Fuzzy Association Rules

An Implementation in R

Master Thesis

Vienna University of Economics and Business Administration

Author

Bakk. Lukas Helm

Matriculation Number: 0251677

E-Mail: lukas.helm@gmx.at

Coordinator

Priv.Doz. Dr. Michael Hahsler

Institute for Information Business

Vienna, 2.8.2007

Fuzzy Association Rules

Abstract

The idea of empowering classical association rules by combining them with fuzzy set theory has already been around since several years. The original idea derives from attempts to deal with quantitative attributes in a database, where subdivision of the quantitative values into crisp sets would lead to over- or under-estimating values near the borders. Fuzzy sets can overcome that problem by allowing partial memberships to the different sets. Fuzzy set theory provides the tools needed to do the computations in order to be able to deal with the different data structure. This paper will give an introduction to association rules and fuzzy set theory, combining the approaches to enable the mining of fuzzy association rules. Different approaches to fuzzy association rules exist, each of which is presented. Even though a lot of research has been done on the topic and algorithms have been proposed, not many programs provide the functionality yet. In the proceeding of this thesis, a package has been developed which can be applied on quantitative data and demonstrate the mining of fuzzy association rules.

Keywords

Fuzzy Association Rules, Fuzzy Set Theory, Quantitative Association Rules, Quality Measures, Frequent Patterns, FP-Growth

Table of Contents

1 Introduction

In the 1990s, the evolution of information technologies and especially the networks like the Internet enabled companies to easily record data from their customers. Since then, huge amounts of data have been collected and stored in the databases of many enterprises. Due to the fact that a lot of business intelligence is hidden in the data, the companies need tools to find out patterns and regularities. As many of the databases are very large containing a huge number of tuples and attributes, efficient automated tools are necessary for acquiring useful information.

Therefore, many data mining tools have been developed that allow a great variety of analysis techniques, mostly derived from classical statistics. Since its introduction in [AgIS93], the technique of association rules mining has received great interest by the data mining community and a lot of research has been done resulting in the development of many different algorithms. Association rules are especially useful for conducting a market basket analysis where transaction data can be analyzed. Regularities in data of a supermarket for example can be found in this way. An association rule could be “If a customer buys bread and milk, he will mostly buy butter as well”. This information is very useful for business because promotion actions can be designed accordingly.

Association rules mining is especially efficient for its use in the Internet. The data are easily available and mining can thereby be done quickly. Discovered rules can help online-shops in personalizing their website and cross-selling their products by making recommendations. New transactions can be used quickly for mining and give the company a competitive advantage.

A problem of classical association rules is that not every kind of data can be used for mining. Rules can only be derived from data containing binary data, where an item either exists in a transaction or it does not exist. When dealing with a quantitative database, no association rules can be discovered. This fact led to the invention of quantitative association rules, where the quantitative attributes are split into intervals and the single elements are either members or nonmembers of those intervals. With this approach, a binary database can be constructed out of a quantitative one.

The quantitative approach allows an item either to be member of an interval or not. This leads to an under- or overestimation of values that are close to the borders of such “crisp” sets. To overcome this problem, the approach of fuzzy association rules has been developed. It allows the intervals to overlap, making the set fuzzy instead of crisp. Items can then show a partial membership to more than one set, overcoming the above addressed, so-called “sharp boundary problem”. The membership of an item is defined by a membership function and fuzzy set theoretic operations are incorporated to calculate the quality measures of discovered rules. Using this approach, rules can be discovered that might have got lost with the standard quantitative approach.

This thesis gives an overview of association rules mining as well as the approach of fuzzy associations mining and an introduction to fuzzy set theory which is necessary for mining fuzzy association rules. Additionally, an R-package for mining fuzzy association rules is introduced that has been developed in the proceeding of this thesis. The following chapter describes the structure of the thesis.

1.1 Structure

This thesis is subdivided into five chapters. Chapters one through four provide the theoretical background necessary to understand fuzzy association rules, chapter five presents the conducted project with the outcome of an R-package for fuzzy association rules mining.

1. **Data Mining**

   This chapter contains basic principles, concepts and strategies of data mining together with a classification of the topics where the role of association rules mining is explained. Finally, a process model is presented.

2. **Association Rules**

   The classical concepts of association rules are introduced in this second chapter. It contains the basics of association rules as well as concepts for mining them. In the end, algorithms for mining the rules are presented.

3. **Fuzzy Set Theory**

   The third chapter deals with fuzzy set theory which is the basis for the mining of fuzzy association rules in the subsequent chapters. It compares

fuzzy sets to crisp sets, gives an introduction on fuzzy logic and the operations that can be performed on fuzzy sets.

4. **Fuzzy Association Rules**

   This chapter deals with fuzzy association rules. Different approaches to the term exist which are examined. Also, the modified quality measures and algorithms derived from classical association rules are introduced.

5. **The Project**

   The last chapter deals with the project conducted for the thesis. Architectures and approaches used for building the tool are described. Finally, a more detailed description on how the function work and the implementation itself is given.

2 Data Mining

The following chapter will give a brief introduction to data mining (DM). DM is defined as the discovery of interesting information, patterns or trends from a large database or data warehouse [HaNe01]. Data mining is a subprocess of Knowledge Discovery in Databases in which the different available data sources are analyzed using various data mining algorithms. Speaking of DM we refer to “a multi-disciplinary field involving machine learning, statistics, databases, artificial intelligence, information retrieval and visualization” [Liu07].

The two high-level goals that data miners want to achieve are prediction and description [FaPS96a]:

- Prediction is used to find patterns which can be used to project the future.
- Description represents discovered patterns to the user in a human-understandable form.

The gained knowledge is either represented as a model or generalization of the mined data. Many different data mining techniques have been developed which will be discussed in more detail later. A great number of these techniques descend from classical statistics, nevertheless there are newer approaches that use artificial intelligence approaches. This makes a prediction of trends possible which would not have been feasible with the traditional statistical methods. In addition, the data needed to conduct the data mining is widely available in the age of the Internet and e-commerce.

2.1 Knowledge Discovery in Databases (KDD)

Frawley et al. [FrPM92] state that “Knowledge discovery is the nontrivial extraction of implicit, previously unknown, and potentially useful information from data.” In order to get this information, we try to find patterns in the given data set. To know if a pattern is valuable, the assessment of its interestingness and certainty is crucial. Patterns that are interesting and certain enough according to the user's measures are called knowledge. The output of a program that discovers such useful patterns is called discovered knowledge.

According to [FrPM92], KDD exhibits four main characteristics:

- High-Level Language: The discovered knowledge is represented in a language that does not necessarily have to be directly used by humans, but its expression should be comprehensible.
- Accuracy: The measure of certainty implies whether the discovered patterns portray the contents of a database properly or not.
- Interestingness: Discovered knowledge is considered interesting if it fulfills the predefined biases. By denoting a pattern interesting, we mean that it is novel, potentially useful and the discovery process is nontrivial.
- Efficiency: Even for large Datasets, the running time of the algorithm is acceptable and predictable.

Data and patterns are defined in [FaPS96a]: “Here, data is a set of facts and pattern is an expression in some language describing a subset of the data or a model applicable to that subset. [...] Patterns should be understandable, if not immediately then after some post-processing.” The so-called KDD Process consists of several steps that are in place to achieve the defined goals for knowledge discovery.

2.1.1 The KDD Process

The KDD Process is interactive and iterative and requires decisions made by the user. [FaPS96b] proposes nine basic steps (see Figure 1).

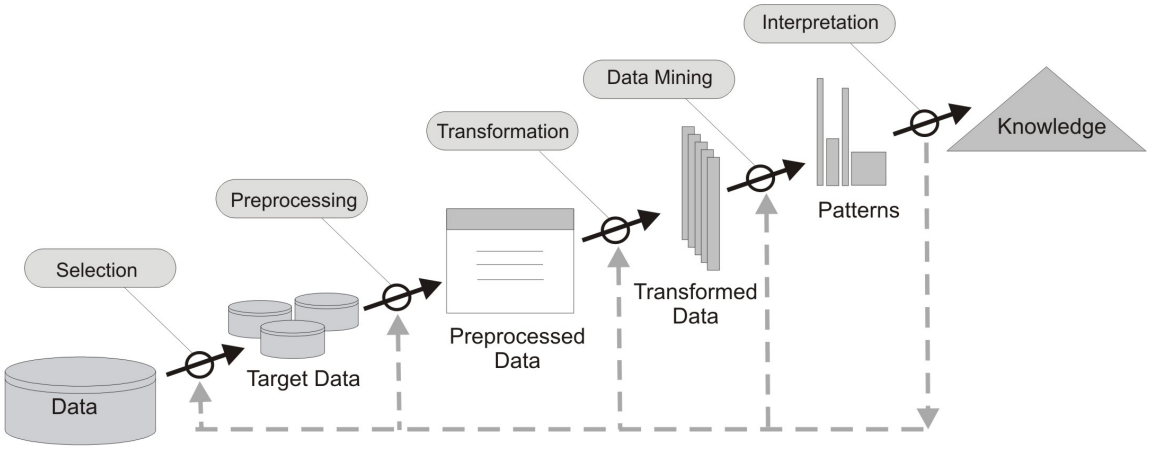

Figure 1: The KDD Process

1. Data Understanding: learning the application domain for prior knowledge and goals of the application.

2. Creating a target data set: selecting the subset of the data on which the data mining will be performed.
3. Data cleaning and preprocessing: removing noise or outliers, developing strategies for handling missing data.
4. Data reduction: reduce dimensionality of the data set in order to get rid of data that is unnecessary for completing the mining task and thereby keep the computing time low.
5. Selecting the data mining method: the most important task here is to find the method that will best suit the completion of the KDD goals.
6. Choosing the data mining algorithm: there are many different data mining algorithms. Deciding on an efficient one to search for patterns in data is critical and includes decisions about appropriate models and parameters.
7. Data mining: applying the previously chosen algorithm to the data set and searching for interesting patterns in a particular representational form.
8. Interpreting mined patterns includes the visualization of mined patterns and a possible return to any of the steps 1-7 if the results are unsatisfactory.
9. Consolidating discovered knowledge: documenting the results and incorporating them into another system.

[FaPS96a] emphasizes a distinction between KDD and data mining. While KDD refers to the overall process of discovering useful knowledge from data, data mining is “a step in the KDD process consisting of applying data analysis and discovery algorithms that, under acceptable computational efficiency limitations, produce a particular enumeration of patterns over the data.”

2.2 Concepts

The last decade brought a huge advance in database technology which lead to a huge amount of data being collected. Thus, we are facing a great chance to make use of this data by extracting previously unknown patterns. Parallel processing constitutes an important technique to realize large-scale data mining applications because they handle a huge amount of data and therefore involve a lot of computation [SuSi06].

A main goal of data mining is to provide business with information in order to make predictions for future use. For this reason, data mining emerged in the 80s and made great progress in the 90s. It still is a research field of great interest. As a consequence, many different data mining concepts have been developed.

2.2.1 Data Warehousing

Organizations build data warehouses by integrating their different operational databases. Data warehousing is the process of centralized data management and retrieval [SuSi06]. The term is relatively new although the concept has been around for years. It represents a vision of installing a central repository of all organizational data relevant to data mining. The goal is to ensure easy access by the user and to allow quick access for data analysis. Data warehousing techniques are already being applied in many companies. Data mining provides the software and techniques to acquire useful information from the data warehouse. The data warehouse provides the company with a memory whereas data mining provides the company with intelligence.

The most important benefit of a data warehouse is to allow data mining in very large databases at a high level of performance and manageability. It integrates operational databases that might be divergent and thereby hard to access allowing much more efficient data mining. Two steps have to be considered for making a data warehouse valuable:

1. Integrate the internal and external data into the data warehouse with all the data needed for mining.
2. Organize and present the information in ways that assists complex decision making.

2.2.2 Predictive vs. Descriptive Data Mining

The two high-level goals in data mining can be defined as prediction and description [FaPS96c]. Prediction uses some fields or variables in the database to predict future values of other interesting variables. Description sets its focus on making the data comprehensible and interpretable. The boundaries between those two goals are not sharp, because some predictive models can be descriptive meaning that they are understandable and vice versa. Nevertheless, the distinction can help understanding the overall mining goal.

2.3 Data Mining Strategies

According to the goal we want to achieve with data mining, there are several data mining strategies to choose from. These strategies can be broadly classified in supervised learning, unsupervised learning and market basket analysis [RoGe03]. Supervised learning is mainly used for prediction. Several input variables are used to build models which predict a specified output variable. Supervised learning methods either allow only one single or several output attributes. Unsupervised learning does not have any output variable but rather tries to find structures in the data by grouping the instances into different classes. The designation of market basket analysis is to find regularities in data in order to explore customer behavior. Results can help retailers design promotions or recommendations. This chapter will provide a closer look at these strategies.

