1 therefore a first node A_1:1|0 is created since the example is positive. Next, example 4 adds another node A_3:1|0 and example 5 increases the TP count on A_1:2|0. Finally example 9 increases the FP count on A_1:2|1 and creates a new node A_3:0|1 resulting in a tree as depicted in Figure 4.

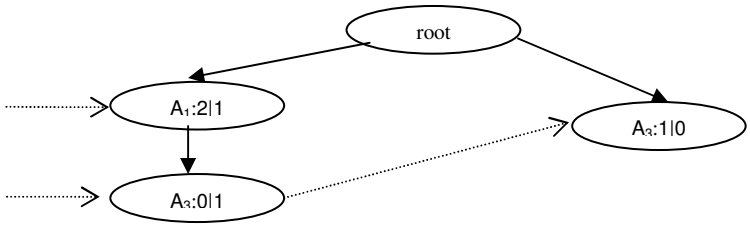


Figure 4.Missing-FP-Tree for SD-Map example

The tree can now be used in calculating the quality measures for specific rules. Some examples are shown now to clarify how this can be done. Consider one would like to create a rule $e \Rightarrow a$. There are three examples that support the rule and two examples which would be counted as false positives. To evaluate the goodness of the rule we need the missing-FP-tree since we cannot use the base distribution p_0=0.6 as the relative frequency for all positive examples since not all examples actually have a value for A_3. Therefore it is necessary to count only those that have a value there using the missing FP-tree. We can see in the tree that there is one positive example that has no value for A_3 and one negative example which we cannot use. Thus p_0=0.625. With p=0.6 this is a rather poor subgroup. To show how to calculate the values of all true positives TP and all false positives FP which can be used in a quality function we chose pattern ae:2|1 which implies a rule $a \wedge e \Rightarrow a$. In this examples the problem is to count all the examples that have missing values for attribute A_1 or A_3 but not to count those examples twice that have missing values in both attributes. This can be achieved by using the formula as presented by Atzmueller and Puppe:

$$\text{missing}(A_1 \vee ... \vee A_n) = \sum_{i=1}^{n} \text{missing}(A_i) - \sum_{m \in 2^M} \text{missing}(m)$$

With $|m| \geq 2$ and $M = \{A_1, ..., A_n\}$. This means one need to add all missing attributes according to the missing-FP-tree and then remove all those examples that have been counted more than once. In this example it means for $TP_{missing}$ that we add (2+0+1)-0=3 and for $FP_{missing}$ we add (1+1+0)-(1)=1. Therefore the adjusted TP'=TP I- $TP_{missing}$=6-3=3 and FP'=FP-$FP_{missing}$=4-1=3. Using these adjusted values it is possible to compute quality functions which are based on the total number N'=TP'+FP' or on TP' and FP'. In this case the rule $a \wedge e \Rightarrow a$ has a value of p=0.66 while p_0=0.5, which at least is an improvement over the original distribution.

4.4 Feature Subset selection

One of the few algorithms that suggest the use of feature subset selection is the Apriori-C algorithm which is used as the basis of Apriori-SD. This is meant to improve predictive quality and to avoid problems with data containing too many attributes which leads to a huge search space. Jovanoski and Lavrač (2001) examined different feature selection options such as statistical correlation, odds ratio and deviations of the RELIEF algorithm. They concluded with suggesting the use of the odds ratio algorithm which performed best in their evaluations. Odds ratio is computed as:

$$OddsRatio(A) = \log \frac{odds(A \mid C_1)}{odds(A \mid C_2)}, where\ \ odds(A) = \frac{P(A)}{1 - P(A)};\ P(A) \neq 0 \wedge P(A) \neq 1$$

Here, C_1 is the target class and C_2 the union of all non-target classes and n the number of examples. *P(A)* represents the probability of an attribute calculated as the relative frequency. The attributes are ordered according to descending odds ratio, that is all attributes are included that are greater than a user specified cut-off parameter. The reason for using only a user specified number of features as discussed by Jovanoski and Lavrač (2001) is the problem of exponential runtime behavior of the Apriori search algorithm with regard to the number of frequent features. Being able to discard some features therefore can significantly speed up the search.

4.5 Conclusion

This chapter has discussed some important premises that have to be met before it is possible to start with the search for local patterns. Most of the topics discussed here are equally important for local pattern discovery as they are for rule learning. Hence, rarely any specific algorithms to deal with those topics have been developed. Table 4 summarizes the approach taken by different algorithms. “Not Specified” in line 2 means that the authors do not suggest a specific algorithm to be used for discretization though discretization as such is necessary. Also, missing values have to be dealt with in any algorithm, though only SD-Map incorporates a method in the algorithm itself. Feature Subset Selection (FSS) is a optional action that might help in improving the speed or the accuracy of the algorithm, but it is not necessary. As can be seen, it was suggested only for use with the Apriori-SD algorithm.

Key processes in pre-processing are ways to deal with missing values and discretization. Unhandled, both problems may either make attributes unusable or lead to distorted results. Therefore both processes are a necessity for every pattern discovery algorithm.

Data structure could be considered a mere technical choice, but some algorithms, like the SD-MAP algorithm gain their efficiency from a specialized data structure. For other algorithms, like the Apriori-SD, the choice of a binary vector is due to the origins of the Apriori algorithm which works on transactional data in which every item of a transaction is represented by a one.

		CN2-SD	Algo-SD	MESDIF	Apriori-SD	STUCCO	CU-Algo	SD-MAP
Discretization	Not Specified	x	x		x	x	x	x
	Fuzzy logic			x				
Data Structure	Standard	x	x			x	x	
	Binary Vector		x	x	x			
	FP-Tree							x
Missing Values								x
FSS					x			

Table 4. Summary of pre-processing by algorithm

After pre-processing, the search for local patterns is performed. This is the topic of the next chapter. The search is the essential part of the discovery process and can be split into two key activities. First, the search itself and second the measurement of pattern quality. These two activities are the key points of the next chapter.

5 Search

This chapter describes the two central problems of pattern discovery algorithms, first the question on how to efficiently search for patterns in a potentially huge search space and second how to evaluate different patterns. Generally one can distinguish between two types of search, first, heuristic search, which uses an evaluation function to rank discovered solutions and typically refines those solutions that present the best solutions so far. Second there is exhaustive search. Exhaustive search aims at searching the total search-space for good solutions. Since such an effort can be considered futile if performed brute force, exhaustive search algorithms introduce sensible restrictions on what patterns should be like. Typically those restrictions are minimum frequency restrictions which guarantee that a pattern does cover at least a minimum number of examples. Using those restrictions, large parts of the search space can be pruned and therefore exhaustive search can be performed efficiently.

The second part of this chapter discusses quality functions. As said already, quality functions are used to rank discovered patterns and lead the search. They can also be used to limit the number of patterns that are presented to the user to the best n patterns. Quality functions can be taken from inductive statistical analysis to test for significant deviations of a pattern from the original distribution. Measurement of good rules is often done using a coverage approach, which traditionally removes examples that are part of an already discovered pattern. In the area of subgroup discovery reweighting examples rather than removing them has been used frequently. Therefore, different weighting schemes are presented in the third section of this chapter.

```
    procedure FINDLOCALPATTERNS (Examples)
1   Patterns={}
2   CandidatePatterns={ }
3   CandidatePatterns=GenerateCandidatePatterns(Examples)
4   while POSITIVE(Examples) ≠ { }
5           Pattern=FINDBESTPATTERN(<Examples,ExampleWeights>,CandidatePatterns)
6           Covered=COVER(Pattern,<Examples, ExampleWeights>)
7           if STOPPINGCRITERION(Patterns,Pattern,<Examples,ExampleWeights>)
8                   exit while
9           ExampleWeights =COMPUTEEXAMPLEWEIGHTS(<Covered,ExampleWeights>)
10          Patterns=Patterns ∪ Pattern
11          CandidatePatterns=CandidatePatterns \ Patterns
12  return(Patterns)

    procedure FINDBESTPATTERN(<Examples,ExampleWeights>, CandidatePatterns)
13  InitPattern=INITIALIZEPATTERN(<Examples,ExampleWeights>,CandidatePattern)
14  InitVal=EVALUATERULE(InitPattern)
15  BestPattern=<InitVal,InitPattern>
16  Patterns={BestPattern}
17  While Patterns≠ { }
18  Candidates=SELECTCANDIDATES(Patterns,<Examples,ExampleWeights>, CandidatePatterns)
19          Patterns=Patterns \ Candidates
20          For Candidate ∈ Candidates
21                  Refinements=REFINERULE(Candidate,Examples)
22                  For Refinement∈ Refinements
23                          Evaluation=EVALUATEPATTERN(Refinement,
24                                                  <Examples,ExampleWeights>)
                            Unless STOPPINGCRITERION(Refinement, <Examples,
25                                                          ExampleWeights>)
                          NewPattern=<Evaluation,Refinement>
26                        Patterns=INSERTSORT(NewPattern,Patterns)
27                        If NewPattern>BestPattern
28                                BestPattern=NewPattern
29                  Patterns=FILTERPATTERNS(Patterns,<Examples,ExampleWeights>)
30  return(BestRule)
```

Figure 5.Generic pattern discovery algorithm

Before going into the details of how the search for local patterns is organized in the respective algorithms, some general discussion on how search can be performed should come first. To present the way those algorithms work, Figure 5 presents a generic pattern discovery algorithm which is based on the generic rule learning algorithm as presented in Fürnkranz (1999).

As opposed to the generic rule learner discussed by Fürnkranz (1999), two changes have been introduced. First, this learner includes working with example weights which has been applied in CN2-SD, Apriori-SD as well as the MESDIF algorithm. Second, a new function, GENERATECANDIDATEPATTERNS, has been introduced in the FINDLOCALPATTERNS routine. The GENERATECANDIDATEPATTERNS function is implemented only by those algorithms that apply an adapted version of an association rule mining algorithm. It is discussed in more detail in section 5.1.3

Essentially, the algorithm tries to discover a best pattern using the FINDBESTPATTERN function and then decreases the weight of all examples that are covered by the pattern. This weight decrease can be done in different ways and is discussed in section 5.3. Function FINDBESTPATTERN uses a heuristic search for good patterns by starting with an initial pattern which is iteratively refined until a stopping criterion is fulfilled. The best pattern is chosen during each refinement. A list of patterns is kept in cases where more than a single pattern is further refined. Otherwise the FILTERPATTERN function returns only a single best pattern. To decide which refinement is best, an evaluation is performed using a quality function which is discussed in section 5.2. Those patterns that were discovered are applied in the COVER method which returns all examples that are covered by the pattern. For those examples, the COMPUTEEXAMPLEWEIGHTS changes the example weights according to a user specified weighing scheme. Weighing schemes are presented in 5.3. The FINDBESTPATTERN function is the essential function for all heuristic search algorithms. Those algorithms that apply an adapted version of an association rule mining algorithm typically only use the function as a post-

processing step. Since GENERATECANDIDATEPATTERNS already discovered all patterns in the case of exhaustive search, FINDBESTPATTERN in this case is only used as a filter to minimize the number of patterns that are presented to the user.

Before going into details described above, some general notes on search strategies are presented first.

5.1 Search type

This part describes the search in more detail. The next section describes general search strategies that can be applied when considering local pattern discovery as search. Sections 5.1.2 and 5.1.3 describe the search as it is performed by the algorithms. First, heuristic search is presented, which is used by CN2-SD, algo-SD and MESDIF and second, exhaustive search algorithms are presented, such as Apriori-SD, CG-Algorithm and SD-Map.

5.1.1 Search strategies

As Fürnkranz (1999) elaborates for rule discovering, there are three principal search strategies. These strategies decide how, among all possible rules, candidate rules are generated. Patterns or rules can be organized in the generality lattice of the hypothesis space. A lattice is a partial ordering of all attribute-value combinations that can possibly be used to create patterns. The ordering follows the relation “is more general then”. For instance, if there are two patterns,

P_1: (?,R,?)=>True

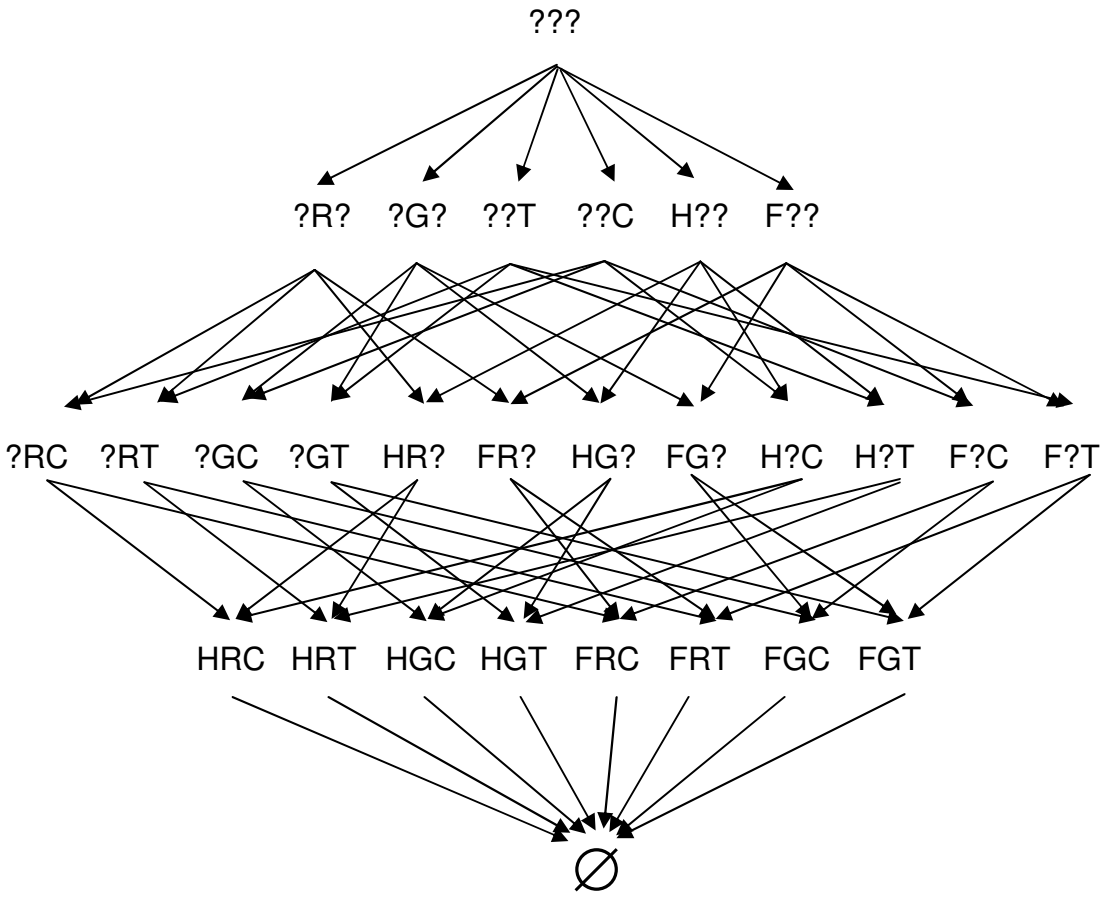
P_2: (?,R,T)=>True

the first one covers all examples that are covered by P_2 plus those that do not have T as the third value. Therefore the first pattern is more general than the second pattern. A third pattern, e.g. (H,?,?)=>True is neither more general nor

more specific, therefore, generality only describes a partial order. Figure 6 describes this situation for three attributes with two values each.