2.3.1 Supervised Learning

Supervised learning is used in almost any domain, mainly for the purpose of prediction. It could also be called classification or inductive learning when used in association with machine learning. The goal is to create a function out of a given set of historical training data. This function generates the desired output, it is for example possible to predict whether a customer will buy a certain product or not. To be able to compute the function, we need enough training data to make an accurate prediction. Historically collected data with information about customers who either bought or did not buy the product after a promotion will enable us to find out which potential customers will react on a promotion campaign. The data about whether the customer reacted to the campaign or not serves as our output variable.

2.3.2 Unsupervised Learning

Unlike in supervised learning, we do not want to predict a specific output here, but rather discover unknown structures that exist within a data set. The technique used for unsupervised learning is clustering. This technique orders the instances of a data set into groups with similar attributes and values. These groups of items are called clusters. It is important to notice that instances of one single cluster are similar to each other, whereas instances of different clusters

are very diverse from each other. By clusters we mean subsets of the overall data set that is being mined. Clusters are created in the mining process without a priori knowledge of cluster attributes. The following section will introduce the two basic types of clustering, i.e. hierarchical and partitional clustering.

- **Hierarchical Clustering**

  Hierarchical clustering does not partition the data into the clusters within a single step. Rather, a series of partitions takes place. The method can be performed agglomerative or divisive. In the divisive version, the algorithm starts off with the initial data set as one big cluster and then further partitions the big group into different smaller clusters using distance measures. The agglomerative version starts by looking at each single element in the data set and linking the two elements with the lowest distance measure. Both methods continue until they reach either the single element or the total set of items (see Figure 2).

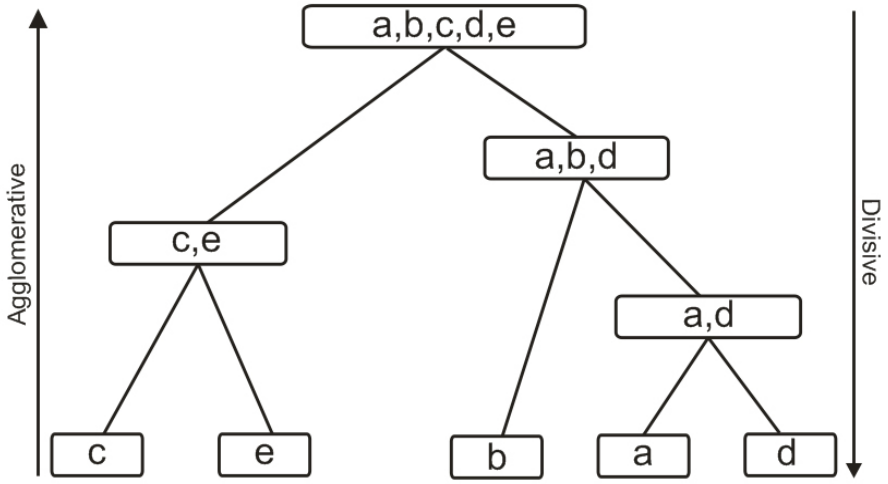

Figure 2: Hierarchical Clustering

- **Partitional Clustering**

  Partitional Clustering is implemented by a well known method, called K-Means Clustering. Here, cluster centers (centroids) are being defined and the data points are assigned to the cluster with the nearest centroid. The algorithm first computes random centers for a predefined number of clusters. Then, it recomputes the centers in order to obtain the best possible centroids.

2.3.3 Market Basket Analysis

Marked Basket Analysis could be put under the domain of unsupervised learning, but in fact it is often treated in literature as a parallel topic. The goal is to discover interesting relationships between retail products in order to help retailers in identifying cross-sale opportunities. Association rule mining is the most common approach for performing this task which will be described in more detail in chapter 3. Such algorithms mainly deal with discovering these items which are frequently purchased together. The name is derived from a person walking through a supermarket throwing all the things to buy in a shopping cart. This “market basket” is then analyzed.

2.4 Tasks

The goals of prediction and description are achieved by using the following primary data mining tasks [FaPS96c]:

1. **Classification:** Individual items are placed into predefined groups. The main task is to find the function that maps data items correctly into the several classes. For example, a bank might want to classify their customers in two groups to foresee which customer will get a loan.
2. **Regression:** A dependent variable is mapped to an independent variable. Here, a function should be learned that maps data points to real-valued prediction value. Cohesion between different variables can thereby be discovered. The results are mainly used for prediction, for example predicting customer's buying behavior after a certain amount of promotion expenditures.
3. **Clustering:** Unlike the regression, clustering is a descriptive task rather than predictive. The goal is to find a finite number of unknown categories in a data set. An application is the subdivision of the customers in a database into several homogeneous subcategories to better understand their behavior. “Closely related to clustering is the task of *probability density estimation* which consists of techniques for estimating from data the joint multivariate probability density function of all of the variables or fields in the database.“ [FaPS96c]

4. **Summarization:** Mainly, summarization provides methods for giving a compact description for a data subset. Simple examples are the mean or standard deviation for all the attributes. More sophisticated methods have also been developed (cp. [ZeZy96]).
5. **Dependency Modeling:** This technique attempts to find models describing significant dependencies between variables. Such dependency models can exist in two levels: (1) The structural level indicates which variables rely locally on each other and is often presented in a graphical form. (2) In the quantitative level, strengths of those relationships can be discovered. Numerical scales are utilized in this case.
6. **Change and Deviation Detection:** The goal of this task simply is to detect deviations from previous measurements or normative values.

2.5 CRISP-DM Model

Conceived in 1996, the CRISP-DM (Cross Industry Standard Process for Data Mining) model has evolved as the standard for conducting data mining activities. At that time, many different data mining approaches had been developed and therefore there was a great need for a unified framework. CRSP-DM emerged as a freely available and non-proprietary framework with a standardized process (see Figure 3).

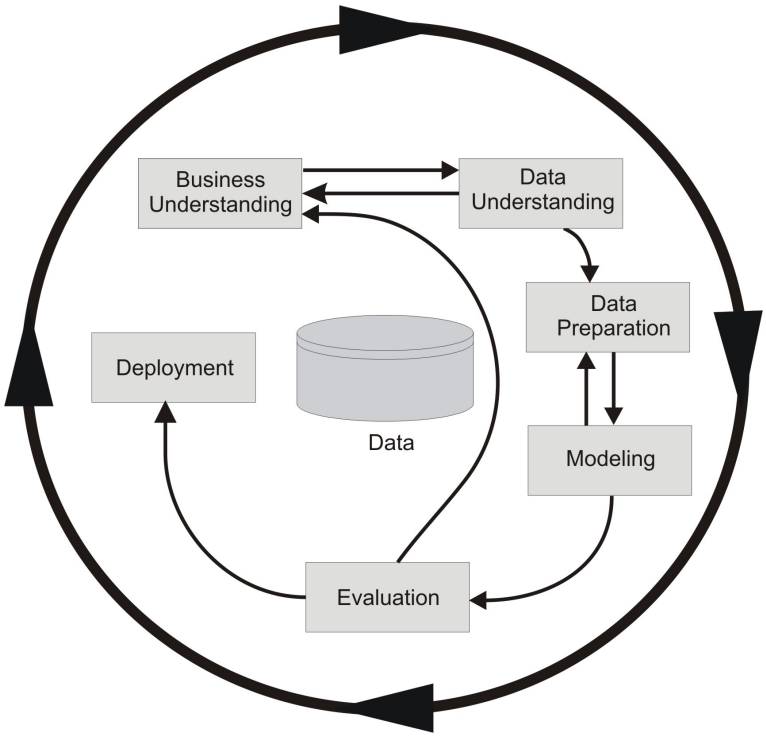

Figure 3: The CRISP-DM Model [CCKK99]

The model defines six phases to conduct a data mining project which are [CCKK99]:

1. **Business Understanding**

   Business understanding is the initial phase of the CRISP-DM model. Most importantly, it focuses on the objectives and requirements of a project from the business perspective. The actual situation in the company is being assessed. After this assessment, the acquired knowledge is converted into the data mining problem definition which is a plan that advises the data mining how to deal with these objectives and requirements of the project

2. **Data Understanding**

   The second phase starts off with a collection of all the available data that might be relevant for the mining project, followed by activities like describing and exploring the data in order to get familiar with them. Another important task is to verify the quality of the data which makes efficient mining possible. This phase helps the participants in getting first insights into the data set.

3. **Data Preparation**

   In the data preparation phase, the relevant data is selected from the overall data set discovered in the previous phase. In a second step, the initial data has to be cleaned, later integrated and formatted. All those activities aim at constructing the final data set that is adequate for mining. The tasks in this phase are likely to be performed multiple times in order to provide a good data set which is crucial for any data mining project.

4. **Modeling**

   The modeling phase deals with selecting possible models for data mining and calibrating the parameters to optimal values for the specific data mining task. Multiple data mining methods might be adequate for mining all of which should be tested at this step. Some models have specific requirements for the data, making a step back to the preparation phase necessary. Discovered models need to be measured and assessed regarding the data mining goal they have to suit.

5. **Evaluation**

   After having built the models for mining, the degree to which it meets the business objectives needs to be measured. A model may have high quality from a data analysis perspective, but might be deficient in meeting the requirements of the business. To certify the achievement of these goals, the model needs further evaluation. The steps for generating the model are reviewed to assess whether any important task or factor has somehow been overlooked. The phase ends with a decision on the use of the data mining results.

6. **Deployment**

   The process does not end with the creation of a correct model. Instead, the gained knowledge needs to be organized and presented in ways that the customer can understand and use. In addition, a model has to be monitored and maintained in order to allow future use. A valid model might not be valid throughout time because customer behavior changes and thus the model might need adjustments. The complexity of the deployment phase highly relies on the requirements defined at the begin-

ning of the project. Often, the customer will have to deal with the subsequent steps which will require further explanation by the developer. The end of the process is marked by the generation of the final report including all the previous deliverables and a summarization and organization of the results.

3 Association Rules

The discovery of association rules constitutes a very important task in the process of data mining. Association rules are an important class of regularities within data which have been extensively studied by the data mining community. The general objective here is to find frequent co-occurrences of items within a set of transactions. The found co-occurrences are called associations. The idea of discovering such rules is derived from market basket analysis where the goal is to mine patterns describing the customer's purchase behavior [Liu07].

Today, mining this type of rules is a very important discovery method in the KDD Process [HiGN00]. A simple association rule could look as follows: $Cheese \rightarrow Beer[support = 0.1, confidence = 0.8]$. Put simply, this rule expresses a relationship between Beer and Cheese. The support measure states that beer and cheese appeared together in 10% of all recorded transactions. The confidence measure describes the chance that there is beer in a transaction provided that there is also cheese. In this case, 80% of all transactions involving cheese also involved beer. We can thereby assume that people who buy cheese are also likely to buy beer in the same transaction. Such information can aid retail companies to discover cross-sale opportunities and guide the category management in this way. In addition, it enables companies to make recommendations which can be especially useful for online retail shops.

Association rule mining is user-centric because its objective is the elicitation of interesting rules from which knowledge can be derived [CeRo06]. Interestingness of rules means that they are novel, externally significant, unexpected, non-trivial, and actionable. An association mining system aids the process in order to facilitate the process, filter and present the rules for further interpretation by the user.

A lot of interest in association rule mining was ignited by the publications [AgIS93] and [AgSr94] in 1993/94. In those papers, an algorithm for mining association rules in large databases has been described. This algorithm will be examined in greater detail in chapter 3.4. Since then, association rule analysis has become a mature field of research. The fundamentals of association mining and itemset identification are well established and accepted.

3.1 Basics

We state the problem of mining association rules as follows: $I=\{i_1, i_2, ..., i_m\}$ is a set of items, $T=\{t_1, t_2, ..., t_n\}$ is a set of transactions, each of which contains items of the itemset I. Thus, each transaction t_i is a set of items such that $t_i \subseteq I$. An association rule is an implication of the form: $X \rightarrow Y$, where $X \subset I$, $Y \subset I$ and $X \cap Y = \emptyset$. X (or Y) is a set of items, called itemset [Liu07].

An example for a simple association rule would be $\{bread\} \rightarrow \{butter\}$. This rule says that if bread was in a transaction, butter was in most cases in that transaction too. In other words, people who buy bread often buy butter as well. Such a rule is based on observations of the customer behavior and is a result from the data stored in transaction databases.

Looking at an association rule of the form $X \rightarrow Y$, X would be called the antecedent, Y the consequent. It is obvious that the value of the antecedent implies the value of the consequent. The antecedent, also called the "*left hand side*" of a rule, can consist either of a single item or of a whole set of items. This applies for the consequent, also called the "*right hand side*", as well.

The most complex task of the whole association rule mining process is the generation of frequent itemsets. Many different combinations of items have to be explored which can be a very computation-intensive task, especially in large databases. As most of the business databases are very large, the need for efficient algorithms that can extract itemsets in a reasonable amount of time is high. Often, a compromise has to be made between discovering all itemsets and computation time. Generally, only those itemsets that fulfill a certain support requirement are taken into consideration. Support and confidence are the two most important quality measures for evaluating the interestingness of a rule.