Figure 6.An example generality lattice for three attributes with two values for each attribute.

There are three ways to traversing the lattice. First, an algorithm can use a top-down approach, which means to pass through the lattice starting from the most general rule through iterative specializations to a specific rule. Alternatively, rules can be generated through bottom-up search by employing a specific example as a starting rule which covers exactly one example. This single example rule is then iteratively generalized. The third possibility is to start a bidirectional search which combines both approaches.



In those pattern discovery algorithms studied here, the top-down strategy prevails. Since it is used in the CN2 algorithm, CN2-SD makes no changes to the search strategy. Also those algorithms based on Apriori style mechanisms usually climb through the lattice in a top-down fashion. There is also a case in which the bidirectional search is applied. In the MESDIF algorithm the authors use a genetic search algorithm (5.1.2) to find the best rules by combining the best rules reached so far and then performing generalization or specialization operations randomly. Therefore it cannot always be made sure that the new rule is a refinement of any of both parent rules. An explicit use of a bottom-up approach has not been found in the literature.

5.1.2 Heuristic search

This section goes briefly over the issue of different heuristic search algorithms that have been developed and applied successfully. Search algorithms have been discussed in length in the literature (i.e. Russell & Norvig, 2003, Domschke & Drexl, 2002, Fürnkranz 1999). Here, a brief general outline of the concept is given and then the local pattern discovery algorithms that apply it are shown. Heuristic searches often find good though not necessarily optimal solutions. In local pattern discovery two approaches have been used. Most often, the beam search algorithm has been applied in this context. Despite the fact that beam search has been used often, other possible search strategies could be used as well for the pattern discovery. Secondly, a stochastic search has also been applied in form of genetic search. When considering the algorithm description in Figure 5, please note that none of the algorithms of this section implement the GenerateCandidatePatterns function, therefore CandidatePatterns remains an empty set throughout the runtime of the algorithm.

5.1.2.1 Algorithms based on beam search

Beam search uses a beam of k rules that are refined during a single loop. Among all refinements the best k examples are chosen and further refined until a stopping criterion is fulfilled. Beam search is an extension of the Hill Climbing algorithm that only refines a single best rule, rather than the k best rules. The algorithm as outlined here has the advantage of being efficient, though, as all heuristic approaches it does not guarantee finding the best possible solution. The algorithm may not necessarily perform better than a hill-climbing algorithm. This is due to the possibility that during the search, one local optimum may dominate even the global optimum. Therefore it is likely, that local optima will mislead the search. The simultaneous refinement of the k rules in the beam does not necessarily mitigate this problem. Especially, though not exclusively in cases where the beam is chosen relatively narrow this behavior can happen. In

terms of the description in Figure 5, beam search can be implemented in function FILTERPATTERNS which then filters out all but the best k patterns.

Beam search has been applied in rule discovery amongst others by Clark and Niblett (1989) when designing the CN2 algorithm. For that reason it has been applied in CN2-SD (Lavrač et al., 2004a). Beam search is also used for alternative approaches to subgroup discovery such as the expert-guided search using algorithm SD (Gamberger and Lavrač, 2002). In the case of algorithm SD the authors only try to discover a number of n best rules, with n being a user specified value which is used as the beam size. Therefore, after a certain stopping criterion is matched, the algorithm simply returns all patterns in the beam to the user. They suggest choosing a large beam and postprocessing the discovered patterns to only present the best k<n best patterns. Considering the algorithm given in Figure 5, the return(bestpattern) command should be altered to return the set of rules in the beam. Refinement is performed by both algorithms by creating all possible specializations for all rules in the beam. Refinements are considered to become part of the beam only if they are improvements over the more general rule in order since more general rules are typically preferred since they are expected to generalize better on unknown data.

5.1.2.2 Genetic algorithms

Genetic algorithms are described as a variation to stochastic beam search (Russell & Norvig, 2003). Rather than randomly choosing the k successor rules, in genetic algorithms the different rules are combined in a way which is described as sexual reproduction. As in the case of beam search, the best k rules are stored and referred to as the population. A fitness function or quality function is used to rank the members of the population. Among all members of the population some are selected that are used for cross-breeding. Hereby there is a crossover between both rules A and B. This means that the rules are seen as a number of n attribute value combinations and the new rules consists of the attribute-value combinations 1…h of rule A and h+1…n of rule B with

h<n. The choice of the value h in which to separate and mix the different rules can be arbitrary. Finally, mutation adds a random element which changes the value of a single attribute value combination with some low probability. This helps avoiding local extremities. The exact implementation is algorithm specific. Mutation can remove a single attribute or swap a specific value for a different $v \in V[A]$. It is also possible to add a random attribute value combination to the rule.

Next I discuss the MESDIF search algorithm which employs genetic search. When implementing a genetic algorithm based on the generic algorithm depicted in Figure 5, the most important changes have to be to the REFINEPATTERN process. Here the genetic operations, recombination or breeding and mutation are implemented.

The MESDIF algorithm (Berlanga et al., 2006) applies a genetic search using two population sets. The initial population consists of all examples and the second, namely the elite population, is the part of the population that dominates most other individuals. Domination is the fitness function discussed above. In the MESDIF case, domination consists of three independent goals that are to be maximized at once. For each example or rule, the number of examples that it dominates according to the three goals, is calculated and the best rules are stored in the elite population. Each individual is represented as a rule which in turn is represented as a binary vector where every binary digit corresponds to a variable $v \in V[A]$. The individuals are taken from the elite population and bred with other members of the elite population using binary tournament selection with crossover and mutation. Mutation requires that at least half of all mutation operations call for a flip of a one to a zero, thereby generalizing the rules and therefore increasing the coverage. Since there is no inherent stopping criterion in the algorithm like in the case of separate and conquer rule learning, a stopping criterion has to be stated explicitly. This is done by stating the number of runs that are performed by the genetic search.

As this section has shown, there are three algorithms that have been developed based on heuristic rule learning. CN2-SD and Algo-SD are based on beam search and MESDIF is based on genetic search. All three have been developed in the context of subgroup discovery. Besides the search strategy, those algorithms distinguish themselves through the use of different quality functions. This will be discussed in section 5.2. Before that, the search strategy of the other algorithms is presented. They are implementing exhaustive search, mostly based on association rule mining and frequent pattern discovery.

5.1.3 Exhaustive searches

Besides the aforementioned heuristic search strategies, many algorithms are built on the advances made in the area of association rule discovery. This is a natural step since association rule discovery usually performs an exhaustive search of the search space considering restrictions on properties such as support and confidence and leads naturally to rules and local patterns. For that reason the search strategies of those algorithms are discussed in this section.

Regarding the algorithm in Figure 5, one can consider the search for frequent patterns as an implementation of the GENERATECANDIDATEPATTERNS function. Different research streams in this area have come up with quite different solutions. Ceglar and Roddick's (2006) survey on association rule mining algorithms identifies two main classes of classical of algorithms. First, candidate generation algorithms which identify candidate patterns based on previously discovered itemsets and second pattern growth algorithms that mine for good patters in special data structures. A generic algorithm that could integrate the different streams of association rule mining algorithms has not been found in the literature and is out of the scope of this thesis. Therefore, the implementation of the GENERATECANDIDATEPATTERNS function is algorithm specific and discussed in the following sections. Since many association rule discovery algorithms tend to discover a wealth of patterns, the results should be considered candidate patterns which need to be filtered before being presented to the user. For filtering the most interesting patterns,

coverage approaches have been suggested (Lavrač et. al, 2002; Gamberger and Lavrač, 2000) and will be discussed in chapter 6. These approaches are nearly identical with the coverage which has been applied by algorithms such as CN2-SD or MESDIF. Therefore the search for all possible patterns using association rule mining tools can be regarded as a preprocessing step prior to a coverage approach is then used to evaluate and rank those patterns that have been discovered. In this case the procedure FINDBESTPATTERNS is changed in that way, that INITIALIZEPATTERN chooses one arbitrary candidate from the set of CandidatePatterns. The SELECTCANDIDATES simply selects all CandidatePatterns as candidates. Since all possible patterns have already been discovered and we are now only choosing the best ones, there is no need to refine any candidate. Therefore the return value of refinement should always be the candidate itself. After that, the evaluation is performed according to the chosen quality function (see 5.2). If there are departures from this approach for any algorithm described below, it is mentioned in the respective sections.

5.1.3.1 Apriori-SD and Apriori search

The first algorithm which employs association rule mining tools, the Apriori-SD (Kavšec et al., 2003) algorithms, is based on the Apriori algorithm (Agrawal & Srikant, 1994). Apriori searches through the space of all possible attribute-value combinations looking for frequent itemsets. The algorithm is based on the idea that one can only find frequent itemsets in those cases where all subsets are frequent. For that reason the Apriori algorithm makes a first pass over the database in order to find all *1*-frequent itemsets. Using those itemsets it generates new large itemsets in the k-th loop by creating all combinations of all *k-1* large itemsets, which is called the apriori-join. Largeness is defined along a user specified minimum support saying that at least a certain percentage $x\%$ of transactions must include those itemsets. Among all frequent itemsets found, the algorithm now tries to discover association rules.

This search strategy has been applied by the adaptation for subgroup discovery, the Apriori-SD algorithm. Since Apriori originally does not deal with

labeled data, Apriori-SD adapts the classifying algorithm Apriori-C (Jovanoski & Lavrač, 2001) for subgroup discovery, which is an Apriori variant developed for classification purposes.

The Apriori-C algorithm is an improved version of the *association classification* introduced by Liu, Hsu and Ma (1998). The technique used is based on the idea of first generating all association rules that satisfy minimum support and minimum confidence criteria and then apply a post processing step to turn those rules into a classifier. In order to avoid generating all possible association rules, only those rules whose right hand side contains only the classification attribute are mined. Those rules are called *class association rules* (CARs). Among all CARs that have been discovered, the best n rules are chosen according to some quality function by applying a coverage approach. The Apriori-SD algorithm changes the coverage style of the Apriori-C by implementing a weighting scheme in the post-processing phase instead of eliminating covered examples, thereby finding the best patterns. In essence the algorithm applies the CN2-SD algorithm but rather than trying to discover new patterns through beam search it only evaluates and compares those patterns that have already been discovered in the Apriori step (see section 6). In Figure 5, the Apriori rule mining step is performed in the GenerateCandidatePatterns function and the rest of the coverage approach is applied as discussed above.

5.1.3.2 SD-MAP and FP-Growth

Another frequent pattern mining algorithm that has been modified for subgroup discovery is the FP-growth algorithm (Han et al 2004). It is the basis for the SD-Map algorithm by Atzmüller and Puppe (2006). The algorithm assumes that there is only one single property of interest which is a single attribute value combination rather than all attribute values of a specific attribute with more than two values. This section first describes how to use the adapted FP-growth algorithm to mine frequent itemsets and later discusses how the SD-Map algorithm can be integrated in the generic algorithm shown in Figure 5.

The main changes that have been introduced are the way examples are counted when creating the FP-tree and the possibility to use the FP structure to efficiently deal with missing values (see 4.3). As described in section 4.2.1 the original counting of frequency of attribute values has been replaced by counting the frequency of positive and negative examples for every minimum frequent attribute-value combination. Since for the discovery of frequent itemsets it the absolute number of examples that pass a specific node is still required, this value can be computed simply as the sum of all positive and negative examples counted for that specific node. Splitting the count in the nodes is beneficial since it makes it possible to calculate the quality function during pattern mining already and one avoids searching the original database to get those counts if the original counting would have continued. Also, one can remove the nodes that would otherwise be needed to represent the target value. Next the algorithm is described using a detailed example.

As discussed in section 4.2.1, before applying the FP-growth method, one first needs to create the FP-growth tree. This is done by counting the frequency of all attribute values and sorting them in descending order in a first scan of the database. Those values that do not exceed a specific threshold for a minimum frequency are ignored. In a second scan over the database, those frequent values which are found in an example are then used to create the FP-Tree. Figure 3 describes the FP-tree that has been created using the example data in

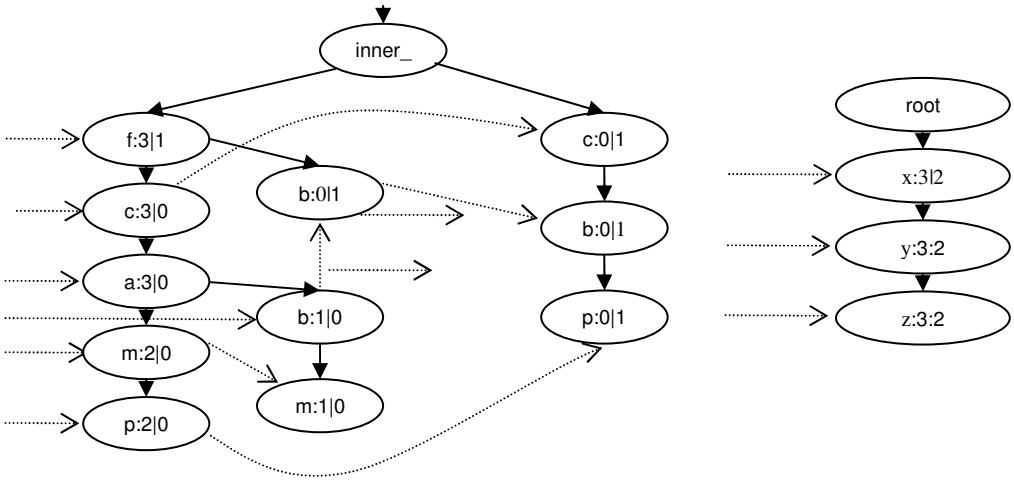


Figure 7.FP-growth multipath tree and single path prefix tree

Table 2. This example describes how to search the FP-tree in order to generate frequent patterns that can later be evaluated in terms of local patterns thereby continuing the example.