Support: The support of the rule $X \rightarrow Y$ is the percentage of transactions in T that contain $X \cap Y$. It determines how frequent the rule is applicable to the transaction set T. The support of a rule is represented by the formula

$$supp(X \rightarrow Y) = \frac{|X \cap Y|}{n}$$

where $|X \cap Y|$ is the number of transactions that contain all the items of the rule and n is the total number of transactions.

The support is a useful measure to determine whether a set of items occurs frequently in a database or not. Rules covering only a few transactions might not be valuable to the business. The above presented formula computes the relative support value, but there also exists an absolute support. It works similarly but simply counts the the number of transactions where the tested itemset occurs without dividing it through the number of tuples.

Confidence: The confidence of a rule describes the percentage of transactions containing X which also contain Y.

$$conf\,(X \rightarrow Y) = \frac{|X \cap Y|}{|X|}$$

This is a very important measure to determine whether a rule is interesting or not. It looks at all transactions which contain a certain item or itemset defined by the antecedent of the rule. Then, it computes the percentage of the transactions also including all the items contained in the consequent.

3.1.1 The Process

The process of mining association rules consists of two main parts. First, we have to identify all the itemsets contained in the data that are adequate for mining association rules. These combinations have to show at least a certain frequency to be worth mining and are thus called frequent itemsets. The second step will generate rules out of the discovered frequent itemsets.

1. **Mining Frequent Patterns**

   Mining frequent patterns from a given dataset is not a trivial task. All sets of items that occur at least as frequently as a user-specified minimum support have to be identified at this step. An important issue is the computation time because when it comes to large databases there might be a lot of possible itemsets all of which need to be evaluated. Different algorithms attempt to allow efficient discovery of frequent patterns. Some of those will be presented in chapter 3.4.

2. **Discovering Association Rules**

   After having generated all patterns that meet the minimum support requirements, rules can be generated out of them. For doing so, a minimum confidence has to be defined. The task is to generate all possible rules in the frequent itemsets and then compare their confidence value with the minimum confidence (which is again defined by the user). All rules that meet this requirement are regarded as interesting. Frequent sets that do not include any interesting rules do not have to be considered anymore. All the discovered rules can in the end be presented to the user with their support and confidence values.

3.1.2 Research

The process of mining association rules consists of two parts. The fist part is discovering frequent itemsets in the data. Secondly, we want to deduce inferences from these itemsets. The first step is the much more complex part, thus the majority of related research has focused on itemset discovery. Given E distinct Items within the search space, we have to explore $2^{|E|}$ possible combinations. Due to the fact that $|E|$ is often large, naive exploration techniques are frequently intractable [CeRo06].

Research is focusing on the following topics:

- Restrict the exploration by developing and applying interest measures and pruning strategies.
- Reducing the IO-cost by making use of hardware advances, enabling large datasets to become memory resident or techniques like intelligent sampling.
- Creating useful data structures to make analysis more tractable.
- Producing condensed conclusion sets which allow the whole set to be inferred from a reduced set of inferences, lowering storage and simplifying user interpretation.

A variety of algorithms for performing the association rule mining task have already been developed, most of which focus on finding all relevant inferences in a data set. In addition, increasing attention is given to algorithms that try to im-

prove computing time and user interpretation. Important algorithms for mining association rules will be introduced in chapter 3.4.

3.2 Binary Association Rules

By the term binary association rules, we refer to the classical association rules in market basket analysis. Here, a product can either be in a transaction or not, making only boolean values (true or false, represented by 1 and 0) possible. Every item in a transaction can thus be defined as a binary attribute with domain {0,1}. The formal model is defined in [AgIS93] as follows:

"Let $I=i_1, i_2, \ldots, i_m$ be a set of binary attributes, called items. Let T be a database of transactions. Each transaction t is represented as a binary vector, with $t[k]=1$ if t bought the item i_k, and $t[k]=0$ otherwise. There is one tuple in the database for each transaction. Let X be a set of some items in I. We say that a transaction t satisfies X if for all items $i_k \in X, t[k]=1$."

An association rule is, as already stated in chapter 3.1, an implication of the form $X \rightarrow Y$ where X and Y are sets of items contained in I and Y is not present in X. We call a rule satisfied in T with the confidence factor $0 \leq c \leq 1$ if at least c % of transactions in T that support X also support Y. The notation $X \rightarrow Y|c$ can be used to express that our rule has a confidence factor of c.

In [AgIS93], the problem of rule mining is divided into two subproblems:

- It is necessary to identify all combinations of items that have a transaction support above a certain threshold, called minsupport. We will call those sets of items that show a sufficient support large or frequent itemsets, and those not meeting the threshold small itemsets. Syntactic constraints can also be taken into consideration, for example if we are only interested in rules that contain a certain item in the antecedent or the consequent.
- After identifying the itemsets that satisfy the minsupport, it is important to test whether it satisfies the confidence factor c. Only the previously defined large itemsets have to be tested at this stage. The confidence is computed by dividing the support of the whole itemset by the support of the antecedent.

After having solved the first problem in finding all relevant large itemsets, the second part is rather straightforward. In order to discover large itemsets, the Apriori algorithm was developed as the first and nowadays best known algorithm for mining association rules. The Apriori and other algorithms will be explored in greater detail in chapter 3.4.

3.3 Quantitative Association Rules

The previous chapter provided an overview of binary association rules, where the items can only be represented by boolean values. We will refer to this as the boolean association rules problem. In reality, a database contains not only binary attributes, but also quantitative and categorical ones that can not be mined with the classical technique. Discovering rules in such kind of data can be referred to as the quantitative association rules problem [SrAg96]. The domain of a transaction is a subset of the real numbers rather than {0,1}.

A possibility to deal with such a quantitative attribute is to replace it with several boolean attributes. If the quantitative attributes are categorical or contain only few values, mapping them into binary values is straight forward. Conceptually, instead of having one field for an attribute, we have as many fields as attribute values. In other words, the boolean value corresponding to $\langle attibute1, value1 \rangle$ would be 1 if $attribute1$ had $value1$ in the original database, and 0 otherwise. This only works if there is a very limited number of values in the original data. As the number of different values increases, we will need to split the values into intervals and map each attribute to the corresponding new boolean attribute. From now on, a binary algorithm can be used to discover association rules.

Looking at the sample database in Table 1, we can see that it is necessary to create intervals for all the attributes because they are numeric. The procedure is quite simple. It is just necessary to choose adequate intervals for each row of the table and then map every single tuple to the corresponding new binary attribute. The new table will have more columns with the exact number depending on how many intervals have been chosen for each attribute. Table 2 Shows a mapping table with sample intervals chosen for each attribute.

ID	Age	Income
111	19	900
112	33	1500
113	24	2100
114	37	2500
115	29	1100

Table 1: Sample Database

ID	Age: <20	Age: 20-29	Age: >29	Inc: <1000	Inc: 1000-1999	Inc: >1999
111	1	0	0	1	0	0
112	0	0	1	0	1	0
113	0	1	0	0	0	1
114	0	0	1	0	0	1
115	0	1	0	0	1	0

Table 2: Mapping Table

Two problems arise using this mapping method [SrAg96]:

- “MinSupport”: If the number of intervals found for a single quantitative attribute is high, the support of one single of these intervals can be low. Thus, without lowering the number of intervals some existing rules involving this attribute might not be found after mapping it to binary attributes via intervals.
- “MinConfidence”: By building larger intervals in order to cope with the first problem, we are facing another challenge. The lower the number of intervals is, the more information will get lost. Rules might then appear differently than in the original data.

We are now confronted with a trade-off situation: if the intervals are too large, we might not reach the minimum confidence, if they are too small, we might fail to achieve minimum support. To cope with the “MinSupport” problem, it would be possible to consider all potential continuous ranges over the ranges of the quantitative attribute. The “MinConfidence” problem can be overcome by increasing the number of intervals, without encountering the “MinSupport” problem. This method of increasing the number of intervals while at the same time combining the adjacent ones generates two new problems:

- “ExecTime”: By using the above method, the number of items per record increases, hence the execution time will increase as well.
- “ManyRules”: If a value has minimum support, any range containing this value will have minimum support as well. Hence the number of rules will blow up many of which will not be interesting.

Obviously, there is a trade-off between the different problems. If we build more intervals for coping with the “MinConfidence” problem, we are facing an increase in the execution time and additionally, many uninteresting rules might be generated.

3.4 Algorithms

Several algorithms have been developed since the introduction of the Apriori algorithm [AgIS93]. Those algorithms are attempts to improve the efficiency of frequent pattern and/or association rule discovery. Most of the algorithms focus on either frequent itemset generation or discovering the association rules from the frequent itemsets. In contrary, Apriori provides solutions for both problems. This chapter will give a brief overview of some important mining algorithms. Exploring all the available algorithms for mining association rules would go beyond the scope of this thesis. Most of the algorithms have been developed for use with binary association rules, but they will equally work with the above described quantitative association rules.

There are two main strategies for developing an association rule mining algorithm. Those are called breadth-first search (BFS) and depth-first search (DFS) [HiGN00]. We can think of a lattice including all possible combinations of an itemset (Figure 4).

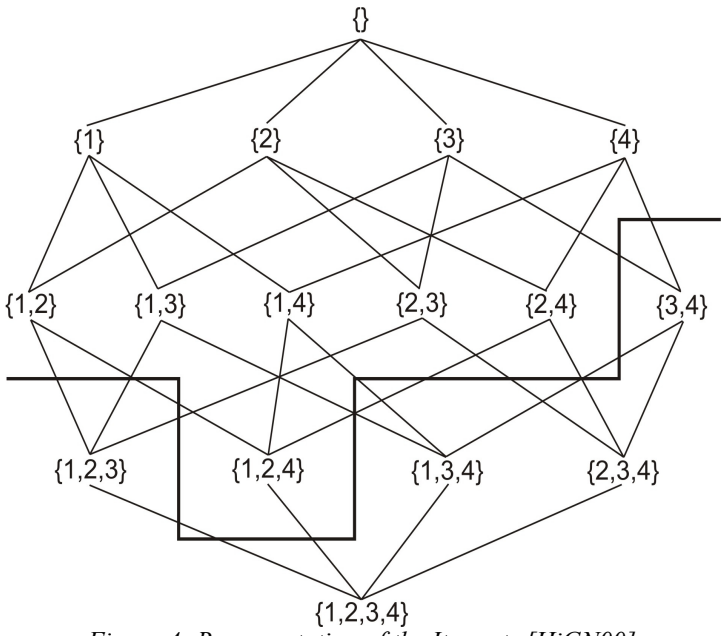

Figure 4: Representation of the Itemsets [HiGN00]

The bold line represents the border between frequent and infrequent itemsets. All items above the border fulfill the minimum support requirements. It is the task of the algorithms to discover the location of this border. In BFS, the support is first determined for all itemsets in a specific level of depth, whereas DFS recursively descends the structure through several depth levels. Thereby, association rule mining algorithms can be systematized as in Figure 5.

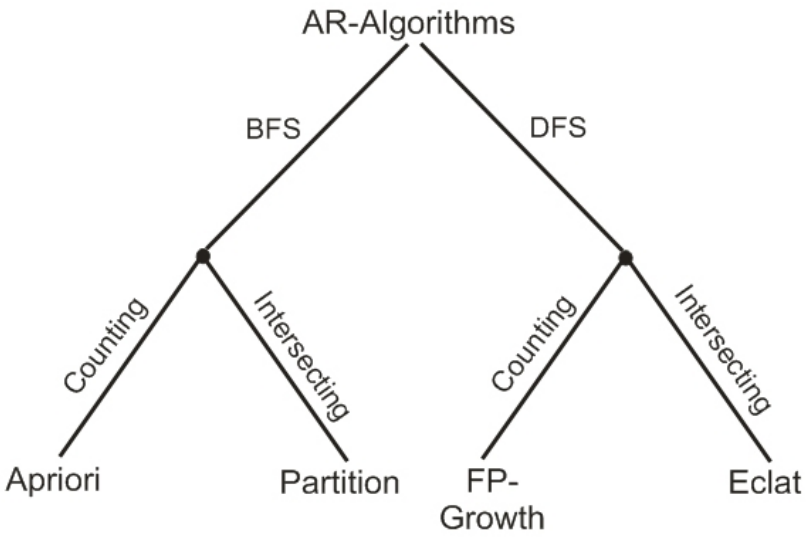

Figure 5: Systematization of Algorithms [HiGN00]

3.4.1 Apriori

The Apriori algorithm was the first attempt to mine association rules from a large dataset. It has been presented in [AgSr94] for the first time. The algorithm can be used for both, finding frequent patterns and also deriving association rules from them. Unlike in [AgIS93], rules having more than one element in the consequent are allowed. We will call such rules multi-consequent rules.