It is important to notice, that in the resulting tree all elements have the same linear head (x,y,z) therefore it is useful to remove the head from the rest of the tree and later merge the results of both parts. First, the lower part of the tree, as seen in Figure 7 on the left, is used to explain how one can extract frequent patterns from this data structure. The FP-growth algorithm is a recursive algorithm that builds FP-trees for all attributes in order to discover all frequent patterns.

The extraction of frequent patterns is done by evaluating all attributes in list L=(x:5,y:5,z:5,f:4,c:4,a:3,b:3,m:3:p:3), the sorted list of attribute-value frequencies, in ascending order. Therefore the first value to be considered is p. Since frequent pattern mining is only done with regard to the total number of examples that share an item or an itemset I only state the sum this total number for the different attributes. Starting with a pattern that only includes p itself leads to (p:3=(2|0 and 0|1)) and there are two possible paths through the tree

(f:4,c:3,a:3,m:2,p:2) and (c:1,b:1,p:1). Since in the first path p is found twice, there are two examples that include (f,c,a,m,p) and one that includes (c,b,p). Consequently, there are two prefix paths {(fcam:2), (cb:1)}. These two are p's subpattern base, called conditional pattern base, since they are conditional on p. These two subpatterns are used to build p's conditional FP-tree. Again one needs to count all attributes to find all attributes that occur more often than the minimum frequency. This leads to only (c:3), since c is part of both prefix paths. Therefore the new tree consists only of one branch which results in the only one frequent pattern (cp:2|1). By using this notation one can calculate the quality function of a pattern $c \wedge p \Rightarrow true$, though this does not yet include the linear head. The head could be incorporated in the pattern but since the quality of this extended pattern would be the same, this specialization would probably not lead to better results. This is true only if such a specialization affects all branches of the FP-tree that contain the pattern in question.

Next, attribute m is assessed. Again its single item pattern is (m:3) and there are two paths through the tree that can reach m. ((fcam:2) and (fcabm:1)). Derived from those, m's conditional pattern base is {(fca:2),(fcab:1)}. It is used to build the m's conditional FP-tree. P is not being considered anymore since p has already been considered and all frequent itemsets that include m and p are already known. This leads to (f:3,c:3,a:3) a tree consisting of only one branch. This tree ((f:3,c:3,a:3|m) has to be recursively mined for all frequent patterns. Hereby all items (f), (c), (a) have to be considered individually. Using the order in the tree we start with pattern (am:3), which corresponds to a conditional pattern base of {(fc:3)}. Using this pattern base (f:3,c:3|am), to two patterns, (cam:3) and (fam:3) can be created, with the conditional pattern base of (cam:3) being {(f:3)}, which is mined as (f:3|cam) and reveals pattern (fcam:3), thereby ending this recursion. Now the second pattern, pattern (cm:3) still has to be searched. Its conditional pattern base is {(f:3|cm)}. The search results in pattern (fcm:3). Now the last pattern including f and m leads to only a single pattern (fm:3). As a result, all patterns including m are {(m:3), (am:3), (fm:3), (cam:3), (fam:3), (fcam:3), (cm:3), (fcm:3)}. Here Han et al. note that it is important to

see that when the FP-tree consists only of a single path, frequent itemsets can be discovered by creating all combinations of items in that path.

Continuing with item (b:3) which has three paths {(f:4,b:1),(f:4,c:3,a:3,b:1), (c:1,b1)}. This leads to the following conditional pattern base {(f:1), (fca:1), (c:1)}. Looking at the pattern base reveals that the largest patterns that could be created are (fb:2) and (cb:2) which are both beneath the minimum support of three and therefore the mining of (b:3) stops. Next to be mined is item (a:3). There is only one path (f:4, c:3, a:3) leading to a single conditional pattern base of (fc:3|a). Therefore all patterns found are {(a:3),(ca:3),(fa:3),(fca:3)}. Mining item (c:4) leads to a sub-pattern base of {(f:4)} which leads to a frequent itemset (fc:3). Item (f:4) has no conditional pattern base.

Remembering that in the beginning the original tree had been split, therefore the frequent itemsets that have been mined so far are not yet complete. As we have seen in the previous tree, single path FP-trees can be mined by simply creating all combinations of items. Therefore we first mine the linear path in this way. This gives the following frequent itemsets {(z:5),(y:5),(x:5), (zy:5),(zx.5),(yx:5), (zyx:5)}. Since these are all itemsets that are in the path of all the itemsets mined above, they all need to be combined, meaning that every frequent pattern in the multipath-FP tree needs to be combined with every frequent pattern in the single-path part of the FP tree. The result consequently consists of all frequent itemsets that are in the multipath FP-tree, all itemsets that are in the linear part of the FP-tree and the combination of both. When choosing more general patterns it is most likely that those itemsets would not be incorporated in the patterns since they do not add anything.

As Han et al. (2004) point out, the tree that is created needs not to be minimal but usually reduces the size of the data significantly and increases the speed of the search for frequent patterns.

Originally SD-Map evaluates all discovered patterns according to a user chosen quality function at the time of discovery of the pattern. This has the advantage

of being fast since the occurrence counts are stored in the tree and therefore known for every pattern. Since Atzmüller and Puppe (2006) suggest limiting the number of returned patterns, again one can first mine all patterns without evaluating the goodness and perform the evaluation in the FINDBESTPATTERN function. The maximal number of patterns would then be the STOPPINGCRITERION of the main loop in the FINDLOCALPATTERNS function. Using the algorithm in Figure 5 offers two alternative approaches to discovery of the best k rules. First, as initially discussed by Atzmüller and Puppe (2006), the quality of a pattern could be calculated according to the occurrence counts without making use of a coverage approach. Alternatively one could also discover the best k patterns using weighted coverage. For the first alternative, all potential patterns are stored including information on occurrence counts. The original implementation could be considered using a reweighting function that always keeps the weights at one. In this case the algorithm in Figure 5 would not be efficient since the evaluation of the patterns does not change throughout the runtime and the same order of patterns is rediscovered in every loop, only the first pattern is missing. In this case it would be more efficient to return a list of best patterns, rather than returning a single best pattern. Nevertheless, since no access to the database is necessary and all information for calculating the quality of the patterns will be stored with the patterns, filtering the best k patterns should perform reasonable. When making use of a weighting scheme, the occurrence counts cannot be used, but the approach would be in line with others such as CN2-SD and Apriori-SD. In comparison with the original description, a weighting scheme requires scanning the database to calculate the quality function which takes away part of the advantage of the SD-Map algorithm that minimizes database access. Nevertheless, those costs are comparable that of other algorithms, for instance Apriori-SD. The changed approach is similar to that of Apriori-SD, only the technique of discovering frequent itemsets is changed. This also clarifies that both approaches are quite similar. Next, the Contrast Set mining algorithm Stucco is presented.

5.1.3.3 STUCCO and MAX-Miner

Similar to the SD-Map algorithm which uses the FP-growth algorithm in order to generate candidate itemsets from which the patterns are created, Bay and Pazzani (2004) use an adapted max-miner algorithm for their contrast set mining tool STUCCO. The search space consists of all possible combinations of attributes which are represented in a set-enumeration tree. Before discussing individual adaptations made, the MAX-Miner algorithm is presented first.

The Max-Miner algorithm was developed by Bayardo (1998) with the aim of improving on the shortcomings of the Apriori algorithm concerning the problem of being unable to mine long patterns. According to the author the proposed algorithm for finding frequent itemsets scales nearly linearly with the number of maximal patterns in the data independent of the length of the longest pattern while Apriori scales exponentially with the longest pattern.

The Max-Miner algorithm uses the set-enumeration-tree (SE-Tree) (Rymon, 1992) to create all possible itemsets to subsequently search through. The idea of the SE-tree is to impose an order on a previously unordered set of items or attributes-values that should be enumerated and expand the nodes by combining the values according to the imposed order. An example using 4 values is shown in Figure 8.

In order to efficiently pass through the SE-tree it is important to be able to prune uninteresting parts of the tree. Therefore the items are organized in candidate groups. A candidate group g is composed of a header group $h(g)$ which is the itemset enumerated by the current node and a tail group $t(g)$ which consists of all nodes not in the first group. For node 1 this would look like the following. $h(g)=\{1\}$ and $t(g)=\{2,3,4\}$. This can now be used to find boundaries for the support of frequent itemsets. These are calculated by calculating

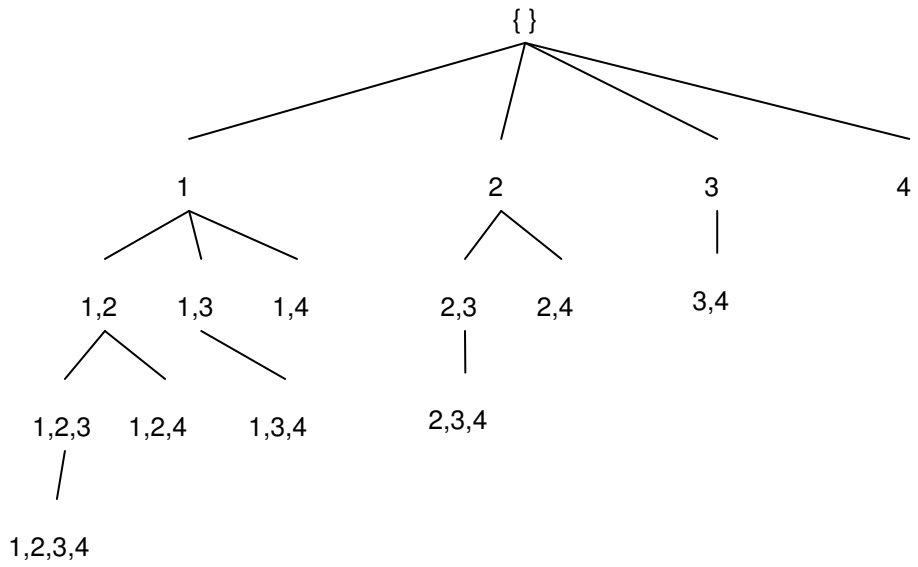


Figure 8.Set enumeration tree for attributes {1,2,3,4}[1]

$h(g) \cup i \forall i \in t(g)$ and $h(g) \cup t(g)$. If $h(g) \cup t(g)$ is a frequent itemset it is automatically the maximally frequent itemset of that branch and therefore there is no need to continue searching in the part of the tree. This is called superset-frequency pruning. Another instance of superset-frequency pruning is pruning a candidate group g if $h(g) \cup t(g)$ is a subset of an already known frequent itemset. If for any item i, $h(g) \cup i$ is not frequent, all nodes in the sub-tree including this item will be infrequent and thus need not be explored further. Therefore before expanding the node this item has to be removed from the tail set. This is called subset-infrequency pruning.

As has been said the items need to be ordered, Bayardo suggests sorting them in ascending order according to support. Those items that are most frequent should be last in line and will therefore be part of most heads or tails when

[1] *cf.. Rymon (1992),Bayardo (1998), Bay & Pazzani (2001)*

trying to find frequent itemsets. This is important since those items are more likely to be in the frequent patterns. The algorithm uses a breadth-first-search approach to find all frequent itemsets.

In Bay and Pazzani's (2004) implementation for contrast set mining the authors replace the pruning strategies suggested by Bayardo with their own improved pruning techniques using properties of the contrast set mining task itself. They distinguish three types of pruning possibilities. First of those is the so-called effect size pruning. Nodes are pruned if the difference between the highest support for this attribute set from any group j and the lowest support form any other group i is smaller than delta. The second type of pruning is based on statistical significance. Therefore a node will be pruned if the expected cell frequencies in the contingency table are too low. The authors suggest an expected value of three or less as a threshold. Also they suggest pruning a node if the maximum value of χ^2 is below the α_l cutoff value. Last, the authors suggest a way of interest based pruning which prunes specializations of contrast sets which do not differ in their support to the subset. As those specializations usually do not contain any new information they are pruned. Also, those nodes are pruned in which one group has an extensively higher support than all the other groups and this relation remains the same regardless of what attributes are appended.

All nodes that remain after pruning is done are transformed into rules with the rule head being the property of interest. Therefore there is no need for filtering any frequent itemsets to contain only those that actually contain the property of interest as would be the case in Apriori-SD. Discovered rules are then tested for statistical significance (see section 5.2) as a quality measure. The algorithm can be performed according to the algorithm depicted in Figure 5. Since the algorithm does not perform a coverage approach, there is no reweighting of examples after each loop in the FINDLOCALPATTERNS function. Since in the case of Stucco the quality function is not used to rank different contrast sets but to determine statistical significance, the discovery of a significant contrast set is

used as the STOPPINGCRITERION in the FINDBESTPATTERN procedure. The whole approach stops when there can no significant pattern be found among the candidate patterns.

5.1.3.4 Branch and Bound

Different from the other approaches presented above Zimmermann and De Raedt (2005) use a branch and bound version of the search in order to find local patterns. This approach is more similar to the heuristic rule learning algorithms presented earlier since it does not return all possible patterns but only the best n patterns, where n is a user specific number. As opposed to the heuristic searches presented in 5.1.2 this approach does guarantee finding the best patterns.

The key idea of the CG-algorithm is to use a correlation measure for which an upper bound can be calculated. This is possible, since a convex function's maximum value can be found at an extreme point which is found in a corner of the feasible region, the region of all potentially possible solutions. Therefore one needs to calculate the quality value for all corner points of the feasible region in order to find the upper bound. The feasible region can be described as a convex set, which is defined as a set in which all points between any two points a and b in the set are also part of the set. In order to find a good upper bound, with good meaning a low upper bound, it is necessary to use the minimal convex set which bounds the solution space. This minimal set is called the convex hull. The upper bound can be used to prune away known to be uninteresting parts of the search space. It is calculated for every possible rule specialization in order to prune those specializations whose upper bound is beneath a certain threshold which is increased during runtime of the algorithm. The rule with the highest upper bound is further specialized since one hopes to find the best solution where the upper bound is maximal. Next, the question of how to identify and constraint the feasible region is discussed, but first some notation has to be introduced.