3.4.1.1 Discovering Frequent Itemsets

Generation of frequent itemsets, also called large sets here, makes use of the fact that any subset of a large itemset must as well be large. The number of items contained in an itemset is called its size, an itemset of size k is called a k-itemset. Within the itemset, the items are kept in lexicographic order. To represent the algorithm, the notation in Table 3 will be used.

k-itemset	An Itemset having k items
L_k	Set of large k-itemsets (those with minimum support). Each member of this set has two fields: i) itemset and ii) support count.
C_k	Set of candidate k-itemsets (potentially large itemsets). Each member of this set has two fields: i) itemset and ii) support count.

Table 3: Notation [AgSr94]

Each itemset has a count field associated with it, storing the support value. The pseudocode of the Apriori algorithm is given in Table 4. Firstly, the database is passed over in order to count the occurrences of single elements. If a single element has a support value that is below the defined minimum support, it does not have to be considered anymore because it hence can never be part of a large itemset. A subsequent pass k consists of two phases:

1. The discovered large itemsets of pass $k-1$, i.e. the sets L_{k-1}, are used to generate the candidate itemsets, C_k for the current pass.
2. The database is scanned once more in order to determine the support for the candidate itemsets C_k. If the support is above the minimum support, the candidates will be added to the large itemsets. Discovering the right candidates is crucial in order to prevent a long counting duration.

```
1)  L1 = {large 1-itemsets};
2)  for ( k=2 ; Lk-1 ≠ Ø ; k++ ) do begin
3)    Ck apriori-gen( Lk-1 ); // New candidates
4)    forall transactions t∈D do begin
5)       Ct = subset(Ck,t) ; // Candidates contained in t
6)       forall candidates c∈Ct do
7)          c.count++;
8)    end
9)    Lk = { c∈Ck | c.count ≥ minsup}
10) end
11) Answer = UkLk ;
```

Table 4: Apriori Algorithm [AgSr94]

The `apriori-gen` function takes the large itemsets of the previous iteration as an input. These itemsets are joined together, forming itemsets with one more item than in the step before. After that, a prune step will remove any itemsets whose subcombinations have not been part of the discovered sets in former iterations. The candidate sets are being stored in a hash-tree. This tree can either contain a list of itemsets, which is a leaf node, or a hash table, which is an interior node. The nodes contain the candidate itemset themselves. The `subset` function starts from the root node going towards the leaf nodes in order to find all the candidates contained in a transaction t. The itemsets starting with an item that is not contained in t will therefore be ignored by the function.

3.4.1.2 Discovering Association Rules

As stated before, association rules are allowed to have multiple elements in the antecedent as well as in the consequent. Only large itemsets are used to generate the association rules. The procedure starts with finding all possible subsets of the large itemset l. For each of those subsets, a rule is setup in the form $a \rightarrow (l-a)$. If the confidence of the rule is as least as big as the user-defined minimum confidence, the rule is considered to be interesting. All subsets of l are explored in order not to miss any possible dependencies. But, if a subset a of l does not generate an interesting rule, the subsets of a do not have to be explored. This will save computation power that would otherwise be wasted.

3.4.2 Frequent Pattern Growth (FP-Growth)

The FP-Growth algorithm allows generating frequent itemsets and tries to avoid generating a huge amount of candidates that is necessary for the Apriori algorithm. The heart of this algorithm is a compact representation of the original data set without losing any information. This is achieved by organizing the data in a tree form, called the Frequent Pattern Tree, FP-Tree in short. The approach evolved out of the belief that the bottleneck of Apriori-like algorithms is the candidate-generation and -testing. The FP-Growth algorithm has been introduced in [HaPY99]. The algorithm first constructs the tree out of the original data set and then grows the frequent patterns. For a faster execution, the data should be preprocessed before applying the algorithm.

3.4.2.1 Preprocessing the Data

The FP-Growth algorithm needs the following preprocessing in order to be efficient: An initial scan over the dataset computes the support of the single items. As items that have themselves a support value below the minimum support can never be part of a frequent itemset, they can be discarded from the transactions [Borg05]. The remaining items are recombined so that they appear in a decreasing order with respect to their support. The algorithm will work just fine without sorting the dataset, but it will perform much faster after doing so. With an ascending order, the algorithm performs even worse than using a random order. Table 5 gives an example of how a transaction database will be preprocessed for the FP-Growth algorithm.

Original DB		***Preprocessed DB***
abd		*bda*
bcde	supp(*b*) = 6	*bde*
bd	supp(*d*) = 5	*bd*
ade	supp(*e*) = 5	*dea*
ab	supp(*a*) = 4	*ba*
abe	(supp(*c*) = 2)	*bea*
cde		*de*
be	minsupp = 3	*be*

Table 5: FP-Growth Preprocessing

3.4.2.2 Constructing the FP-Tree

After having preprocessed the data, an FP-Tree can directly be constructed. A scan over the database has to be made, adding each itemset to the tree. The first itemset will be the first branch of the tree. In the transaction database of Table 5, the first branch of the tree would be the items *b*, *d* and *a.* The second transaction shares a common prefix with the already existing set in the tree. In this case, the values along the path of the common prefix will be increased by one, and the remaining items will make new nodes for the tree. In our example, only one new node for *e* will be created. It is simply linked as a child of its ancestor. The tree corresponding to the transaction database of Table 5 is shown in Figure 6. It represents the database in a compact format without the loss of any information.

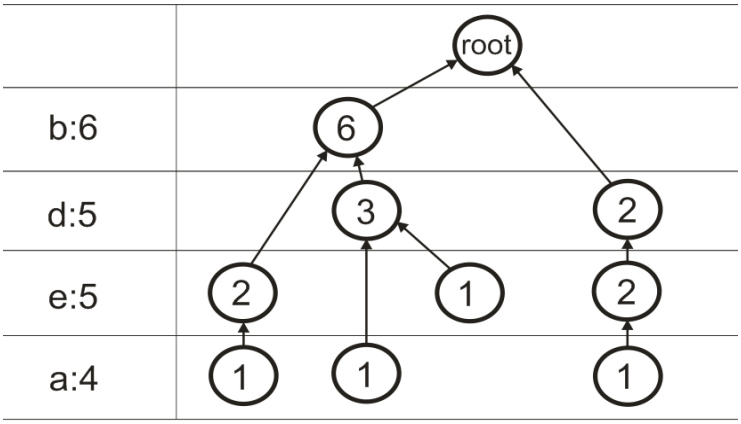

Figure 6: FP-Tree

Each node of the FP-Tree consists of three fields [HaPY99]:

- *item-name*: In this field, the name of the item that the node represents is stored.
- *count*: The field *count* represents the accumulated support of the node within the current path.
- *node-link*: In order to build the structure of the tree, links have to be built between the nodes. The field *node-link* stores the ancestor of the current node, and null if there is none.

Having that done, mining the database is not necessary anymore, now the FP-Tree is used for mining. The support of an itemset can easily be determined by following the path and using the minimum value of *count* from the nodes. For example, the support of itemset $\{b, e\}$ would be 2, whereas the support of itemset $\{b, e, a\}$ would only be 1.

3.4.2.3 Mining the FP-Tree using FP-Growth

The FP-Tree provides an efficient structure for mining, although the combinatorial problem of mining frequent patterns still has to be solved. For discovering all frequent itemsets, the FP-Growth algorithm takes a look at each level of depth of the tree starting from the bottom and generating all possible itemsets that include nodes in that specific level. After having mined the frequent patterns for every level, they are stored in the complete set of frequent patterns. The procedure of the algorithm can bee seen in Table 6.

```
Procedure FP-Growth ( Tree, α ) {
1) if Tree contains a single path P
2) then for each combination (denoted as β )
    of the nodes in the path P do
3) generate pattern β∪α with support =
     minimum support of nodes in β ;
4) else for each a_i in the header of Tree do {
5) generate pattern β=a_i∪α with
     support=a_i.support ;
6) construct β 's conditional pattern base and
    then β 's conditional FP-Tree Tree_β ;
7) if Tree_β≠Ø
8) then call FP-Growth( Tree_β, β ) }
}
```

Table 6: FP-Growth Algorithm ***[HaPY99]***

FP-Growth takes place at each of these levels. To find all the itemsets involving a level of depth, the tree is first checked for the number of paths it has. If it is a single path tree, all possible combinations of the items in it will be generated and added to the frequent itemsets if they meet the minimum support.

If the tree contains more than one path, the conditional pattern base for the specific depth is constructed. Looking at depth a in the FP-Tree of Figure 6, the conditional pattern base will consist of the following itemsets: $\langle b, e:1\rangle$, $\langle b, d:1\rangle$ and $\langle d, e:1\rangle$. The itemset is obtained by simply following each path of a upwards. Table 7 shows the conditional pattern bases for all depth levels of the tree.

item	conditional pattern base
a	$\{\langle b, e:1\rangle, \langle b, d:1\rangle\langle d, e:1\rangle\}$
e	$\{\langle b:2\rangle, \langle b, d:1\rangle\langle d:1\rangle\}$
d	$\{\langle b:3\rangle\}$
b	$\emptyset$

Table 7: Conditional Pattern Bases

From the itemsets in the conditional pattern base, a so called conditional FP-Tree is constructed. This works in the same way as the construction of the initial tree, using the conditional pattern base as the transaction database. After constructing the conditional FP-Tree, the FP-Growth function is called again, mak-

ing it a recursive function. The function is called until the tree contains only one single path or is empty. All the itemsets found in the various conditional FP-Trees are stored and returned in the end as a list of all frequent itemsets in the FP-Tree and also in the database, respectively.

4 Fuzzy Set Theory

In the mathematical sense, a set is a collection of different items that, in a certain way, belong together. Georg Cantor, who was a main inventor of the set theory, defined a set as follows [Cant95]: “By a set we mean an aggregation M of certain unequal objects m in our opinion or in our thought (which are called “elements” of M) to a whole.” Such crisp sets do not always satisfy the needs of real world applications, because they only allow a membership of 1 or 0, i.e. member or non-member. In the real world, it is not at all times possible to assign an object clearly to a certain group of objects. Rather, it might lie in between two different sets.

As an example, we could think of the horsepowers (hp) of a car. Let us assume that we want to divide the horsepowers into the three categories weak, medium and strong. Using crisp sets, we could state that, for example, a car below 100 hp is weak, a car between 100 and 200 hp is medium and a car over 200 hp is strong. The problem that arises is even though a car with 99 hp is almost as strong as a car with 101 hp, it would be classified significantly lower than the other car. This is called the sharp boundary problem. Fuzzy sets can help overcome this problem by allowing different degrees of membership, not only 1 and 0. Objects can thereby be members of more than one set and therefore give a more realistic view on such data. The major concepts of fuzzy set theory will be introduced in this chapter.

4.1 Crisp and Fuzzy Sets

4.1.1 Crisp Sets

Crisp sets have a well defined universe of members. A set can either be described by the list method (naming all the members) or by the rule method (properties that all members have to satisfy). Sets are denoted by capital letters, their members by lower-case letters. The list method is denoted as follows:

$$A=\{a_1, a_2, ..., a_n\}$$

For the rule method, we write:

$$B=\{b|b\ has\ properties\ P_1, P_2, ..., P_n\}$$

If the elements of a set are sets themselves, this set is referred to as a family of sets. $\{A_i | i \in I\}$ Defines the family of sets where i an I are called the set identifier and the identification set. The family of sets is also called an indexed set.

Any set A is called a subset of B if every member of A is also a member of B. This is written as $A \subseteq B$. If $A \subseteq B$ and $B \subseteq A$, the two sets contain the same members and thus are equal. Equal sets are denoted by $A = B$, the contrary, namely unequal sets, are written as $A \neq B$. If $A \subseteq B$ and $A \neq B$ are true, this indicates that B contains at least one object that is not a member of A. In this case, A would be called a proper subset of B and written $A \subset B$. The empty set that contains no members is denoted by $\emptyset$.

Elements are assigned to the sets by giving them the values 0 or 1. Every element that shows a 1 is a member of the set. The number of elements that belong to a set is called its cardinality. All sets that have been created by the rule method might contain an infinite number of elements.

The set containing all members of set *B* that are not members of set A is called the relative complement of A with respect to set B, written $B - A$. If set B is the universal set, the compliment is absolute and denoted by $\bar{A}$.

The union of two sets A and B is a set containing all elements that are in A or in B, denoted by $A \cup B$, whereas their intersection is a set that contains only those elements which are members of A and B, denoted by $A \cap B$. Two elements are disjoint if they do not have any elements in common, that means, if $A \cap B = \emptyset$. A collection of disjoint subsets of A is called a partition on A if the union of those subsets makes the original set A. The partition is denoted by the symbol $\pi(A)$, formally $\pi(A) = \{A_i | i \in I, A_i \subseteq A\}$.

All of the operations union, intersection and complement apply to several rules. At first, union and intersection are commutative, that means that the order of the operands does not affect the result:

$A \cup B = B \cup A$, $A \cap B = B \cap A$.