In order to easily use occurrence counts for calculating correlation measures, Zimmermann and Bosch (2005) define a stamp point as follows.

Definition 6 (Stamp point): For a given data set E every rule r of the form $b \Rightarrow h_1 \vee ... \vee h_d$ induces a tuple $\langle x, y_1, ... y_d \rangle$ of variables. This tuple is called the stamp point.

In Definition 6 the x is the occurrence count/support of the rule body b while the y_i are the occurrence counts for a rule including rule body b and consequent h_i. In the binary case with only a single outcome in the rule head a stamp point would look like (x,y) with $x \geq y$.

To be able to find the rule with the highest upper bound, all possible specializations have to be generated. These specializations of rule r can be called r'. $S_{act} = \{sp(r')\}$ is defined as the set of all stamp points from all rules that are specializations of r. These stamp points can be written as $sp(r') = \langle x', y_1', ..., y_d' \rangle$. In order to avoid computing S_{act}, one can compute $S_{poss} \supseteq S_{act}$, where S_{poss} is the set of all possible stamp points. This can be used as an upper bound and is calculated for each rule and only the rule with the highest upper bound is explored further by actually generating all specializations. In order to calculate S_{poss} one needs to consider the following constraints:

1. $x \geq x' \geq 0$
2. $\forall i : x \geq y_i \geq 0$
3. $\forall i : y_i \geq y_i' \geq 0$
4. $\forall i : x' - y_i' \geq x - y_i$

These constraints form convex sets. Since S_{poss} is the intersection of those sets, S_{poss} is a convex set itself (Zimmermann and de Raedt, 2005). Looking for extreme values, one only needs to consider the points that form the convex hull, which is the smallest convex set containing all points of the convex set, and this is what is done by the algorithm. In the two-dimensional case, the convex hull is

made up of only four points $\langle x, y\rangle, \langle y, y\rangle, \langle x-y, 0\rangle, \langle 0,0\rangle$. Of those points only $\langle y, y\rangle, \langle x-y, 0\rangle$ have to be considered since $\langle x, y\rangle$ is no different from r and therefore rule r should be chosen. The point $\langle 0,0\rangle$ describes a rule that does not cover any examples and is therefore of no use. Hence the maximum value of the correlation measures of those two points has to be chosen as the upper bound. The work as described here is restricted to a single binary target literal as the rule head though the authors also expand their work to rules with multiple outcomes. The key problem here is efficiently computing the upper bound. This is described with regard to the suggested quality function category utility in the following chapter.

As mentioned above, the branch and bound approach is similar to rule learning. It can even be implemented using the algorithm in Figure 5 with only few changes. The search would start with BestPattern being the most general pattern and SELECTCANDIDATES would always choose the best pattern. This candidate would be refined and an evaluation of the refinement would be calculated representing the upper bound for all refinements. There is one change that has to be applied though. The values calculated for the refinements are pattern evaluations that represent the upper bound of this or a specialization of this pattern. When testing if the new pattern is better than the current best pattern, one cannot use this value. Rather than testing this for every refinement, here the actual evaluation for the candidate, from which the refinements were generated, needs to be calculated and compared to the best pattern so far. This value will then be the lower bound. The FILTERPATTERNS process can then filter all patterns whose upper bound is below this lower bound.

A branch and bound approach has also been applied by Suzuki (2004) for the exception rule discovery task MEPRO. The search goes depth-first through a tree which consists of all rule pairs. In each level of the tree, there is a literal added to either the strong rule or the exception rule. The upper bound is then calculated according to the quality measure ACEP (see 5.2). Similar to the

description above, the best k rule-pairs that have been discovered are stored and the k-th rule pair is the lower bound. Every branch of the tree that does not manage to reach at least that lower bound can be pruned. Therefore the algorithm can efficiently search the tree and time efficiently present solutions.

5.2 Quality Functions

In order to be able to find rules and judge whether one is better than another one a quality functions has to be chosen. This is true in the case of the rule learner based algorithms which use heuristic searches as well as for those algorithms applying exhaustive searches. The literature has discussed a wealth of different quality functions (e.g. Fürnkranz, 1999, Fürnkranz and Flach, 2005). This section examines some of those quality functions that have been discussed with regard to local pattern discovery. Considering notation, I chose one in which p means all covered positive examples, n means all covered negative examples, P means all positive examples in the distribution and N refers to all negative examples in the distribution. This does imply that there is only a binary classification which is not necessarily a restriction to the applicability of the quality function. Nevertheless, I feel it helps in better understanding the quality functions. Often a measure does not count the number of examples but the sum of example weights. In those cases, formulas have been added with a " ' " to indicate that for instance p' does not mean the number of covered positives but the sum of weights of all covered positive examples. The quality functions are presented here with regard to the algorithms that use them. Therefore first, the quality function for the STUCCO algorithm for contrast set mining is shown, then the functions for CN2-SD, algorithm-SD, CU-algorithm, MESDIF and Exception Rule learning with MEPRO are shown. This is followed by some quality functions that have been discussed in the literature with regard to local pattern discovery,

Contrast Set Mining - χ^2

$$\chi^2 = \sum_{i=1}^{r}\sum_{j=1}^{c}\frac{(O_{ij} - E_{ij})^2}{E_{ij}}$$

The χ^2 test compares the discovered distribution of examples in the different contrast sets with the expected distribution according to the total set of examples. Usually, this value is compared to a value from the χ^2 distribution table in order to find out whether one can assume the null hypothesis, for instance that both distributions are the same is true.

In the χ^2 formula above, the O_{ij} represents the observed frequencies in row i and column j, while the E_{ij} represent the expected frequencies for that cell. The expected frequencies are calculated according to the null hypothesis that there is no difference between the different groups, therefore Therefore it is calculated as

$$E_{ij} = \sum_j O_{ij} \sum_i O_{ij} / N$$

with N being the total number of examples. This is compared to a χ^2 statistic with $(r-1)*(c-1)$ degrees of freedom.

In terms of contrast set mining the hypothesis tested using the χ^2 test is depicted below.

$$\exists ij\ P(contrastSet = true \mid G_i) \neq P(contrastSet = true \mid G_j) \quad (1.)$$

$$\max_{ij} \left| \text{support}(\text{contrastSet}, G_i) - \text{support}(\text{contrastSet}, G_j) \right| \geq \delta \quad (2.)$$

	Set 1	Set 2	Set 3	Set 4	Total
Positive	130	12	412	50	604
Negative	1102	350	2135	320	3907
Totals	1232	362	2547	370	4511

Table 5. Example Table of hypothetical contrast set data

The first property (1) is the null hypothesis that the probability of a discovered contrast set between two groups is significantly different. This is checked using the χ^2-measure. The second property (2) is simply a test on whether the support difference is greater than the user defined threshold. Therefore, the quality function chosen by Bay and Pazzani (2001) defines the quality of the found contrast set as a combination of the χ^2 function and the support difference. One important choice when doing statistical testing is the choosing a level of significance according to the users needs. Typically, a level of $\alpha = .95$ is chosen This means there is a 5% chance of type 1 error, meaning that one would erroneously reject a null hypothesis although it is actually true. The problem with choosing a significance level is that in data mining, there are hundreds or even thousands of hypothesizes tested. Even though the chance of type 1 error is still only 5% for each test, in absolute terms there may be many cases in which for instance contrast sets are evaluated to be statistically significant, though in reality they just occurred by chance. In order to avoid that problem, Bay and Pazzani (2001) employ the Bonferroni inequality. The Bonferroni inequality states that the propability of the union of a set of events is less or equal the sum of the individual probabilities. Events in the context of statistical testing are test with a certain user selected significance level. In a series of tests the Bonferroni inequality states that the type 1 error of that test

series is equal or less than the sum of the individual significance levels. Using the inequality one can guarantees the that the type 1 error of several test remains below a threshold α if the sum of significance levels of all tests is equal to or below α. Since it cannot be said for sure how many test will have to be performed, the authors suggest setting α to

$$\alpha_l = \min(\frac{\frac{\alpha}{2^l}}{|C_l|}, \alpha_{l-1})$$

where i is the current depth of the search tree and $|C_l|$ is he number of candidates at level i. Obviously those strict cut-offs lead to an increase of type 2 errors, which means that significant contrast sets are wrongfully rejected. The authors nevertheless still find it helps in only discovering the most important and consequently most interesting contrast sets.

Statistical tests are also used by others in the realm of subgroup discovery. For instance Klösgen and May (2002b) employ the binomial test in their subgroup discovery system SubgroupMiner:

$$\frac{\frac{p}{p+n} - \frac{P}{P+N}}{\sqrt{\frac{P}{P+N}(1-\frac{P}{P+N})}} \sqrt{n} \sqrt{\frac{|P+N|}{|P+N|-(p+n)}}$$

With $p+n$ representing the size of the subgroup, $\frac{p}{p+n}$ represents the share of the target group in the subgroup and $\frac{P}{P+N}$ represents the share of the target group in the total population.

CN2-SD, Apriori-SD - **WRAcc**

$$WRAcc(r) = (\frac{p'+n'}{P'+N'}) \cdot (\frac{p'}{p'+n'} - \frac{P'}{P'+N'})$$

Weighted Relative Accuracy trades off the generality of the rule and the relative accuracy compared to the default accuracy. The measure is therefore using the insight that a rule can only be interesting if its distribution deviates considerably from the prior distribution of the class. To avoid too specific rules, generality $(\frac{p'+n'}{P'+N'})$ is used as a weight in order to find considerably general rules. One important issue is the weighting scheme used with this measure. This is further described in section 5.3.

Wrobel (1997) uses a variation of the WRAcc measure for the MIDOS algorithm. For evaluating found subgroups, the algorithm employs a variation of the WRAcc heuristic:

$$q(r) = \begin{cases} 0 & if \ \frac{p'+n'}{P'+N'} < s_{\min} \\ (\frac{p'+n'}{P'+N'}) \cdot (\frac{p'}{p'+n'} - \frac{P'}{P'+N'}) & otherwise \end{cases}$$

The only difference is that Wrobel additionally demands a minimum support. In those cases where a rule covers less than the minimum percentage of all examples, the rule is automatically evaluated to zero.

Accuracy itself has only been used in early subgroup discovery systems. It has been suggested by Klösgen (1996) in the context of subgroup discovery as one of several possible rule quality functions.

$$Accuracy(r) = \frac{p + (N - n)}{P + N}$$

Accuracy measures a single rule and compares all correctly classified examples to the total number of examples. As Fürnkranz (1999) points out the measure is equivalent to maximizing $p - n$ since P and N are equal for all rules.

Algorithm-SD

Gamberger and Lavrač (2002) have suggested two possible quality measures for algorithm-SD. The aim of the Algorithm-SD is to be able to perform subgroup discovery which uses expert knowledge. Therefore their suggestions for quality functions q aim at making it possible for the expert to influence the search. The suggested functions include a user settable parameter g which can be used to find different rules according to the user preferences. First there is the

q_g quality measure

$$q_g = \frac{p}{n+g}$$

q_g trades off covered positive and negative examples with a user settable factor g. Generally a larger g means a more general rule, consisting of more falsely covered examples, is acceptable. A smaller g would mean a higher accuracy, since covering negative examples is made relatively expensive. The parameters allow guiding the search towards more specific rules, using smaller values for g, the authors recommend a value between 0 and 1. In the same way, more general rules can be found employing a larger value of g, the authors suggest a value larger than 10, will present more general rules. A value of 1 will weight each covered TP equal to one covered FP. Secondly the authors also suggested the measure

q_c quality measure

$$q_c = p - c * n$$

Gamberger and Lavrač (2002) introduce the parameter c in order to be able to include a cost factor that allows the user to specify the number of additional covered positive examples for every new false positive. It serves the same cause as the factor g in the q_g –measure. This quite intuitive measure is equal in its results to measure q_g when an exhaustive search of the TP/FP Space is

performed. Yet, the authors argue q_c will prefer more specific rules in a heuristic setting as proposed with algorithm SD. Additionally, the authors argue that general rules will be cut of by the q_c measure due to its constant slope of equal rules. For that reason the authors encourage the use of the q_g measure.

Besides those two measures, Algo-SD also requires that all discovered patterns satisfy a user specified minimum support criterion. Using such a constraint makes sure that discovered patterns cover a significant part of the examples.

CU-Algorithm - Category Utility

$$CU(C_1, C_2) = \frac{1}{2} \sum_{C \in \{C_1, C_2\}} P(C) \sum_{A \in \mathrm{A}} \sum_{v \in V[A]} P(A = v \mid C)^2 - P(A = v)^2$$

Zimmermann and de Raedt (2005) suggest the use of the category utility. The measure had originally been applied to clustering algorithms and programs (Mirkin, 2001; Fisher, 1987). It measures the "goodness" of a category. In the term above, the Cs represent the different clusters. It compares the probability of a feature taking on a specific value with regard to a cluster and compares it to the a priori probability of this feature value without knowledge of the cluster. The idea of the measure for usage in the CU-algorithm is to consider all elements covered by the rule body as part of the first cluster and all those that are not covered as part of the second one. The only attributes which are of interest here are the binary target attributes that are used for classification. Therefore Zimmermann and de Raedt reformulate category utility as

$$CU(b_r, \mathrm{A}_t) = \sum_{A \in \mathrm{A}_t} \sum_{v \in \{true, false\}} \sum_{b \in \{b_r, \neg b_r\}} \left(\frac{1}{2} P(b)(P(A = v \mid b)^2 - P(A = v)^2)\right)$$

It can now be described as the sum of partial category utilities.

$$CU(b_r, \mathrm{A}_t) = \sum_{A \in \mathrm{A}_t} CU(b_r, \{A\})$$

This can be put in the p-n notation as follows.

$$\sum_{A\in \mathrm{A}}\left[\begin{array}{l}\left(\frac{1}{2}\frac{p+n}{P+N}\left(\frac{p}{p+n}\right)^2-\left(\frac{p}{P+N}\right)^2\right)+\left(\frac{1}{2}\frac{P+N-(p+n)}{P+N}\left(\frac{p}{P+N-(p+n)}\right)^2-\left(\frac{p}{P+N}\right)^2\right)\\ +\left(\frac{1}{2}\frac{p+n}{P+N}\left(\frac{n}{p+n}\right)^2-\left(\frac{n}{P+N}\right)^2\right)+\left(\frac{1}{2}\frac{P+N-(p+n)}{P+N}\left(\frac{n}{P+N-(p+n)}\right)^2-\left(\frac{n}{P+N}\right)^2\right)\end{array}\right]$$

All p and n have to be considered with regard to the attribute in question but this description is not as concise as one would wish for, therefore indices have been omitted.

For Category Utility the authors suggest calculating the upper bound as follows

$$\max_{0<x'<x}\left\{\sum_{A\in \mathrm{A}}\max_{p'_i\in\{p^A_{\min},p^A_{\max}\}}CU(x',p'_i)\right\}$$

It iterates over all possible values of x' which represent the number of covered examples for a fictitious specialization and searches for the maximum value for all category utility values. The summation goes over all possible outcomes, meaning all possible rule heads. The maximum and minimum values for p^A are determined by the value of x' since $p^A \le x'$ and obviously p^A cannot be larger than the total amount of covered examples of this class. This means for calculation of the upper bound only $2d(x-2)$ calculations have to be performed.

MESDIF

Confidence (Precision)

$$Conf(b \Rightarrow h) = \frac{p}{p+n}$$

Confidence, also known as precision in rule learning, measures the number of true positive examples covered among all covered examples. It actually is not only important in terms for the MESDIF algorithm, but Apriori style algorithms

use confidence to evaluate association rules. In that case, the measure is interpreted as a percentage of examples covered by a certain rule over all examples, since originally in the market basket context there is no classification. For the MESDIF algorithm, Berlanga et al. (2006) suggest the use of an adapted version of the confidence measure.

$$Conf(b \Rightarrow h) = \frac{APC(p)}{APC(p) + APC(n)}$$

APC is the Antecedent Part Compatibility and can be considered a weighting function. It measures the degree of membership of an example. Since here fuzzy rules are applied, an example that is covered by the rule body needs not necessarily be covered completely since it can only be covered to some degree depending on the fuzzy value. Therefore, confidence is calculated as the sum of the compatibility degree between examples that fulfill the rule divided by those that only fulfill the rule body. This quality measure is only one part of a multiobjective optimization algorithm that also tries to maximize original support as well as a measure the authors call support.

Original Support

$$Sup(r) = \frac{p'}{P'+N'}$$

Support measures the number of positive examples covered by a rule compared to the total number of examples. Berlanga et al. (2006) use the measure to compare the level of interest of a rule compared to all other rules. They do this by introducing example weights which are decreased for each rule that covers that particular example, therefore making sure that other more interesting rules are found without loosing the example altogether. The weights of each example are calculated as $\frac{1}{k}$ with k being the number of times the example has been covered already if it has been covered at least once.

Furthermore the authors suggest their own definition of support as a third quality measure.

Support

$$\sup 1(r) = \frac{p}{P}$$

This measure is used in order to compare the interestingness of the rule in terms of how many examples of the class are covered with regard to the total number of examples of that class. The aim of this measure is to endorse diversity of the discovered rules. For calculating this measure, the number of examples is used rather than the weights. If the example weights where to be used, support and original support would be equivalent.

Exception Rule learning with MEPRO

J-Measure

$$J(p;n;P;N) = \frac{p+n}{P+N}\, j(p;n;P;N)$$

$$where\ j(h;b) = \frac{p}{P+N} * \log_2 \frac{\frac{p}{P+N}}{\frac{P}{P+N}} + \frac{n}{P+N} * \log_2 \frac{\frac{n}{P+N}}{\frac{N}{P+N}}$$

The quality function J measures the quantity of information which is compressed by a rule. It has originally been suggested as a quality measure for the ITRULE rule learning algorithm (Smyth, P. and Goodman, 1992) for a rule $b \rightarrow h$. In order to be applicable in the context of rule pair exception rule discovery, Suzuki suggests the use of an adapted version of this measure called ACEP

ACEP

$$ACEP(h,b_\mu,h',b_\nu) \equiv J(h;b_\mu)J(h';b_\mu \wedge b_\nu)\ .$$

The ACEP measure is a specialized measure developed for use with exception rule pairs. The h represents the class of the strong rule b_μ and h' represents the class of the exception rule $h_\mu \wedge h_\nu$. In this case, the p and n values are counted with regard to the classification defined in the rule heads h and h'. It is a multiplicative version of the J-measure which includes the values for the strong rule as well as the exception rule. By using a multiplicative form, the authors avoid domination of high values as would be the case in an additive conjunction.

SD-Map - q_{RG}-measure

$$q_{RG} = \frac{\frac{p}{p+n} - \frac{P}{P+N}}{\frac{P}{P+N} \cdot (1 - \frac{P}{P+N})}$$

In their application Atzmüller and Puppe (2006) do not encourage the use of a particular quality function since the key issue for the algorithm is efficient discovery of good patterns. Nevertheless a postprocessing step is suggested to only present the best k patterns to the user. Therefore the authors suggest the q_{RG}-measure as one possibility to evaluate rules. The measure compares the deviation of the relative frequency of the target in the subgroup with the relative frequency of the target variable in the whole population. The divisor is a normalizing factor which should be the same for all rules considering there are no missing values. In case of missing values the numbers for the whole population can be calculated according to the description in 4.3. The q_{RG}-measure is part of the WRAcc though here it does not encourage generality of discovered rules. Therefore one could as well use WRAcc in a way as discussed in section 6.

5.3 Weighting

As discussed earlier, using examples weighting has been employed in CN2-SD as well as the MESDIF algorithm. Using weights for the examples, the authors

mitigate the effect of eliminating covered examples that leads to skewed example subsets. It also helps finding better rules that do in fact perform mildly worse than the first found rule, but still would do significantly better than rules found after removal of those examples and therefore can be considered interesting from a pattern discovery point of view, though less so from a classification point of view. Thus, it is important to evaluate what different possibilities for weighting examples there are.

Lavrač et al. (2004a) suggest two different ways of example weighting. A multiplicative weighting scheme that uses a parameter $0 < \gamma < 1$ to compute the weight $w(e_j, i) = \gamma^i$, with $w(e_j, i)$ being the weight of example e_j which has been covered by i rules. Alternatively they suggest the use of an additive weight, which is computed as $w(e_j, i) = \frac{1}{1+i}$. According to their experiments, Lavrač et al. (2004a) suggest using the additive weighting scheme since on average it performed better than the multiplicative scheme and is easier to calculate. The additive weight is also suggested by Berlanga et al. (2006) in the MESDIF algorithm, yet, since it does weight examples only after they have been covered at least once, the authors do without adding one to the denominator. According to Lavrač et al., initially all examples should be weighted with a value of 1 which then decreases with each applicable rule. As discussed above, the weighted relative accuracy measure was adapted using the example weights rather than just counting examples.

An alternative weighting scheme has been developed by Scholz (2005). The key motivation for this enhanced scheme is the question of how to make sure that the algorithm does not only find already known rules, but rather uncorrelated, new rules. This also gives opportunities to incorporate existing knowledge in the search and thereby helps to avoid rediscovering it thereby leading to more useful discovered knowledge. Prior knowledge could be in the form of patterns that were discovered during the runtime of the algorithm or manually discovered patterns. The author essentially develops the idea of

sample weights further by applying a technique which is similar to boosting techniques. The aim of the weighting scheme is to remove discovered knowledge from the distribution without altering the underlying original distribution too much.

The key idea is to produce a new distribution from the old distribution in which the rule r, which represents the prior knowledge, is no longer supported. This means that in the new distribution D' the probability of finding a positive example among all examples is the same as finding a positive example among those for which rule h is true. Therefore the rule body b and positive examples Y_+ are independent.

$$P_{D'}(Y_+ \mid b) = P_{D'}(Y_+) \quad 3.$$

In order to avoid a distortion of the new distribution D' beyond the original distribution and thereby destroying knowledge contained in the original distribution or even falsely creating new correlations the author describes six constraints that must hold.

$$P_{D'}(b) = P_D(b) \quad 4.$$

$$P_{D'}(Y_+) = P_D(Y_+) \quad 5.$$

$$P_{D'}(x_+ \mid b \cap Y_+) = P_D(x_+ \mid b \cap Y_+) \quad 6.$$

$$P_{D'}(x_+ \mid b \cap Y_-) = P_D(x_+ \mid b \cap Y_-) \quad 7.$$

$$P_{D'}(x_+ \mid \bar{b} \cap Y_+) = P_D(x_+ \mid \bar{b} \cap Y_+) \quad 8.$$

$$P_{D'}(x_+ \mid \bar{b} \cap Y_-) = P_D(x_+ \mid \bar{b} \cap Y_-) \quad 9.$$

Constraints 4 and 5 imply the prohibition of changes to the probability of events that are not influenced by the rule, meaning that the probability of a random draw from the new distribution should not be different. The example space can

be partitioned in four groups. Every example x belongs to either one group. The first partition includes those examples which are covered by rule r and are positive, the second partition includes examples where rule r is applicable and x is a negative while partitions three and four are those examples that are not covered by r and are either positive or negative. Constraints 6-9 imply that the new partition is defined proportionally to the original partition. This means that if the probability of seeing on specific example halves, the probability of those examples which either share the property of interest or not and those that are of the same class or not will also halve. Scholz (2005) argues that if that was not done, one would without reason prefer some examples over some others in the new distribution.

The weights are computed as discussed in the following. Rather than using weights of 1 for all examples at the beginning, example weights are chosen so that class priors are equal for all classes. This is done as it is proven by Scholz (2005) that in such a stratified sample the subgroup discovery task using the WRAcc utility function can as well be solved using a standard rule learning algorithm with accuracy as its utility function. Therefore a generic rule learner can be used to find appropriate rules. After a rule has been discovered the examples have to be reweighted with regard to constraints 6-9. This is done by multiplication of the example weights of all examples x with

$$Lift(h \to Y_+, x) = \begin{cases} Lift(b \to Y_+) \; for \; x \in b \cap Y_+ \\ Lift(b \to Y_+) \; for \; x \in b \cap Y_- \\ Lift(\bar{b} \to Y_+) \; for \; x \in \bar{b} \cap Y_+ \\ Lift(\bar{b} \to Y_+) \; for \; x \in \bar{b} \cap Y_- \end{cases}$$

Lift is calculated as

$$Lift(A \to B) = \frac{P(A \cap B)}{P(B)} = \frac{P(B \mid A)}{P(B)}$$

It measures the relative frequency of an outcome given a specific rule relative to the general frequency of that outcome. As shown by Scholz, the recalculation of

the example weights as described above ensures the applicability of the constraints as defined above.

The significance of the work is that it incorporates prior knowledge, usually in the form of already found rules. It thereby limits the amount of rules found and avoids rules that are highly correlated and therefore cannot be considered interesting. Consequently, rules found are described as being diverse and therefore potentially more useful than with other weighting schemes. The weighting can either be put in place through example weights or resampling, making the algorithm applicable even to algorithms that are difficult to work with example weights.

5.4 Conclusion

This section discussed how the search for good patterns is organized. It discussed the different approaches which are based on either a heuristic or an exhaustive search in the space of possible local patterns. Table 6 summarizes the search type and the applied quality functions for all algorithms.

Sections 5.1.2 and 5.1.3 have shown that heuristic as well as exhaustive searches can be implemented based on the same generic algorithm based on rule learning. The reason for that is that is that the frequent pattern mining algorithms typically are used to generate candidate local patterns only. Those candidate patterns are then evaluated to find the k best patterns which are presented to the user. The k best patterns can be discovered using a coverage approach. In the case of frequent pattern mining algorithms, this is discussed as post-processing. Since this approach is basically the same approach as in heuristic local pattern discovery, both forms can be incorporated in a single generic rule discovery algorithm. One can consider the use of frequent pattern mining algorithms as an early filter for what can be considered potentially good rules. In the context of association rule mining based algorithms, the applied quality function can have a different meaning than it has for the other algorithms. Here a quality function, such as χ^2 or support, defines only the

minimum requirement a pattern has to satisfy. They are not used for comparing different patterns. For that, another quality function has to be applied in order to discover the best patterns, unless all discovered pattern should be presented.

Unlike exhaustive algorithms, CN2-SD, Algo-SD, MESDIF, CU-Algo and MEPRO only generate a predefined number of patterns. In its original version, CU-Algorithm tries to discover only the single best pattern, though that can be changed to either discover the best k pattern during the first search or to discover the best pattern remove or reweight covered examples and repeat the process k times.

In terms of quality functions, the survey has shown that quite different quality function have been applied for supervised local pattern discovery. Lavrac et al. (2002) explicitly suggest the use of WRAcc as a measure specifically for subgroup discovery, though this has not become a defining element of subgroup discovery. It has been applied for CN2-SD as well as Apriori-SD. In order to justify subgroup discovery and other approaches as an independent field of research, quality functions should be used that distinguish the field from inductive rule learning, since the goal is not predictive accuracy but interesting descriptions of the data. This has been achieved so far through application of weighting schemes in combination with WRAcc as well as combinations of

	Search Strategy	Search Type	Quality Function	Problem Definition	Weighting
CN2-SD	Heuristic	Beam Search (CN2)	WRAcc	Subgroup Discovery	Yes
Algo-SD	Heuristic	Beam Search	q_g, q_c	Subgroup Discovery	No
MESDIF	Heuristic	Genetic Search	Confidence, original Support, Support	Subgroup Discovery	Partially
Apriori-SD	Exhaustive	Apriori Search	WRAcc	Subgroup Discovery	Yes
STUCCO	Exhaustive	Max-Miner	χ^2	Contrast Set Mining	No
CU-Algo	Exhaustive	Branch and Bound	Category Utility	Cluster Grouping	No
SD-MAP	Exhaustive	FP-Tree	not specified	Subgroup Discovery	(No)
MEPRO	Exhaustive	Branch and Bound	ACEP	Exception Rule Mining	No

Table 6. Summary of algorithms for local pattern discovery

different quality functions or by incorporating cost factors so user specific settings are possible.