The second rule, called associativity, states that union and intersection can be applied pairwise in any order without changing the result:

$A \cup B \cup C = (A \cup B) \cup C = A \cup (B \cup C)$, $A \cap B \cap C = (A \cap B) \cap C = A \cap (B \cap C)$.

Union and intersection both are idempotent operations because applying any of those operations on a set with itself will give the same set:

$A \cup A = A$, $A \cap A = A$.

The distributive law is satisfied for both union and intersection in the following way: $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$ $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$.

DeMorgan's law constitutes that the complement of the union of two sets matches the union of their complements:

$\overline{A \cap B} = \overline{A} \cup \overline{B}$, $\overline{A \cup B} = \overline{A} \cap \overline{B}$.

For further information, see [KlFo88].

4.1.2 Fuzzy Sets

Fuzzy sets can generally be viewed as an extension of the classical crisp sets. They have been first introduced by Lofti A. Zadeh in 1965 [Zade65].

> “Fuzzy sets are generalized sets which allow for a graded membership of their elements. Usually the real unit interval [0; 1] is chosen as the membership degree structure.” [Gott06]

Crisp sets are discriminating between members and nonmembers of a set by assigning 0 or 1 to each object of the universal set. Fuzzy sets generalize this function by assigning values that fall in a specified range, typically 0 to 1, to the elements. This evolved out of the attempt to build a mathematical model which can display the vague colloquial language. Fuzzy sets have proofed to be useful in many areas where the colloquial language is of influence. Let X be the universal set. The function μ_A is the membership function which defines set A. Formally: $\mu_A : X \rightarrow [0,1]$.

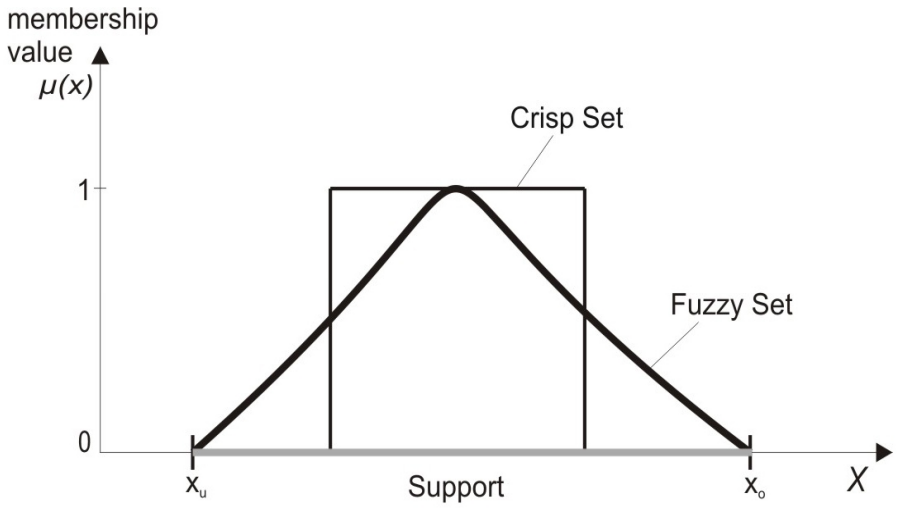

Figure 7: Fuzzy Set

In Figure 7, we see a graph of a crisp set and a fuzzy set. The fuzzy set can look very different depending on the chosen membership function. Using this function, it is possible to assign a membership degree to each of the elements in X. Elements of the set could but are not required to be numbers as long as a degree of membership can be deduced from them. For the purpose of mining fuzzy association rules, numeric elements are used for quantitative data, but other categories might also exist where no numerical elements will be found (e.g. something is a fruit or a vegetable). It is important to note the fact that membership grades are not probabilities. One important difference is that the summation of probabilities on a finite universal set must equal 1, while there is no such requirement for membership grades [KlFo88].

For many applications, this type of fuzzy sets will suffice. Although, it seems paradoxical that the measures used to represent fuzziness are themselves precise real numbers. Therefore, an extension to the concept of fuzzy sets has been developed, allowing membership grades to become blurred (see Figure 8).

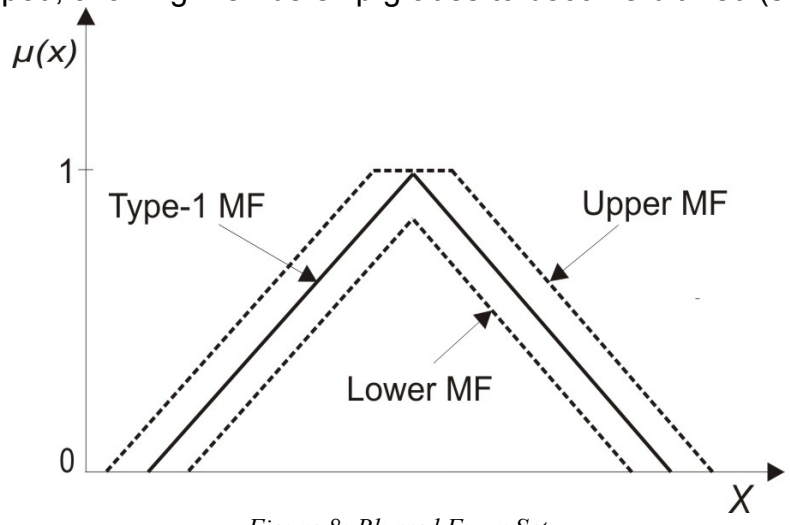

Figure 8: Blurred Fuzzy Set

These special sets are called type 2 fuzzy sets, whereas, in this context, ordinary fuzzy sets are named type 1 fuzzy sets. The membership grades of a type 2 fuzzy set are themselves type one fuzzy sets. If the membership grades of a set are type 2 fuzzy sets, the resulting set will be a type 3 fuzzy sets. Using this method, even higher grades of fuzzy sets can be defined.

4.1.2.1 Concepts

The following concepts are important when dealing with fuzzy sets [KlFo88]:

- Support: The support of a fuzzy set A is given by a crisp set that contains all of the elements whose membership degree in A is not 0: $supp(A)=\{x \in X | \mu_A(x) > 0\}$ .The empty fuzzy set has an empty support set.
- Height: The height of a fuzzy set is defined by the largest membership value attainable by an element of the set. The fuzzy set is called normalized if at least one of its elements attains the highest membership grade.
- α-cut: The α-cut of a fuzzy set is defined by a crisp set A_α containing all elements that have a membership grade to the fuzzy set that is greater than 0.
- Scalar cardinality: The summation of the membership grades of all elements in a fuzzy set is called its scalar cardinality.

4.2 Fuzzy Logic

“Fuzzy logic may be viewed as an extension of multivalued logic. Its uses and objectives, however, are quite different. Thus, the fact that fuzzy logic deals with approximate rather than precise modes of reasoning implies that, in general, the chains of reasoning in fuzzy logic are short in length and rigor does not play as important a role as it does in classical logical systems. In a nutshell, in fuzzy logic everything, including truth, is a matter of degree.” [Zade88]

Fuzzy logic deduces its greater expressive power from including probability theory and probabilistic logic. According to [Zade88], the main differences between traditional logic and fuzzy logic are the following:

- Speaking of two-valued logic, a proposition is either true or false. Fuzzy logic allows truth values to range over fuzzy subsets. Therefore, the fuzzy

truth value could be viewed as an imprecise characterization of a numerical truth value.

- Fuzzy logic allows crisp predicates, as in two-valued logic, but also fuzzy ones, for example “big”, “tall” or “beautiful”.
- Two-valued logic allows only the two quantifiers “all” and “some” whereas fuzzy logic allows the use of quantifiers like “most”, “many”, “several”, “few” and so on. These fuzzy quantifiers can be viewed as a second order fuzzy predicate.
- Both fuzzy and and non-fuzzy predicate-modifiers can be represented by fuzzy logic. This leads to a system which enables computing with linguistic variables, i.e. variables whose values are words and expressions from a natural or synthetic language.
- In two-valued logic, a proposition can be qualified by associating it with a truth value (“true” or “false”), a modal operator (such as “possible” or “necessary”) “or an intensional operator (such as “know” or “believe”).
  Fuzzy logic proposes three different types of qualification:
  - Truth-qualification, for example “not quite true”.
  - Probability-qualification, something is “unlikely”.
  - Possibility-qualification, might be expressed by “almost impossible”.

As an example, we will consider two proportions [Zade96]:

- p_1 = *Carol lives near Mary*
- p_2 = *Mary live near Pat*

In this case, the phrase “lives near” is a fuzzy constraint. To the question “How far does Carol live from Pat?”, we can now give an answer. This answer could look like the proportion p_3 = *Carol lives not far from Pat*.

It is obvious that imprecise concepts (as shown above) can be modeled using fuzzy logic. To demonstrate how fuzzy logic works, an extension of the classical two-valued logic to a three-valued one is necessary. Besides the values 1 and 0, we will now add 0.5 to express indeterminacy. To evaluate the primitives $\vee, \wedge, \Rightarrow, \Leftrightarrow$ known from two-valued logic, several logics have been developed,

with the results differing from each other. Some of the most frequently used logics are presented in Table 8.

		Łukasiewicz				Bochvar				Kleene			
a	b	∧	∨	⇒	⇐	∧	∨	⇒	⇐	∧	∨	⇒	⇐
0	0	0	0	1	1	0	0	1	1	0	0	1	1
0	0.5	0	0.5	1	0.5	0.5	0.5	0.5	0.5	0	0.5	1	0.5
0	1	0	1	1	0	0	1	1	0	0	1	1	0
0.5	0	0	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0	0.5	0.5	0.5
0.5	0.5	0.5	0.5	1	1	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
0.5	1	0.5	1	1	0.5	0.5	0.5	0.5	0.5	0.5	1	1	0.5
1	0	0	1	0	0	0	1	0	0	0	1	0	0
1	0.5	0.5	1	0.5	0.5	0.5	0.5	0.5	0.5	0.5	1	0.5	0.5
1	1	1	1	1	1	1	1	1	1	1	1	1	1

Table 8: Three-Valued Logics

The logics differ from each other in the treatment of the new truth value 0.5. The three-valued logics can be generalized to n-valued logics [KlFo88]. For such logics, the degree of truth is usually labeled by rational numbers in the interval [0, 1]. The fist person to introduce a n-valued logic was Jan Łukasiewicz in the 1930s. He proposed his logic as a generalization of three-valued logic with the primitives defined as follows:

$$\begin{gathered}\bar{a}=1-a\,,\\ a\wedge b=min(a\,,b)\,,\\ a\vee b=max(a\,,b)\,,\\ a\Rightarrow b=min(1,1+b-a)\,,\\ a\Leftrightarrow b=1-|a-b|.\end{gathered}$$

We will use the Łukasiewicz Logic for the fuzzy logic computations. Such a n-valued logic is often denoted as L_n in literature. Infinite-valued logics are a special case of n-valued logics, where the truth values are taken from all rational numbers of the unit interval [0, 1].

4.3 Fuzzy Operations

To perform operations on fuzzy sets, we need to reconfigure our operations for crisp sets. Generally, the same operations can be applied for fuzzy sets which are complement, union and intersection. According to the concepts of fuzzy logic, the operations will be defined in the rest of this section.

- **Complement**

  The difference between the complement for a crisp sets and the fuzzy fuzzy complement is, that there can be elements that have nonzero values for both, the fuzzy set and its complement. It is clear that the complement for the values 0 and 1 behave in the same way as they do for crisp sets, that is the complement of 0 is 1 and the complement of 1 is 0. Formally: $c(0)=1, c(1)=0$.

  We also assume that the function c (the fuzzy complement) is monotonic nonincreasing, meaning for all $a\,,b\in[0,1]$, $if\ a<b$, $then\ \ c(a)\geq c(b)$.

  The fuzzy complement is defined as follows: $\mu_{\bar{A}}(x)=1-\mu_A(x)$. The complement value of a membership grade of 0.4 would thereby be 0.6 (see Figure 9).

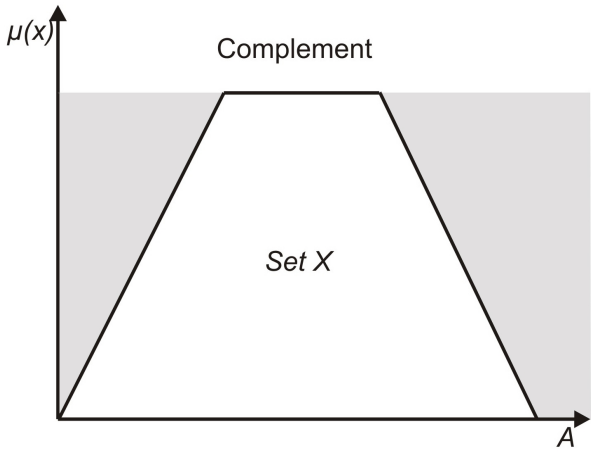

Figure 9: Fuzzy Complement

- **Union**

  The union of two fuzzy sets can be generally specified by a function of the form: $u:[0,1]\times[0,1]\rightarrow[0,1]$.