Table 6 summarizes the main properties of the algorithms that were presented in this chapter. One can see that most algorithms that have been presented here have been developed in the realm of subgroup discovery. The second column shows the search type that is applied. This refers to the algorithms on which the local pattern mining algorithms are based. In the case of SD-Map, the no weighting scheme has been set in brackets since there is no explicit quality function suggested by the authors. Since one could use the results and filter them though application of the WRAcc function, it is possible to use a weighting scheme with this algorithm, as has been suggested above. In the case of the MESDIF algorithm, example weights are used only for the original support function. All other functions work with weights fixed to one.

As discussed above, post processing is often necessary in order to avoid presenting too many and often uninteresting patterns. The following chapter presents how post processing is done in most algorithms.

6 Post processing

The post processing phase is mostly concerned with which rules should be presented to the user. This is a problem for those algorithms that perform an exhaustive search of the pattern space, since this type of search is most likely to generate a wealth of rules that cannot or should not be presented to the user. In the previous chapter, this step has been integrated in the core process of the algorithm in Figure 5. Despite this, the issue should be discussed in this chapter a bit more, since also alternative approaches can be used to implement post processing. Since the goal of post processing is filtering all but the most interesting patterns, first the term interesting is briefly discussed.

Generally it is difficult to discuss interestingness since interestingness can be seen as a subjective measure as well as an objective measure. Klösgen (1996) describes subjective interestingness measures such as usefulness which relates the knowledge found by a pattern to the objectives of the user. Unexpectedness (Silberschatz & Tuzhilin, 1995) means that it is surprising to the user, which means it should take into account background knowledge of the user. Actionability (Silberschatz & Tuzhilin, 1995) is another such subjective measure, which means that a user should be able to use the discovered knowledge in some way. Since all those measures of interestingness somehow relate to the specific user and his or her knowledge, it is hardly possible for an algorithm to derive such knowledge. Nevertheless, there are ways to present what can be assumed to be interesting patterns, for instance on a basis of what has already been presented, thereby reducing the total number of patterns shown to the user.

Such an approach has been suggested for STUCCO (Bay and Pazzani, 2002). They suggest two different ways to post process data and reduce the number of patterns presented to the user. The first of those is statistical surprise. This means the user is shown the most general contrast sets first and only later will be shown more complicated contrast sets if they are surprising with regard to

what has been shown previously. To do that they first use a log-linear model of expected values given knowledge of presented rules and only present rules that deviate from those expected values. Second, they suggest looking for linear relations in cases where there groups can be seen as time series data, since those relations might be easily understandable and might therefore be interesting. An idea similar to the first one has been suggested by Knobbe and Ho (2006). Here the authors describe measures which are used to build what they call pattern teams. Pattern teams are sets of patterns aim at fulfilling six criteria. Those criteria include that patterns should not overlap strongly or complement each other strongly, new patterns should not be added to the set if they approximately cover only a linear combination of examples already in the set, patterns should be mutually exclusive and when using patterns as a classifier they should be the best classifier possible and finally, patterns teamed together should be on the convex hull of the ROC space of all patterns.

Rather than filtering surprising patterns by statistical surprise or linear relationships, an obvious constraint is to present only a fixed number n of patterns that have been found and make sure only the best patterns according to some criterion are chosen. This can be done during the runtime of the algorithm or the algorithm can search all valid patterns and then post process those results. This approach is used by Apriori-SD algorithm. Here, the authors follow Lavrač et. al (2002) which generalizes the approach of subgroup discovery through rule induction based on weighted coverage which has been implemented in of CN2-SD. To measure rule quality, WRAcc is used. Rather than removing covered examples completely they merely decrease example weights according to the additive reweighing strategy. This also means that possible improvements through alternative weighting schemes as suggested by Scholz (see 5.3) could also improve the performance of Apriori-SD. Similarly for algorithm-SD, a post processing step was suggested to minimize the number of presented rules by first choosing the maximal number n of rules that should be shown to the user and then choosing as diverse such n rules as possible, though the suggested algorithm does not guarantee statistical independence.

The approach was first introduced by Gamberger and Lavrač (2000). They propose choosing the rule that covers most examples as a first rule. Examples are weighted so that each covered example's weight is calculated as an additive weight. Using this approach, diverse rules are selected since similar rules acquire a less favorable quality value. The idea is quite similar to the idea presented earlier only in this case the evaluation function is simply the sum of all covered example weights rather than WRAcc. This is the same approach that has been integrated into the algorithm depicted in Figure 5.

This chapter completes the presentation of the different algorithms. It shows ways of dealing with the problem of too many discovered patterns. Essentially, algorithms like Algo-SD and Apriori-SD perform a weighted coverage approach to evaluate rules. SD-Map does not specifically suggest how to perform this task but also suggests comparing discovered patterns and only presenting the most interesting ones. This could be done either directly when mining for interesting patterns, though this does not make it possible to remove or reweight covered examples. Therefore a post processing step similar to the mentioned above has been suggested in chapter 6. The coverage approach which has been discussed here is similar to the approach chosen by heuristic search algorithms like CN2-SD, the key difference is that there are no refinements needed since all valid patterns are known beforehand.

7 Related topics in supervised local pattern detection

This chapter reviews two related topics that have not been discussed previously. First, there is the issue of searching for local patterns in relational databases and second there has been a specific application for contrast set mining in time series data. Both applications are discussed in the following two sections.

7.1 Subgroup Discovery in Relational Databases

So far all algorithms that have been discussed have used a single relation in order to find local patterns. However, there is the issue of finding interesting local patterns in relational databases. For this problem, several algorithms have been developed (Železný and Lavrač, 2006; Lavrač, Železný and Flach, 2002; Wrobel, 1997; Klösgen and May, 2002a,b). In the following, I briefly describe those algorithms.

RSD (Železný and Lavrač, 2006; Lavrač, Železný and Flach, 2002) works by using first order feature construction from which it creates a single relation. On this relation RSD employs the CN2-SD subgroup discovery rule learning algorithm which produces propositional rules. For feature construction RSD uses a Prolog program as background knowledge. This is done by only using the background knowledge independently of the actual data. Next the features are instantiated using the most common values. Then the features are used to reproduce the single relation that is needed in order to be able to apply the CN2-SD algorithm. Nevertheless, it is essentially only an extension of the CN2-SD algorithm but does not add in terms of expressivity, since first-order logic is only applied for feature construction while the approach itself is again purely propositional.

While RSD only produces a single relation of constructed features the MIDOS algorithm (Wrobel, 1997) actually works on the multi-relational table. Thus, it describes the hypothesis language space as a propositional hypothesis space

which uses first order predicates in order to represent the different relations of the database and uses conjunctions of those first order literals as the group description.

$$r_0(X,Y,Z) \wedge r_1(Y,U) \wedge r_2(<Z,R) \wedge r_3(U,R)$$

The variables r_0 to r_4 represent the relations and within these literals there are foreign links (Y,Z,U,R) which have to be considered according to the foreign links stored in the data base. Refinement of rules[2] is done according to the foreign links that are found in the relations. These are ordered and guarantee that each path through the tree which consists of relations and foreign links is visited at most once. The search is organised as a general to specific search which can use breadth-first, depth first or best first search algorithms. The search is organised along the graph of database relations with the links being the edges. Due to the optimal refinement operator the algorithm can also search the space of possible subgroups in parallel. The search is aimed at finding the best k subgroup descriptions. To be able to find the best k subgroups one needs to prune uninteresting parts of the search space. Thereby two types of pruning are applied. First, if the quality measure is zero the part of the tree can be pruned since results cannot become any better if more items are added. Second, MIDOS uses optimistic estimate pruning which is based on the fact that the algorithm only looks for the best k subgroups. If the optimistic estimate for the q(h) is less than the worst of the already found solutions, this part of the tree can be pruned as well. The optimistic estimate is calculated as the WRAcc with the $\frac{n'(p)}{N}$ set to one.

[2] See also Hoche and Wrobel (2001)

Klösgen and May (2002a,b) describe the SubgroupMiner system that is used to infer subgroups within spatial data. The similar to MIDOS SubgroupMiner also works directly on the multi-relational table rather than preprocessing the database and forming only a single relation. Hence the hypothesis language consists of concepts that contain attribute value pairs for each relation and links between the different relations within each concept.

7.2 Subgroup Discovery in Time Series Data

Lin and Keogh (2006) expanded the notion of contrast sets to work with time series data. They aim at identifying key patterns that are able to distinguish between two sets T and S of time series data. For doing that they introduce the concept of Time-Series-Diffs (TS-Diff). A TS-Diff(T,S,n) is a subsequence C of time series T of length n that has the largest distance to its closest match in S. They aim at finding the most relevant pattern according to TS-Diff. Their quality measure in order to distinguish different pattern is the Euclidean Distance.

$$Dist(T,S) = \sqrt{\sum_{i=1}^{n}(t_i - s_i)^2}$$

To be comparable, both time series' have to be normalized to have a zero mean and a standard deviation of one. They expand the brute force algorithm of comparing each subsequence of length n in T with each subsequence of length n in S by ordering all subsequences in T and S so that a good first pattern, meaning one with a large nearest-match distance can be found quickly. This enables early stopping of comparisons later in the search since a pattern does not need to be fully searched if its closest match is already closer than the best found so far. In order to be able to produce an ordering which makes such an ordering possible the authors suggest the use of the Symbolic Aggregate ApproXimation (SAX) (Lin et al., 2003). In SAX, they transform time series data in symbolic representation using a limited alphabet of size α and a word size w.

Using the transformed data, they store all α^w possible patterns in a hash table and count how often each pattern has appeared. Using this knowledge they try to organize the outer loop going over data in T in order to find a good first pattern by going through the buckets in S which are empty and finding those buckets in T that are non empty. Those should have relative higher probability of having a large difference to their closest match in S. In order to be able to abort the search early, the first patterns in the inner loop searching through S that are searched should be relatively close to the comparing pattern in T. Therefore when the i-th pattern in T is being evaluated the corresponding SAX word A is created and the pattern is compared first with all those sequences of S that are found in the corresponding hash table bucket of A. These subsequences should be relatively similar to the one we are comparing it to, therefore assuming a relatively great difference has been found early, the search will not have to be continued since the pattern already has found a matching pattern closer to it.

As we can see, the rule learning makes use of the special data structure of time series analysis. The groups are trivially defined by the two time series that are being evaluated. Due to the data structure, the problem has originally a complexity of $O(n^2)$ due to the comparisons. Lin's and Keogh's analysis has shown that in the best case scenario the running time of the algorithm will be only $O(n)$ if the patterns are visited in perfect order. Their heuristic makes it possible to increase the possibility of near-perfect ordering with only a few extra loops over the data. This leads to an improvement in runtime that can be expected, since it is a heuristic approach only this does not change the upper limit of the running time behavior of $O(n^2)$, but still, in practice, it can be expected to be significantly faster.

8 Conclusion

This thesis surveyed the literature on supervised pattern discovery. The different definitions that have been developed in different research communities have been presented in a single paper and discussed. It can be seen that problems that are discussed in the literature with regard to rule learning also apply to most of the algorithms mentioned here. This is hardly surprising considering that all algorithms presented here are modifications of standard rule learning algorithms or association rule learning systems. Generally the survey has shown that almost any rule learning algorithm can be used for supervised local pattern discovery. Typically the difference with regard to standard rule learning is the emphasis on discovering descriptive patterns rather than predictive rules. For that reason, there is less emphasis on predictive accuracy. This can be seen in the way patterns are chosen. For the discovery of local patterns, statistical testing methods, like the χ^2 test or binomial tests, are applicable. This is because the absolute interestingness compared to other patterns is less important than the identification of patterns as such. Since such an approach could lead to quite many discovered patterns a post processing step is needed to filter the best patterns. In this case a quality function is needed that can rank the identified patterns. As discussed in section 5.2 WRAcc is such a function which had been suggested for the discovery of subgroups in data.

Considering the algorithms that have been discussed one of the most important differences has been the type of search applied. This difference is less important than is appears to be. The reason for that is, that pattern discovery algorithms which use association rule discovery often add a post-processing step in which the best k patterns are chosen. This is usually done by implementing a coverage approach which does without the search for new patterns but only evaluates those patterns that have already been discovered and then removes or reweights covered examples. This is quite similar to the

algorithms based on rule learning. Therefore one can consider the association rule search as a special kind of refinement. Rather than refining a singe best rule or a beam of k best rules, refinement is done by deliberately choosing the best rule in a limited set of pre-discovered rules. For that reason, it was possible to integrate the association rule learning based discovery algorithms in the generic rule learning algorithms as presented in chapter 5.

Appendix A: Basic Statistics

This appendix describes some important basic statistical terms, axioms and theorems. It follows the definitions and presentation of Mittelhammer (1999)

Probabilities

Probabilities are used to quantify the level of certainty or uncertainty associated with observations made in a situation whose outcome is determined by chance. Some important terms in the context of probability theory are experiment, sample space, outcome and event.

An experiment is a process for which the outcome cannot be specified in advance, but for which all possible outcomes are known in advance. The outcome is the final result or observation of the experiment. A sample space is a set that contains all possible outcomes of a given experiment. This is typically described as Ω. Since it is a set, one can classify the sample space by the number of element it contains. These can be finite, countably infinite, or uncountably infinite. A sample space that is finite or countably infinite is called a discrete sample space while an uncountably infinite sample space, such as all the points of an interval, is called a continuous sample space. An event is a subset of the sample space, therefore events are collections of outcomes of an experiment. Those events that consist only of a single element $\omega \in \Omega$ are called elementary events.

Probability theory is based on three axioms defined by the Russian mathematician A. N. Kolmogorov. Before introducing those axioms, first, the event space $\mathcal{A}$ has to be discussed. The event space is the domain of the function $P: \mathcal{A} \to R$, which assigns a probability to every possible event. Therefore the event space is the set of all events in the sample space Ω.

Axiom 1: $For\ any\ event\ A \subset \mathcal{A}, P(A) \geq 0$

Axiom 2: $P(\mathcal{A}) = 1$

Axiom 3: Let I be a finite or countably infinite index set of positive integers, and let $\{A_i : i \in I\}$ be a collection of disjoint events contained in S. Then, $P(\bigcup_{i \in I} A_i) = \sum_{i \in I} P(A_i)$

A disjoint event means that $A \cap B = \emptyset$, for two composed events A and B.

Axiom 1 states that a probability can never be negative and axiom 2 states that the maximum probability is 1. Since $\mathcal{A}$ contains all possible outcomes, the probability for an outcome which is in $\mathcal{A}$ is 1. The third axiom means that probabilities are additive. A function that satisfies all three axioms is called a probability measure. All problem types of assigning probabilities to events in a sample space share a common mathematical structure. It is a 3-tuple which is called the probability space $(\Omega, \mathcal{A}, P)$. Ω contains all possible outcomes, $\mathcal{A}$ is a set of sets which represents all events for which a probability will be defined and P is a probability set function which assigns probabilities to events in $\mathcal{A}$.

Some important probabilistic theorems

Theorem A.1: Let $A \subseteq \Omega$, then $P(A) = 1\text{-}P(A^c)$ with $A^c = \Omega \setminus A$ which means A^c is the complement of A.

Theorem A.2: $P(\emptyset)=0$

Theorem A.3: Let A and B be two events in a sample space such that $A \subset B$. Then $P(A) \leq P(B)$ and $P(B\text{-}A)=P(B)\text{-}P(A)$

Theorem A.4: $A \subset \Omega$ and $B \subset \Omega$, then $P(A) = P(A \cap B) + P(A \cap B^C)$

Theorem A.5: $A \subset \Omega$ and $B \subset \Omega$, then $P(A \cup B) = P(A) + P(B) - P(A \cap B)$

For proof of these theorems, I refer the reader to Mittelhammer (1999).

Conditional Probability and Independence

Conditional probabilities are used to determine changes in probabilities according to some additional information. For instance, if the question is to determine the outcome of tossing two coins sequentially, probabilities change after the outcome of the first coin is determined. Potential outcomes are {(H,H),(H,T),(T,H),(T,T)}. Prior to the first toss, all events have probabilities of ¼. If the first toss' outcome is H, both events, {(T,H),(T,T)} are impossible, which means P((T,H))=0 and P((T,T))=0, while the probability for both other outcomes is 1/2. Therefore there is a need to define conditional probabilities.

Definition A.1: (Conditional Probability) The conditional probability of event A, given event B, is given by $P(A \mid B) = \frac{P(A \cap B)}{P(B)}$ for two events A and B in the sample space.

A probability function based on conditional probabilities has to obey to the same axioms as a normal probability function. Therefore, the theorems discussed above are applicable accordingly. Note that conditional to B means that the event B is certain, therefore the conditional probability $P(B \mid B) = 1$.

This can be put in terms of relative frequency in cases where there is a finite sample space, an event space that contains all subsets of the sample space and a probability function that assigns a probability for every $A \subset S$ as $P(A) = \frac{N(A)}{N(S)}$. In this case conditional probability on an event B is given by

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)} = \frac{\frac{N(A \cap B)}{N(S)}}{\frac{N(B)}{N(S)}} = \frac{N(A \cap B)}{N(B)}$$

An alternative presentation of conditional probabilities is known as Bayes's rule. It is based on the theorem of total probability.

Theorem A.6: (Theorem of total probability) Let the events $B_i, i \in I$, be a finite or countably infinite partition of the sample space, S, so that $B_j \cap B_k = \emptyset$ for any $j \neq k$ and $\bigcup_{i \in I} B_i = S$. Let $P(B_i) > 0 \forall i \in I$. Then $P(A) = \sum_{i \in I} P(A \mid B_i) P(B_i)$.

This means that the total probability of A is distributed over all partitions of the sample space. Derived from that is Bayes's rule.

Theorem A.7 (Bayes's Rule) Let the events $B_i, i \in I$, be a finite or countably infinite partition of the sample space, S, so that $B_j \cap B_k = \emptyset$ for $j \neq k$ and $\bigcup_{i \in I} B_i = S$. Let $P(B_i) > 0 \forall i \in I$. Then, provided $P(A)=0$:

$$P(B_j \mid A) = \frac{P(A \mid B_j) P(B_j)}{\sum_{i \in I} P(A \mid B_i) P(B_i)}$$

In a two event case, Bayes's rule can be written as follows:

$$P(B \mid A) = \frac{P(A \mid B) P(B)}{P(A \mid B) P(B) + P(A \mid \neg B) P(\neg B)}$$

Opposed to conditional probability, in which one event influences the outcome of the experiment, there is the notion of independence.

Definition A.2: (Independence) For two events A and B in the sample space, A and B are called independent if $P(A \cap B) = P(A) P(B)$. If A and B are not independent, they are dependent events.

For two independent events A and B we can write

$$P(A \mid B) = P(A \cap B) / P(B) = P(A) P(B) / P(B) = P(A)$$

$$P(B \mid A) = P(B \cap A) / P(A) = P(B) P(A) / P(A) = P(B)$$

so one can see that the probability is not influenced by the other events. Note that if A and B are independent, A and B^c, A^c and B as well as A^c and B^c are also independent.

Random Variables and Distribution Functions

Many results of experiments have the form of real numbers. Other experiments that do not have real numbers as outcomes often have a countable number of outcomes that can be codified as real numbers. Such an outcome is called a random variable.

Definition A.3: (Random Variable) Let $(\Omega, \mathcal{A}, P)$ be a probability space. If $X: \Omega \to R$ is a real-valued function having as its domain the elements of S, then $X: \Omega \to R$ (X for short) is called a random variable.

Using random variables induces a real valued sample space $R(X) = \{x \mid x = X(w), w \in S\}$, where X is the random variable and x is the image of X in the sample space. Next one needs to define the probability space that is induced by a random variable. For that one needs to define a probability set function and the event space. The induced probability space can be written as $(R(X), \mathcal{A}_x, P_x)$ with

$$R(X) = \{x \mid x = X(w), w \in S\}$$

$$\mathcal{A}_x = \{\mathrm{A} \mid \mathrm{A \text{ is an event in } R(X)}\}$$

$$P_x(A) = P(B), B = \{w \mid X(w) \in A, w \in A\}, \forall A \in \mathcal{A}_x$$

$R(X)$ is the sample space expressed as a real number, $\mathcal{A}_x$ is the event space for outcomes of the random variable and P_x is the probability set function which is defined on events in $\mathcal{A}_x$.

Next the probability density function needs to be defined to be able to define the probability set function. This is first done for the discrete case.

Definition A.4: (Discrete Random Variable)A random variable is called discrete if its range consists of a countable number of elements.

Definition A.5: (Discrete Probability Density Function) The discrete probability density function f, for a discrete random variable X is defined as f(x)=probability of x, $\forall x \in R(X)$ and f(x)=0 $\forall x \notin R(X)$.

Since any event in $\mathcal{A}_x$ is a subset of $R(X)$ we can use Axiom 3 to define that $P_x(A) = \sum_{x \in A} f(x)$ with $A \in \mathcal{A}_x$. The probability density function f can be described as a table or as an algebraic specification which is typically preferable. In the case of discrete probability function $f(x)$, the results can be interpreted as the probabilities of the elementary events $x \in R(x)$. In the case of continuous random variables $f(x)$ cannot be interpreted as a probability since there are uncountably many elementary events in $R(x)$. Therefore we cannot just sum up the probabilities of all elementary events. Rather than that, it is necessary to integrate over uncountably infinite events in order to define a probability.

Definition A.6: (Continuous Random Variable) A random variable is called continuous if its range is uncountably infinite and if there exists a nonnegative-valued functions $f(x)$, defined for all $x \in (-\infty, \infty)$, such that for any event $A \subset R(X), P_X(A) = \int_{x \in A} f(x)dx$ and $f(x) = 0 \forall x \notin R(X)$. The function $f(x)$ is called a continuous probability density function

Note that for any elementary event x, $f(x)=0$, therefore it should not be interpreted as the probability but only as the density value of $f(x)$. In order to calculate probabilities integration is necessary.

Derived from the probability density function f one can create the cumulative distribution function F.

Definition A.7: (Cumulative Distribution Function) A cumulative distribution function of a random variable X is defined by $F(b) = P(x \leq b) \forall b \in (-\infty, \infty)$. We consider two cases for which F has to be defined:

1. Discrete X:
$$F(b) = \sum_{\substack{x \leq b \\ f(x)>0}} f(x)$$
2. Continuous X:
$$F(b) = \int_{-\infty}^{b} f(x)dx$$

A cumulative density function (CDF) can be used to ask question like what is the probability of getting less than 3 or less when throwing a fair dice. To answer such a question one can use the CDF. In the dice example one can simply add all events smaller or equal to the result three to receive ½ as the result.

A CDF's domain is the real line, while the range is the interval [0,1]. Also,

$$\lim_{b \to -\infty} F(b) = \lim_{b \to -\infty} P(x \leq b) = P(\varnothing) = 0$$
$$\lim_{b \to \infty} F(b) = \lim_{b \to \infty} P(x \leq b) = P(R(X)) = 1.$$

Furthermore, if $a < b$ then $F(a) = P(x \leq a) \leq P(x \leq b) = F(b)$. It is possible to create a discrete probability density function from a continuous density function by choosing a countable collection of outcomes $x_1<x_2<x_3<\ldots$ for a discrete random variable X. $f(x)$ can then be defined as

$f(x_1)=F(x_1)$

$f(x_i)=F(x_i)-F(x_{i-1}),\ i=2,3,\ldots,$

$f(x)=0\ for\ x \notin R(X)$

Mathematical Expectation

The expected value of a random variable can be considered to be the center of gravity of its density function. It can be defined as

Definition A.8: (Expected Value) The expected value of a discrete random variable exists, and is defined by $E(X)=\sum_{x\in R(X)} xf(x)$, iff $\sum_{x\in R(X)} |x| f(x) < \infty$,

Note that the existence criterion is necessary to make sure that the sum absolutely converges. Only in that case it is possible to define the expected value. The above definition is valid only for discrete values. The definition for a continuous random variable is given next:

Definition A.9 (Expected Value)The expected value of a continuous random variable exists, and is defined by $E(X)=\int_{-\infty}^{\infty} xf(x)dx$, iff $\int_{-\infty}^{\infty} |x| f(x)dx < \infty$

One can consider the expected value as a weighted average of the possible outcomes of the random variable in the continuous case. The expected value is also known as the mean of the random variable X and denoted by the symbol μ.

Another important measure for random variables is the variance of a random variable and the standard deviation.

Definition A.10: The variance of a random variable X is calculated as $E(X-\mu)^2$ and will be denoted as σ^2, or by $var(X)$.

Definition A.11: The standard deviation of the random variable X is the nonnegative square root of its variance ($\sqrt{\sigma^2}$) and is denoted by σ

Both the standard deviation and the variance are measures for the spread of the density function. The less spread there is around the mean, the smaller are both the variance and naturally the standard deviation.

So far the discussion has evolved only on univariate random variables where only a single probability density function has been defined. As an extention to this there is the multivariate case in which there is more than one real-valued function defined on the elements of the sample space. In this case the probability density function is called the joint probability density function, f, which is defined on a multivariate random variable $X=(X_1,\ldots,X_n)$. For the multivariate case, one can define measures to analyze the interaction of two or more random variables. For instance the question on whether there is a linear association between two random variables X and Y. This can be measured through the use of covariance:

Definition A.12: Covariance is calculated as $\sigma_{XY} = E((X-E(X))(Y-E(Y)))$. It is denoted by σ_{XY} or by $cov(X,Y)$.

Covariance can alternatively be calculated as

$$\sigma_{XY} = E(XY) - E(X)E(Y)$$

Proof:

$$\begin{aligned}\sigma_{XY} &= E((X-E(X))(Y-E(Y))) \\ &= E[XY - E(X)Y - XE(Y) + E(X)E(Y)] \\ &= E(XY) - E(X)E(Y)\end{aligned}$$

Covariance is bound by $|\sigma_{XY}| \leq \sigma_X \sigma_Y$. In order to be able to compare covariances, a scaled version has been developed. It is called the correlation between X and Y

Definition A.13 The correlation between two random variables X and Y is defined by $corr(X,Y) = \rho_{XY} = \frac{\sigma_{XY}}{\sigma_X \sigma_Y}$

Correlation is bound $-1 \leq \rho_{xy} \leq 1$, where $\rho_{xy} = -1$ means there is a perfectly negative relationship between the random variable X and Y while $\rho_{xy} = 1$ implies a perfectly positive relationship. $\rho_{xy} = 0$ means there is no linear relationship between X and Y, though there might still be a different kind of common variance. If both random variables are independent, correlation $\rho_{xy} = 0$, though since there might be another kind of dependence, $\rho_{xy} = 0$ does not imply independence.

Hypothesis Testing

The goal of hypothesis testing is to draw conclusions from a random sample thereby being able to control the probability of error. To be able to test a hypothesis it first needs to be stated in a way that it is part of the probability space of the random sample. The goal of the statistical test is to either accept what is called the null hypothesis or to reject it, if the outcome of the random sample does not fit the expectation under the null hypothesis.

Definition A.14 (Statistical Hypothesis) A set of potential probability distributions for a random sample from a process or population is called a statistical hypothesis. The hypothesis to be tested is called the null hypothesis which is denoted as H_0. If H_0 is rejected is, it is rejected in favor of the alternative hypothesis H_A.

A further goal of statistical testing is to provide means to control for making incorrect decisions. When considering hypothesis testing whose outcome is either true or false, there are four possible outcomes in terms of correctness or incorrectness of the results. First, the hypothesis is correct and the test says just that and second, the hypothesis is incorrect and the test rejects it. In those two cases the statistical test has decided correctly. Similarly there are two situations in which error occurs.

Defintion A.15 (Statistical Error)Let H be a statistical hypothesis being tested for acceptance or rejection. Then the two types of errors that can be made by the statistical test are
1. type I error: rejecting H when H is true
2. type II error: accepting H when H is false.

In statistical testing, one controls type I error actively by choosing a significance level α. For instance choosing a significance level $\alpha = 0{,}05$ means that one in 20 test is expected to reject the null hypothesis though it is actually true. Simply decreasing the significance level does not help since a less strict rejection criterion will increase the risk of type II errors. This is since choosing a significance level partitions the random sample outcome space in acceptance and rejection areas. When trying to avoid type I error, this means the acceptance area is enlarged which leads to a increased probability of falsely accepting H.