  According to $u(0,0)=0; u(0,1)=u(1,0)=u(1,1)=1$, the fuzzy union will give equal results to the crisp union for 0 and 1, but it allows to deal with other values as well. Further characteristics are commutativity ($u(a,b)=u(b,a)$), monotonicity ($if\ a\leq a' \wedge b\leq b',\ then\ \ u(a,b)\leq u(a',b')$), and associativity ($u(u(a,b),c)=u(a,u(b,c))$).

  To compute a fuzzy Union, we use the following form: $\mu_{A\cup B}(x)=max[\mu_A(x),\mu_B(x)]$ (see Figure 10). According to the form, the maximum of both memberships is taken for the union. If we see two membership grades of 0.3 and 0.6, the applied fuzzy union would be 0.6.

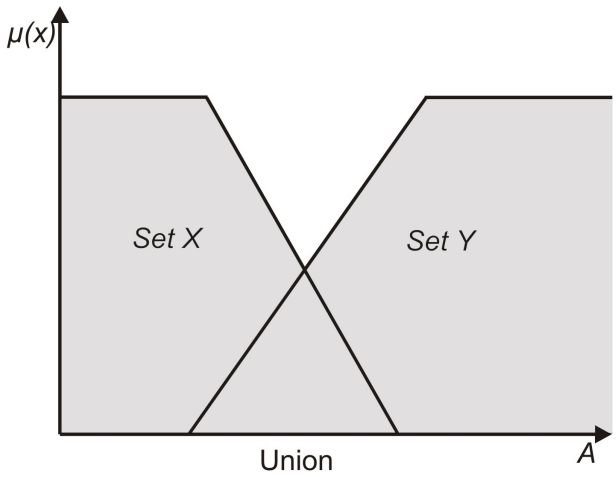

Figure 10: Fuzzy Union

- **Intersection**

  The discussion of the fuzzy intersection is strongly related to the fuzzy union. The same function, namely $i:[0,1]\times[0,1]\rightarrow[0,1]$ specifies the fuzzy intersection.

  Again, the fuzzy intersection works the same way as the classical intersection for binary data, defined by $i(1,1)=1; i(0,1)=i(1,0)=i(1,1)=0$. Also, it is commutative, monotonic and associative as shown for the fuzzy union.

  To compute membership values with fuzzy intersection, the minimum is used: $\mu_{A\cap B}(x)=min[\mu_A(x),\mu_B(x)]$ (see Figure 11). Thus, a fuzzy intersection of 0.3 and 0.6 would give 0.3 as a result.

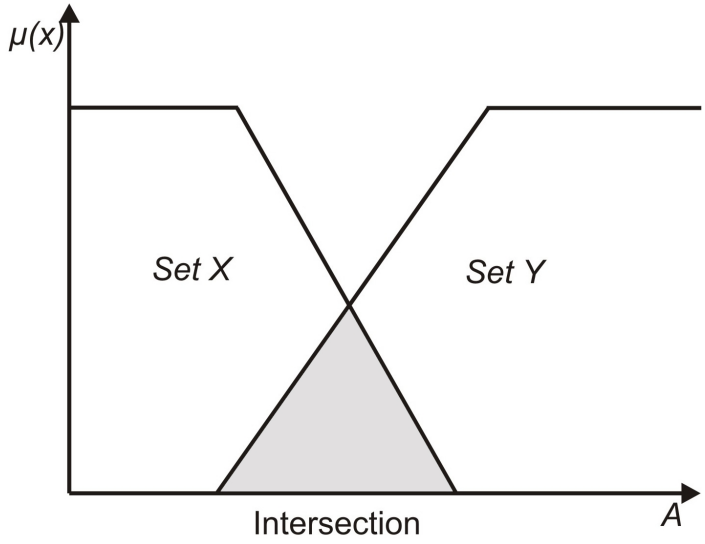

Figure 11: Fuzzy Intersection

4.3.1 Triangular Norms

Triangular norms, in short t-norms, are an important factor in fuzzy association rule mining. “A triangular norm (a t-norm for short) is a commutative, associative, non-decreasing function in $T:[0,1]^2\rightarrow[0,1]$ such that $T(x,1)=x$ for all $x\in[0,1]$. [...] The basic continuous t-norms are the minimum, $T_M(x,y)=min(x,y)$, the product, $T_P(x,y)=xy$ and the Łukasiewicz t-norm, $T_L(x,y)=max(0,x+y-1)$.” [MeNa99]

In literature on fuzzy association rules, a t-norm is often used as a generalization for the intersection of fuzzy sets which has already been described in Chapter 4.3. It satisfies the boundary conditions $T(x,0)=0$ and $T(x,1)=x$ for all $0\leq x\leq 1$. T-norms are often denoted by $x \otimes y$, but we will use the notion $T(x,y)$. It is important to know which t-norms are admissible for conducting a fuzzy intersection. Generally speaking, admissible t-norms can be derived from the n-valued logics, for example the minimum or the product. Therefore, the intersection of two fuzzy sets equals their t-norm: $x \cap y = T(x,y)$. Not all t-norms are admissible, though.

A generalized logical disjunction is represented by a t-conorm, which can be used for defining the union of fuzzy sets. The notion of a t-conorm is $S(x,y)$, although $x \oplus y$ is similarly applied in literature. A t-conorm satisfies all conditions of t-norms, i.e. commutativity, associativity and it is non decreasing. The boundary conditions here are $S(x,0)=x$ and $S(x,1)=1$ for all $0\leq x\leq 1$. Usual choices for t-conorms are the maximum or the algebraic sum. As stated above, t-conorms are used to define the union of fuzzy sets: $x \cup y = S(x,y)$. Usual choices for t-norms and t-conorms can be found in Table 9.

t-norm	**t-conorm**
$T_M(x,y)=min(x,y)$	$S_M(x,y)=max(x,y)$
$T_P(x,y)=xy$	$S_P(x,y)=x+y-xy$
$T_W(x,y)=max(x+y-1,0)$	$S_W(x,y)=min(x+y,1)$

Table 9: Well-known t-norms and t-conorms [CoCK03]

An important issue here is to define which t-norms are admissible for use with fuzzy sets. According to [DuHP06], a t-norm is admissible if it is a copula. A copula is a “function that joins or couples multivariate distribution functions to their one-dimensional marginal distribution functions.” (cp. [Nels06]) For further information on triangular norm based fuzzy logics, see [BuKZ95].

5 Fuzzy Association Rules

Based on classical association rule mining, a new approach has been developed expanding it by using fuzzy sets. The new fuzzy association rule mining approach emerged out of the necessity to mine quantitative data frequently present in databases efficiently. Algorithms for mining quantitative association rules have already been proposed (see chapter 3.3). When dividing an attribute in the data into sets covering certain ranges of values, we are confronted with the sharp boundary problem.

Figure 12: Crisp Set

Elements near the boundaries of a crisp set will either be ignored or overemphasized [KuFW98]. For example, one can consider a set representing persons of middle age, ranging from 30 to 50 years old (see Figure 12). In this example, a person aged 29 years would be a 0% representative and a 31 year old would be 100%. In reality, the difference between those ages is not that great. Implementing fuzziness can overcome this problem.

The same problem can occur if one is dealing with categorical data. Sometimes, it is not ultimately possible to assign an item to a category. As an example, one can say that a tomato is a vegetable but also, in a way, a fruit. Crisp sets would only allow assigning the item to one single category, fuzzy sets allow different grades of membership to more than one set. Three different approaches to fuzzy association rules can be found in literature which will be discussed in

the following chapters: The quantitative approach, fuzzy taxonomic structures and the approximate itemset approach.

5.1 Approaches

5.1.1 Quantitative approach

Kuok et al. descibe fuzzy association rules as follows [KuFW98]: “Mining fuzzy association rule is the discovery of association rules using fuzzy set concepts such that the quantitative attribute can be handled” As in classical association rules, $I=\{i_1, i_2, ..., i_m\}$ represents all the attributes appearing in the transaction database $T=\{t_1, t_2, ..., t_n\}$. I contains all the possible items of a database, different combinations of those items are called itemsets. Each attribute i_k will associate with several fuzzy sets. In order to represent the fuzzy sets associated with i_k, we use the notion $F_{i_k}=\{f_{i_k}^1, f_{i_k}^2, ..., f_{i_k}^l\}$ where $f_{i_k}^j$ is the j^{th} fuzzy set in F_{i_k}. As an example, the attribute salary could look as follows: $F_{Salary}=\{high, medium, low\}$. Fuzzy sets and their corresponding membership functions have to be defined by domain experts. Each of the fuzzy sets can be viewed as a [0,1] valued attribute, called fuzzy attribute.

A fuzzy association rule has the following form:

$$If\ X\ is\ A\ then\ Y\ is\ B$$

In this case, $X=\{x_1, x_2, ..., x_p\}$ and $Y=\{y_1, y_2, ..., y_q\}$ are itemsets which are subsets of I. It is important to notice that those two sets must be disjoint and thus do not have any attributes in common. $A=\{f_{x_1}, f_{x_2}, ..., f_{x_p}\}$ and $B=\{f_{y_1}, f_{y_2}, ..., f_{y_q}\}$ contain the fuzzy sets that are associated with X and Y. Known from classical association rules, $X\ is\ A$ is the antecedent, $Y\ is\ B$ is the consequent. If a sufficient amount of records approves this rule, we will call it satisfied.

For example, we might want to partition the variable *Age* into three fuzzy sets. The fuzzy sets and their membership functions will have to be defined by a domain expert. For easy demonstration, we will just define the borders of the sets and split the overlapping part equally between the so generated fuzzy sets. For an example, we will use the following borders for the fuzzy sets of the variable

age: $Age.Low=\{0-33\}, Age.Medium=\{27-55\}, Age.High=\{48-\infty\}$. The generated fuzzy sets is shown in Figure 13. For all areas having no overlap of the sets, the support will simply be 1 for the actual itemset. If there is an overlap, the membership can be computed by using the borders of the overlapping fuzzy sets. The added support will here always sum up to 1.

The formula for computing the membership varies depending on whether it is at the upper border of a set or at the lower border:

$\mu(x)=\frac{hb(f_{i_k}^n)-x}{hb(f_{i_k}^n)-lb(f_{i_k}^{n+1})}$ for the computation of membership at the high border,

$\mu(x)=\frac{x-lb(f_{i_k}^n)}{hb(f_{i_k}^n)-lb(f_{i_k}^{n+1})}$ for the lower border,

Where $lb(f_{i_k}^n)$ is the low border of a set, $hb(f_{i_k}^n)$ is the high border and x is the original value of any attribute in the database.

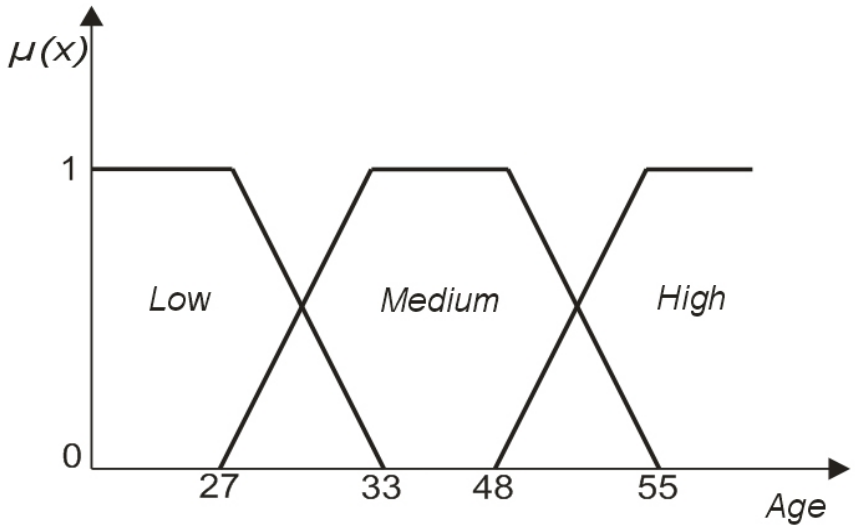

Figure 13: Fuzzy Partition of a Quantitative Attribute

5.1.1.1 Fuzzy normalization

When we are dealing with quantitative attributes mapped to fuzzy sets we might, depending on the membership function, find that the membership values to the sets of one single entity does not add up to one [Gyen00]. This depends on how we defined our fuzzy sets and the corresponding membership functions in advance. If we are dealing with a mix of quantitative and categorical attributes, we

might find it unreasonable that the quantitative attribute has the potential to contribute more to a rule than a categorical one. The entry in a database in Table 10 serves as an example.

Level:junior	Level:senior	Age:low	Age:medium	Age:high
0	1	0.2	0.9	0.1

Table 10: Without Fuzzy Normalization

The categorical attribute *Level* can have any number of attributes, and still only one of them will contribute with the value 1, all of the others will show 0. The quantitative attribute *Age* represented by fuzzy sets, however, contributes with 1.2 in this case. It is unreasonable for one transaction to contribute more than others. Here, the fuzzy normalization process takes place. It will further transform the transaction to values of age that sum up to 1. The new values can be calculated easily by dividing the value of a single element by the sum of all the fuzzy values corresponding to that attribute (see attribute *Age* in .Table 11).

Level:junior	Level:senior	Age:low	Age:medium	Age:high
0	1	0.167	0.75	0.083

Table 11: With Fuzzy Normalization

5.1.2 Fuzzy Taxonomic Structures

Similarly, the approach illustrated in chapter 5.1.1 can also be used when dealing with taxonomic structures that are not crisp but fuzzy. A taxonomy is a user-defined categorization of the available items [GrKW01]. It is a hierarchy represented by a tree where a child node belongs to one single parent node. In some cases, we might need to allow a graded membership to more than one parent nodes. As an example, a tomato could be regarded as both, a fruit and a vegetable. In order to demonstrate such dependencies, the concept of fuzzy taxonomic structures is introduced in [WeCh99] (for an example, see Figure 14).

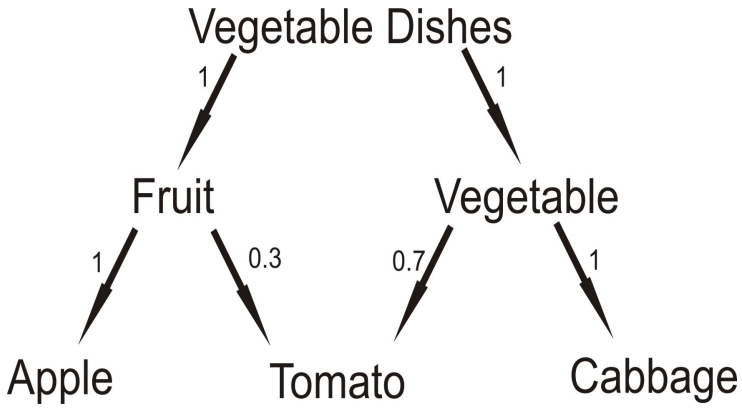

Figure 14: A Fuzzy Taxonomic Structure [WeCh99]

In crisp taxonomic structures, a child belongs to its ancestor with degree one. That means, any child can only have one single ancestor. Fuzzy taxonomies eliminate this assumption by allowing graded memberships to different parent nodes at the same time. Each child node belongs to its parent nodes by a certain degree μ, where $0 \leq \mu \leq 1$. Any node x is called an ancestor of node y if a direct path exists from x to y. Node y is then called the descendant of node x.

The values within the tree are no fuzzy sets and there is no membership function for them. Instead, the structures have to be defined a priori by domain experts. After definition, normal procedures of mining fuzzy association rules can be used to mine data that is organized in the form of a fuzzy taxonomic structure. With this procedure, even categorical data can be enriched with more information and used for fuzzy association rule mining.

5.1.3 Approximate Itemset Approach

The task of mining frequent itemsets deals with discovering frequent episodes in a sequence of events [MaTV97]. The data can be viewed as a sequence of events associated with a certain time of occurrence. Speaking of a frequent episode, we mean a collection of events that frequently occurs jointly. A basic problem of the analysis of such episodes is to discover them in the first place. Episodes are partially ordered sets of events. Looking at these episodes enables to discover regularities, for example an event X is followed by an event Y in most of the cases. The crucial problem here is the definition of how close

together two items have to be in a timely manner in order to qualify as an episode. A time window has to be defined for this purpose.

If we are dealing with data where the items are frequently delayed or lost, problems will occur in discovering episodes. In [WaBK05], an approach of mining fuzzy frequent itemsets is proposed for the purpose of dealing with such situations. In this context, the term “fuzzy” refers to a set of items that may not be discovered exactly in the supporting transaction, but only approximately. The traditional approach would potentially discard a possibly interesting episode just because it does not satisfy the user-specified minimum support measure. This happens because the transaction simply occurs too rarely. The task now is to allow a certain number of mismatches to account for items that were possibly lost or delayed.

5.2 Quality Measures

In order to enable the evaluation of a fuzzy association rule, we use the standard approach for calculating support and confidence, replacing the set-theoretic operations by the corresponding fuzzy set-theoretic operations [DuHP03]:

$$supp(A \rightarrow B) = \sum_{(x) \in D} T(A(x), B(x))$$

$$conf(A \rightarrow B) = \frac{\sum_{(x) \in D} T(A(x), B(y))}{\sum_{(x,y) \in D} A(x)}$$

The usual choice for the t-norm is the minimum as demonstrated in chapter 4.3.1, yet the product has also been applied [DuHP06]. The rule $A \rightarrow B$ can equally be displayed by summing up the individual supports that are provided by the tuples $(x, y) \in D$:

$$supp_{[x,y]}(A \rightarrow B) = T(A(x), B(y))$$

Additionally, if A supports B, B will automatically also support A. This is due to the fact that the support is computed by simply summing up the memberships of the different items in the database. Therefore:

$$supp_{[x,y]}(A \rightarrow B) = supp_{[x,y]}(B \rightarrow A)$$

The support measure is especially important to determine frequent itemsets with respect to the user-defined minimum support, just as in binary association rules. The confidence is particularly used for investigating the interestingness of the discovered rules. A rule will only be interesting if its confidence is above the specified minimum, and it becomes more interesting the bigger the support is.

Some research on fuzzy association rules stresses the importance of two-sidedness of knowledge when evaluating the rules [CoCK03]. A whole new spectrum of knowledge can be expressed by complementing the degree of membership with a degree of non-membership. Therefore, we have to distinguish between “not positive examples” and “negative examples” of the rules. The support of a rule consists of the elements belonging to both the antecedent (A) and the consequent (B). Equally, these same elements defining the support can also be viewed as the positive examples of the particular rule. Being positive examples of the rule $A \rightarrow B$, the same elements are also positive examples of the rule $B \rightarrow A$.

Now, we have to define what a negative example of a rule might look like. Looking at the rule $A \rightarrow B$ again, a negative example would be a tuple where A is satisfied but B is not. This time, $A \rightarrow B$ does not have the same negative examples as $B \rightarrow A$. Also, we have to distinguish between a negative example and a non-positive example. In a non-positive example, even the antecedent A is not satisfied.

The opposition of a fuzzy rule can best be described as the opposite of its support. While the support sums up all the values that an itemset shows for the different variables, the opposition accounts for the fuzzy complements of these values. See Table 12 for support and opposition values. In this context, coA stands for the complement of a fuzzy set A .

	$A \rightarrow B$
minimum support (minsupp)	$\lvert A \cap B \rvert$
maximum opposition (maxopp)	$\lvert coA \cup coB \rvert$
minimum opposition (minopp)	$\lvert A \cap coB \rvert$
maximum support (maxsupp)	$\lvert coA \cup B \rvert$

Table 12: Measures

After validating the support of a rule, its confidence can be explored. In [CoCK03], so called pessimistic and optimistic confidence measures are introduced. Formally, those measures look as follows:

$$conf_p(A \rightarrow B) = \frac{minsupp(A \rightarrow B)}{maxopp(A \rightarrow B)}$$

$$conf_o(A \rightarrow B) = \frac{maxsupp(A \rightarrow B)}{minopp(A \rightarrow B)}$$

"When determining the pessimistic confidence of a rule $bread \rightarrow butter$ we have the following assumption in mind: if those people who did not buy bread, would have bought bread, they would not have bought butter as well. For the optimistic confidence measure on the other hand we assume that if those people who did not buy bread, would have bought bread, they would have bought butter as well." [CoCK03]

5.2.1 Problems

Some problems arise when computing the confidence of a rule [DuPS03], if the sum is taken over the set of tuples in the database. As a result, a large number of tuples with small membership grades is permitted to have the same effect as one single element with membership grade one. This may lead to unintuitive results after the assessment of the association rules. As an alternative, one could think of giving greater significance to those tuples with higher membership degrees. The proposed solution is the use of a scalar cardinality of fuzzy sets based on the weighted summation of the cardinalities of its α-cuts. The new confidence measure then looks as follows:

$$conf(A \rightarrow B) = \sum_{i=1}^{t-1} (\alpha_i - \alpha_{i+1}) \frac{|(A \cap B)|_{\alpha_i}}{|A|_{\alpha_i}}$$

This methods puts greater emphasis on elements with higher membership degrees due to the fact that an element with membership α_k occurs in each summand of $k, k+1, \ldots, t$.

As described before, the minimum is the most common choice for a t-norm representing the fuzzy intersection. This occurs together with a loss of information. If the membership grades of two items differ only in the same set where both elements have a higher grade, the difference will be lost [DuPS05]. A com-

pensatory t-norm might do a better job for this kind of data. The database of table Table 13 serves as an example. Even though the tuples differ in their membership values, the minimum will still produce the same result.

ID	A(a)	B(b)	min(A(a), B(b))
1	1	0.1	0.1
2	0.1	1	0.1
2	0.1	0.1	0.1

Table 13: Example Membership Table

5.3 Discovering Fuzzy Sets

The traditional way to discover the fuzzy sets needed for a certain data set is to consult a domain expert who will define the sets and their membership functions. This requires access to domain knowledge which can be difficult or expensive to acquire. In order to make an automatic discovery of fuzzy sets possible, an approach has been developed which generates fuzzy sets automatically by clustering [FWSY98]. This method can be used to divide quantitative attributes into fuzzy sets, which deals with the problem that it is not always easy do define the sets a priori.

The proposed method uses a known clustering algorithm to find the medoids of k clusters. The whole process of automatically discovering fuzzy sets can be subdivided into four steps:

- Transform the database to make clustering possible (the value of all the attributes has to be positive integer).
- Find the k medoids of the transformed database using a clustering method.
- For each quantitative attribute, fuzzy sets are constructed using the medoids.
- Generate the associated membership functions.

In [FWSY98], the CLARANS algorithm is proposed to conduct the clustering. After discovering k medoids, we can compute k fuzzy sets out of them. We define $\{m_1, m_2, \ldots, m_k\}$ as the k medoids from a database. The i-th medoid can be defined as $m_i = \{a_{i1}, a_{i2}, \ldots, a_{in}\}$. If we want to discover the fuzzy sets for the

j-th attribute, ranging from min_j to max_j, our mid-points will be $\{a_{i1}, a_{i2}, \ldots, a_{in}\}$. The fuzzy sets will then show the following ranges: $\{min_j - a_{2j}\}, \{a_{1j} - a_{3j}\}, \{a_{(i-1)j} - a_{(i+1)j}\}, \ldots, \{a_{(k-1)j} - max_j\}$.

Finally, the membership functions for the fuzzy sets have to be computed. We can get our membership function looking at the definition of the sets above. For the fuzzy set with mid-point a_{kj}, the membership function looks as follows: If $x \leq a_{(k-1)j}$, the membership of x is 0. Also for $x \geq a_{(k+1)j}$, $\mu_x = 0$ because in both cases, the value lies outside the range of the fuzzy set. If x takes exactly the value of the mid-point a_{kj}, the membership is 1. For all other cases, we have to use a formula in order to compute the specific membership:

$$\mu_x = \begin{cases} \dfrac{x - a_{(k-1)j}}{a_{kj} - a_{(k-1)j}} & if \quad a_{(k-1)j} < x < a_{kj} \\ \dfrac{x - a_{(k+1)j}}{a_{kj} - a_{(k+1)j}} & if \quad a_{kj} < x < a_{(k+1)j} \end{cases}$$

A distinction between two types of fuzzy sets has been introduced in [XieD05]. These two types are called equal space fuzzy sets (Figure 15) and equal data points fuzzy sets (Figure 16). Equal space fuzzy sets are symmetrical and all occupy the same range in the universal set. In contrary, equal data points fuzzy sets cover a certain number of instances and thus are not symmetrical.