The process of performing a statistical test can be divided into four steps. First, one needs to state the null hypothesis H_0 which is being tested and the alternative hypothesis H_A which has to be assumed true if H_0 is rejected. Second, the outcome of the test has to be calculated depending on the type of test that is being applied. This outcome has to be compared to the values for accepting or rejecting the hypothesis. To be able to do that, one needs to choose a significance level which is used to partition the set of random-sample

outcome in a critical region, in which the hypothesis is rejected, and an acceptance region. Now in a fourth step one can decide whether the calculated test statistic lies within the acceptance region and should therefore be accepted or not.

Literature

Agrawal R., Imielinski T., & Swami A. N.. *Mining association rules between sets of items in large databases*. In Proceedings of the 1993 ACM International Conference on Management of Data (SIGMOD'93), 1993.

Agrawal R., & Srikant R. (1994). *Fast Algorithms for Mining Association Rules*, In Proceedings. of the 20th Intl. Conf. on Very Large Databases. Sep 12-15, Chile, 487-99

Atzmüller, M., & Puppe, F: (2006) SD-Map - A Fast Algorithm for Exhaustive Subgroup Discovery. In. J. Fürnkranz, T. Scheffer, and M. Spiliopoulou (Eds.): *Knowledge Discovery in Databases*: *PKDD* 2006: Lecture Notes in Artificial Intelligence 4213 6-17. Springer-Verlag Berlin Heidelberg

Bay, S. D. & Pazzani, M. J.. (2001) *Detecting group differences: Mining contrast sets*. Data Mining and Knowledge Discovery, 5(3):213–246, 2001.

Bayardo, R. J. (1998). *Efficiently mining long patterns from databases*. Proceedings of the ACM SIGMOD Conference on Management of Data. p. 85-93

Berlanga, F., del Jesus, M. J., Gonzáles, P., Herrera, F., & Mesonero, M. (2006). Multiobjective Evolutionary Induction of Subgroup Discovery Fuzzy Rules: A Case Study in Marketing. In P. Perner (Ed.): *Advances in Data Mining*. 6th Industrial Conference on Data Mining, ICDM 2006, Lecture Notes in Artificial Intelligence 4065, pp. 337-349. Springer-Verlag Berlin Heidelberg

Bonchi, F., & Giannotti, F. (2005) Pushing Constraints to Detect Local Patterns. K. Morik et al. (Eds.): *Local Pattern Detection* - Lecture Notes Artificial Intelligence 3539, pp. 1–19, Springer-Verlag Berlin Heidelberg

Ceglar, A. & Roddick, J. F. (2006). *Association Mining*. ACM Computing Surveys, 38 (*2*). Article 5

Domschke, W. & Drexl, A. (2002). *Einführung in Operations Research*. 5. Auflage, Springer Berlin, Heidelberg, New York. pp 114-124

Fayyad, U. M., Piatetsky-Shapiro, G. & Smyth, P. (1996). From Data Mining to Knowledge Discovery: An Overview. In Fayyad, U. M., Piatetsky-Shapiro, G. and Smyth, P and Uthurusamy R. (Eds.). *Advances in Knowledge Discovery and Data Mining*. AAAI Press / The MIT Press. pp. 1-34

Fayyad, U. M., & Irani, K. B. (1993). Multi-Interval Discretization of Continuous-Valued Attributes for Classification Learning. In Ruzena Bajcsy (Ed.): *Proceedings of the 13th International Joint Conference on Artificial Intelligence*. Chambéry, France, 1993. Morgan Kaufmann, pp. 1022-1029

Fisher, D. H. (1987). Knowledge Acquisition Via Incremental Conceptual Clustering. *Machine Learning* 2(*2*), pp. 139-172

Fürnkranz, J. (1999). Separate and Conqure Rule Learning. *Artificial Intelligence Review* 13, pp. 3–54, 1999.

Fürnkranz, J. (2005). From local to global patterns: Evaluation issues in rule learning algorithms. In Morik, K., Boulicaut, J.-F., and Siebes, A. (Eds.), *Local Pattern Detection*, pp. 20-38. Springer-Verlag.

Fürnkranz, J. & Flach, P. (2005). *ROC 'n' rule learning -- towards a better understanding of covering algorithms*. Machine Learning, 58(*1*):39-77

Gamberger, D., & Lavra˘c, N. (2000) Confirmation rule sets. In Zighed, D.A., Komorowski, J. & Zytkow, J. (Eds.) *Principles of Data Mining and Knowledge Discovery PKDD 2000*, Lecture Notes in Artificial Intelligence 1910, pp.34–43, Springer

Gamberger D., & Lavrač, N. (2002a) Generating Actionable Knowledge by Expert-Guided Subgroup Discovery. In T. Elomaa, H. Mannila, and H. Toivonen, editors, *Proceedings of the 6th European Conference on Principles of Data Mining and Knowledge Discovery* - Lecture Notes in Artificial Intelligence 2431, pp. 163--174. Springer-Verlag, August.

Gamberger, D. & Lavrač, N. (2002b). *Expert guided subgroup discovery: Methodology and application*. Journal of Artificel Intelligence Research 17, pp. 501-527

Gamberger, D., & Lavrač, N. (2006) Relevancy in Constraint-Based Subgroup Discovery. In J.-F. Boulicaut et al. (Eds.): *Constraint-Based Mining* - Lecture Notes in Artificial Intelligence 3848, pp. 243–266, 2005. Springer-Verlag Berlin Heidelberg

Goethals, B. (2003). *Survey on Frequent Pattern Mining.* Manuscript http://www.adrem.ua.ac.be/bibrem/pubs/fpm_survey.pdf (25.09.2007)

Han, J., Pei, J., Yin, Y. and Mao, R. (2004) *Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach*. Data Mining and Knowledge Discovery, 8(*1*):53-87

Hand, D. J. (2002). Pattern Detection and Discovery. In D.J. Hand et al. (Eds.): *Pattern Detection and Discovery*, Lecture Notes Artificial Intelligence 2447, pp. 1–12

Hoche, S. & Wrobel, S. (2001) Relational Learning Using Constrained Confidence-Rated Boosting. In C. Rouveirol and M. Sebag (Eds.): *Inductive Logic Programming, 11th International Conference, ILP 2001* – Lecture Notes in Artificial Intelligence 2157. pp. 51–64

Höppner, F. (2005). Local Pattern Detection and Clustering. In Morik, K., Boulicaut, J.-F., and Siebes, A. (Eds.), *Local Pattern Detection* - Lecture Notes Artificial Intelligence 3539, pp 20-38. Springer-Verlag.

Jovanoski, V., & Lavrač, N. (2001) Classification Rule Learning with APRIORI-C. In P. Brazdil and A. Jorge (Eds.): *Progress in Artificial Intelligence* – 10th Portuguese Conference on Artificial Intelligence - Lecture Notes Artificial Intelligence 2258, pp. 44–51,

Kavšek, B., Lavrač, N. & Jovanoski, V. (2003). APRIORI-SD: Adapting Association Rule Learning to Subgroup Discovery. In M.R. Berthold et al. (Eds.): *Advances in Intelligent Data Analysis V* – Proceedings on 5th International Symposium on Intelligent Data Analysis 2003, Lecture Notes in Computer Science 2810, pp. 230–241, 2003. Springer-Verlag Berlin Heidelberg

Kerber R. (1992). ChiMerge: Discretization of Numeric Attributes. In William R. Swartout (Ed.): *Proceedings of the 10th National Conference on Artificial Intelligence*. San Jose, CA, July 12-16, 1992,. The AAAI Press / The MIT Press, pp. 123-128

Klösgen, W. (1996) Explora: A multipattern and multistrategy discovery assistant. In U. M.Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pp. 249–271. AAAI Press,1996.

Klösgen, W. (1998) Deviation and Association Patterns for Subgroup Mining in Temporal, Spatial, and Textual Data Bases. *Proceedings on First International Conference on Rough Sets and Current Trends in Computing: RSCTC'98* - Lecture Notes Artificial Intelligence 1424. pp.1-18.

Kloesgen,W., & May, M. (2002a). *Census Data Mining—An Application*. ECML/PKDD'02 Workshop on "Mining Official Data". http://www.di.uniba.it/~malerba/activities/mod02/pdfs/kloesgen.pdf (29.01.2008)

Klösgen W. & May M. (2002b). Spatial Subgroup Mining Integrated in an Object-Relational Spatial Database. In T. Elomaa et al. (Eds.): *Principles of Data Mining and Knowledge Discovery* – Lecture Notes in Artificial IntelligenceI 2431, pp. 275-286, Springer-Verlag Berlin Heidelberg.

Knobbe, A. J., & Ho, E. K. Y. (2006). Pattern Teams. In J. Fürnkranz, T. Scheffer, and M. Spiliopoulou (Eds.): *Knowledge Discovery in Databases: PKDD 2006*: Lecture Notes in Artificial Intelligence 4213. pp. 577-584. Springer-Verlag Berlin Heidelberg

Kralj, P., Lavrac, N., Gamberger, D., & Krstacic, A. (2007) *Contrast Set Mining Through Subgroup Discovery Applied to Brain Ischaemina Data*. The 11th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2007) 579-586

Lavrač, N. Železný, F. & Flach, P. A.. (2002a). RSD: Relational subgroup discovery through first-order feature construction. *Inductive Logic Programming. 12th International Conference, ILP 2002*. Lecture Notes on Artificial Intelligence 2583 pp. 149-165., Springer-Verlag Berlin Heidelberg

Lavrač, Nada, Flach, Peter, Kavšek, Branko & Todorovski, Ljupčo, (2002b). Adapting classification rule induction to subgroup discovery. *In Proceedings of the IEEE International Conference on Data Mining (ICDM'02),* pp. 266–273.

Lavrač, N. Kavšek, B. Flach, P. A. & Todorovski, L (2004a). *Subgroup Discovery with CN2-SD*. Journal of Machine Learning Research (*5*) pp. 153--188,

Lavrač, N., Cestnik, B., Gamberger, D., & Flach, P. (2004b) *Decision Support through subgroup discovery: Three Case Studies and the Lessons learned*. Machine Learning, 57, pp. 115-143.

Lavrač, N., Železný, P. & Džeroski, S. (2005). Local Patterns: Theory and Practice of Constraint-Based Relational Subgroup Discovery. In Carbonell, J. G. and Siekmann J. (Eds.) *Local Pattern Detection, International Seminar.* Lecture Notes in Artificial Inteligence 3539

Lin, Jessica & Keogh, Eamonn (2006) Group SAX: Extending the Notion of Contrast Sets to Time Series and Multimedia Data. In. J. Fürnkranz, T. Scheffer, and M. Spiliopoulou (Eds.): *Knowledge Discovery in Databases: PKDD 2006*: Lecture Notes in Artificial Intelligence 4213, pp. 284 – 296, 2006. Springer-Verlag Berlin Heidelberg 2006

Lin, J., Keogh, E., Lonardi, S., & Chiu, B. (2003). A Symbolic Representation of Time Series, with Implications for Streaming Algorithms. In *Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*. http://www.cs.ucr.edu/~eamonn/SAX.pdf (28.01.2008)

Liu, B., Hsu W. & Ma Y. (1998). *Integrating Classification and Association Rule Mining*. Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining KDD-98. pp. 80-86

Mirkin, B. (2001). *Reinterpreting the Category Utility Function*. Machine Learning 45, pp. 219-228

Mitchell, T. M. (1997). Machine Learning. McGraw-Hill Companies, Inc. New York. pp 55-78, 274-304

Mittelhammer, R. C. (1999). *Mathematical Statistics for economics and business*. 3. corr. Print. Springer. New York

Morishita, S. & Sese, J. (2000). *Traversing Itemset Lattices with Statistical Metric Pruning*. In: Proceedings of the Nineteenth ACM SIGACT-SIGMOD-SIGART Symposiumon Principles of Database Systems, ACM (2000) 226–236

Russell, S., & Norvig, P. (2003). Artificial Intelligence: A Modern Approach. 2nd edition, Prentice Hall, New Jersey.

Rymon, R. (1992). *Search through Systematic Set Enumeration*. In Proc. of Third Int '1 ConJ on Principles of Knowledge Representation and Reasoning, 539-550.

Silberschatz, A., & Tuzhilin, A. (1995). *On subjective measures of interestingness in knowledge discovery*. In Proceedings of the First International Conference on Knowledge Discovery and Data Mining, 1995 pp. 275–281. http://citeseer.ist.psu.edu/silberschatz95subjective.html. (28.01.2008)

Silverstein, C., Brin, S. & Motwani, R. (1998). *Beyond market baskets: Generalizing Association Rules to Dependence Rules*. Data Mining and Knowledge Discovery, 2(*1*). p. 39-68.

Smyth, P. & Goodman, R. M. (1992). *An Information Theoretic Approach to Rule Induction from Databases*, IEEE Trans. Knowledge and Data Eng., 4(*4*),pp. 301-316.

Scholz, M. (2005). *Sampling-based sequential subgroup mining*. Conference on Knowledge Discovery in Data Proceeding of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining. pp. 265 - 274

Suzuki, E. (2004) *Discovering Interesting Exception Rules with Rule Pair*. ECML/PKDD 2004 15th European Conference on Machine Learning (ECML) 8th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD). www.ke.informatik.tu-darmstadt.de/ events/ECML-PKDD-04-WS/Proceedings/suzuki.pdf (28.01.2008)

Webb, G. I., Butler, S. M., & Newlands, D. (2003) On Detecting Differences Between Groups Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2003)

Wrobel S. (1997). An algorithm for multi-relational discovery of subgroups. In: J. Komorowski and J. Zytkow (eds.). *Principles of Data Mining and Knowledge Discovery*, First European Symposium. pp. 78 - 87, Springer Verlag, Berlin, New York, 1997

Železný, F. & Lavrač, N. (2006). Propositionalization-based relational subgroup discovery with RSD. *Machine Learning* 62 (1-2). 33 - 63

Zimmermann A. & De Raedt, Luc (2005) Inductive Querying for Discovering Subgroups and Clusters. In. Jean-François Boulicaut Luc De Raedt and Heikki Mannila (Eds.) *Constraint-Based Mining and Inductive Databases*. LNCS 3848. pp. 380-399