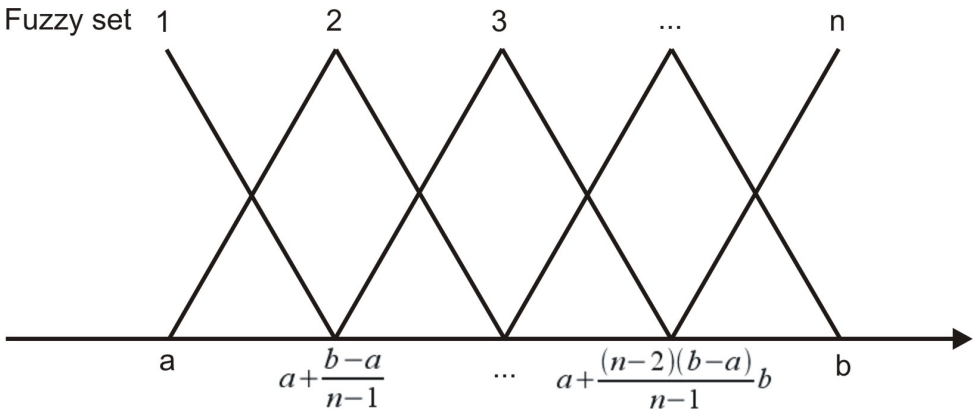

Figure 15: Equal Space Fuzzy Set

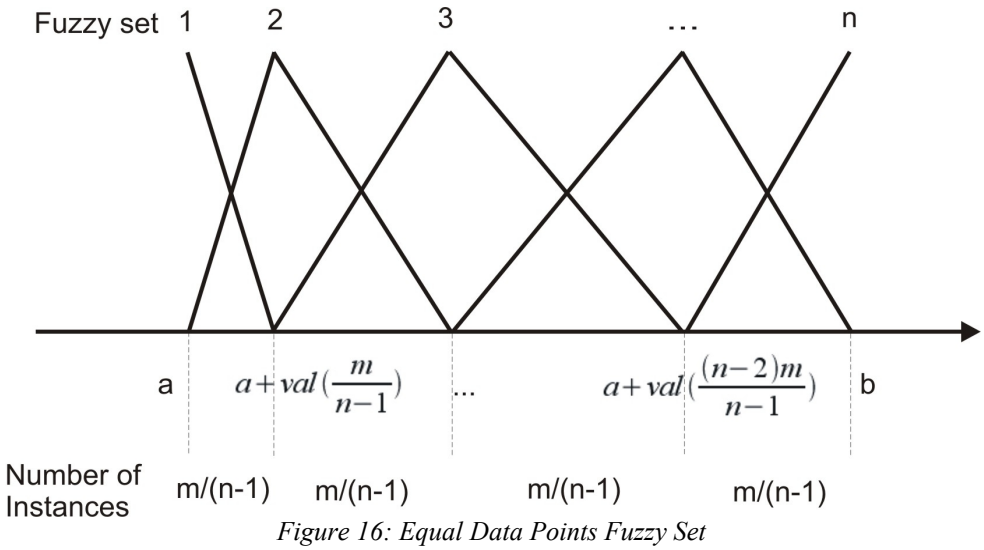

Figure 16: Equal Data Points Fuzzy Set

5.4 Algorithms

Some attempts for developing algorithms to discover fuzzy association rules have already been made. In [ChAu98], an algorithm for mining fuzzy association rules in quantitative databases is proposed. The algorithm, called F-APACS, employs linguistic terms to describe the hidden regularities and exceptions rather than splitting up quantitative attributes into fuzzy sets. The linguistic terms are defined by fuzzy set theory, therefore the association rules discovered here are called fuzzy association rules. An objective interestingness measure is used to define whether attributes are related or not. The use of linguistic terms is an attempt to make rules more understandable for the human user.

In traditional association rule mining techniques, minimum support and confidence thresholds have to be defined by the user. The F-APACS algorithm addresses this problem by using adjusted difference analysis to identify interesting associations between attributes. In addition, the algorithm can discover both, positive and negative association rules. A negative rule tells us that if a record has a certain characteristic, the associated record will not have another characteristic. The algorithm can be found in Table 14.

1) **rules** F-APACS( )
2) **begin**
3) **forall** $d \in \mathcal{D}$ **do**
4) **forall** $\mathcal{L}_{pq}, \mathcal{L}_{jk} \in \mathcal{L}, p \neq j$ **do**
5) $deg_{\mathcal{L}_{pq}\mathcal{L}_{jk}} += \min(\mu_{L_{pq}}(d[I_p]), \mu_{L_{jk}}(d[I_j]))$;
6) **forall** $\mathcal{L}_{pq}, \mathcal{L}_{jk} \in \mathcal{L}, p \neq j$ **do**
7) **if** $interesting(\mathcal{L}_{pq}, \mathcal{L}_{jk})$ **then**
8) $\mathcal{R} = \mathcal{R} \bigcup rulegen(\mathcal{L}_{pq}, \mathcal{L}_{jk})$;
9) **return**($\mathcal{R}$);
10) **end**

Table 14: F-APACS Algorithm

The algorithm starts with a data set. The linguistic terms $\mathcal{L}_{pq}, \mathcal{L}_{jk}$ are represented by fuzzy sets L_{pq}, L_{jk} and the degree to which d is represented by $\mathcal{L}_{pq}, \mathcal{L}_{jk}$ is summarized in $deg_{L_{pq}L_{jk}}$. The interestingness of an association rule is calculated using the adjusted difference measure. For further details on the algorithm, see [ChAu98].

Another algorithm has been suggested in [ChWe02] which is suitable for mining association rules in fuzzy taxonomic structures. The Apriori algorithm is extended to allow mining fuzzy association rules as well. Fuzzy support and confidence measures are applied in order to evaluate the interestingness of a rule. The non-fuzzy algorithm of [SrAg95] decides whether a transaction T supports an itemset X by checking for each item $x \in X$ if the item itself or some descendant of it is present in the transaction. For this reason, all possible ancestors of each item in T are added, forming T'. Now T supports X if and only if T' is a superset of X. A standard algorithm can then be run on the extended transactions to mine the association rules. In the fuzzy case, T' is generated differently. Not only the ancestors of T have to be added, but also the degree to which the ancestors are supported by the transactions.

A different attempt has been made in [HoKC99] which similarly uses the Apriori algorithm as a basis but incorporates fuzzy sets for mining quantitative values in a database. The algorithm first transforms each quantitative attribute into fuzzy sets and maps items to them via membership functions. An Apriori-like al-

gorithm generates the association rules using the previously collected fuzzy counts.

Another Apriori-like approach is presented in [Gyen00]. It addresses the two main steps of association rule mining, namely the discovery of frequent itemsets and the generation of association rules from quantitative databases. The notation in Table 15 will be used for the algorithm.

D	the database
D_T	the transformed database
F_K	set of frequent k-itemsets (having k items)
C_K	set of candidate k-itemsets (having k items)
I	complete itemset
minsup	support threshold
minconf	confidence threshold
mincorr	correlation threshold

Table 15: Notation

The algorithm first searches the database and returns the complete set containing all attributes of the database. In a second step, a transformed fuzzy database is created from the original one. The user has to define the sets to which the items in the original database will be mapped. After generating the candidate itemsets, the transformed database is scanned in order to evaluate the support and after comparing the support to the predefined minimum support, the items with a too low support are deleted. The frequent itemsets F_K will be created from the candidate itemesets C_K. New candidates are being generated from the old ones in a subsequent step. C_k is generated from C_{k-1} as described for the Apriori algorithm in chapter 3.4.1. The following pruning step deletes all itemsets of C_k if any of its subsets does not appear in C_{k-1}. Finally, the association rules are generated from the discovered frequent itemsets. The pseudocode of the algorithm can be found in Table 16.

Main Algorithm$(minsup, minconf, mincorr, D)$

1) $I = Search(D);$
2) $(C_1, D_T) = Transform(D, I);$
3) $k = 1;$
4) $(C_k, F_k) = Checking(C_k, D_T, minsup);$
5) *while* $(|C_k| \neq \emptyset)$ *do*
6) *begin*
7) $inc(k);$
8) *if* $k == 2$ *then*
9) $C_k = Join1(C_{k-1});$
10) *else* $C_k = Join2(C_{k-1});$
11) $C_k = Prune(C_k);$
12) $(C_k, F_k) = Checking(C_k, D_T, minsup);$
13) $F = F \cup F_k;$
14) *end*
15) $Rules(F, minconf, mincorr)$

Table 16: An Algorithm for mining Fuzzy Association Rules

6 The Project

In order to demonstrate the process of fuzzy association rule mining, a prototype has been implemented in the proceeding of this paper. Especially, the quantitative approach to fuzzy association rules is a major part in the present implementation. The user is given the possibility to mine fuzzy association rules out of any quantitative database. Fuzzy sets are generated first, followed by discovering fuzzy frequent itemsets form the newly constructed database. Finally, fuzzy association rules are generated and evaluated.

The purpose of the program is to demonstrate how fuzzy association rules mining can work in practice. The algorithms work on any compatible quantitative data set, although they may not be fast enough to conduct mining on very large databases.

The program has the form of an R-package which can be installed and run on any computer or platform where R itself is running. It provides several functions which can directly be applied by the user. The functions provided can be called by using the standard R-console.

A very important task of the package is to guide the user through all steps of the mining process, from discovering fuzzy sets to the final generation of the association rules. Still, the preprocessing steps like choosing the right variables for a data set have to be performed by the user himself. Also, no interpretation of the rules is provided, which would be a difficult task to complete anyway. So, some work by the user has still to be done for the program to work.

The following chapter will give a more detailed description about the program, the implemented approach, the functions it provides and the architecture it has been developed with, specifically R. Besides that, an example is offered in order to demonstrate the use of the program. It is a small example guaranteeing a clear demonstration, and still large enough to demonstrate all provided functionalities. In the end of the chapter, a forecast is given describing activities still needed to improve the implementation. Weaknesses of the program will be discussed along with the possibilities of making the program faster and more efficient.

6.1 Key Facts

Project Name	Development of an R-package for fuzzy association rules mining
Author	Lukas Helm
Matriculation number of the author	0251677
Organizer	Priv.Doz. Dr. Michael Hahsler, Institute for Information Business, Vienna University of Economics and Business Administration
Start of project	February 6th, 2007
End of project	July 29th, 2007
Semester	Summer 2007
Topic	Development of an R-package for the demonstration of fuzzy association rules providing the following functionality: • Discover fuzzy sets from quantitative data • Implement the necessary fuzzy set-theoretic operations • Generate frequent itemsets from fuzzy data • Generate the association rules out of the frequent itemsets • Evaluate discovered or presumed rules with fuzzy support and fuzzy confidence values

Tabelle 17: Project Overview

6.2 Architecture

The package was developed in the R environment, which is an environment for statistical computing and graphics. It is available for free from the Internet [Rpro] under the General Public License (GPL). This allows you to use and distribute it freely. It is even allowed to sell it as long as the source code is made available and the receiver has the same rights. Many important platforms are supported by R, like Windows, Linux and Mac OS.

R is actually a programming language which is now developed by the R Development Core Team. It can be regarded as an implementation of the S programming language with its semantics derived from Scheme, a multi-paradigm

programming language. Over the recent years, it has become a de-facto standard for data analysis and the development of statistical software. The design of R enables further computations on results of a statistical analysis [Dalg04]. It is based on a command line interface still allowing graphical representations of results. Furthermore, graphical user interfaces have been developed.

The basic R package already supports a high number of statistical and numerical techniques and is additionally highly expandable with packages contributed by users. Packages provide special functionalities which are not included in R. Besides the core set of packages included in standard R, over 1000 more packages are available from the Comprehensive R Archive Network (CRAN). The program developed in the proceedings of this thesis is designed in a similar way as these packages.

The language is based on a formal computer language, giving it the advantage of high flexibility. Alternative programs that provide simpler user interfaces may look easier at first sight, but in the long run R offers the flexibility needed to conduct complex statistical calculations.

[Rpro] describes the R-environment as follows:
“R is an integrated suite of software facilities for data manipulation, calculation and graphical display. It includes

- an effective data handling and storage facility,
- a suite of operators for calculations on arrays, in particular matrices,
- a large, coherent, integrated collection of intermediate tools for data analysis,
- graphical facilities for data analysis and display either on-screen or on hard copy, and
- a well-developed, simple and effective programming language which includes conditionals, loops, user-defined recursive functions and input and output facilities.”

The R Foundation is a not for profit organization created by the R Development Core Team. It serves the following three goals:

- Ensure the continued development of R as well as providing support for the R Project and other innovations in statistical computing.

- Provide a reference point for possible supporters or interactors with the R development community.
- Administer the copyrights of the R software and documentation.

In addition, R is an official part of the Free Software Foundation's GNU project.

6.3 Approach

After having described all the theoretical approaches of data mining and fuzzy association rules, this chapter demonstrates which theories and algorithms have been implemented in the program. It does not exactly describe the implemented functions (see chapter 6.4) but lists the concepts and ideas underlying the final implementation. An important point is the application of known association rule mining algorithms to the fuzzy case.

6.3.1 Constructing Fuzzy Sets

In most of the literature on fuzzy association rules, the standpoint is that an expert has to define the fuzzy sets which have to be applied for the quantitative attributes of a database. Some attempts have also been made to automatically discover fuzzy sets by implementing techniques like clustering (see chapter 5.3). In the package, the user has two possibilities: he can choose whether he wants to define the fuzzy sets himself or if he wants the program to find the fuzzy sets.

First, we will have a look at what the fuzzy sets look like in the program. Following the idea that the main purpose of fuzzy sets is to overcome the sharp boundary problem (see chapter 5), it is not necessary to be able to enter a single membership function for every fuzzy set in a database. It is sufficient enough to know where the borders of the fuzzy sets lie. The membership values can then be computed easily, the only critical part is the overlapping range of the sets (see chapter 5.1.1).

If the user decides to define his own fuzzy sets, he thereby just has to put in the desired borders for all sets. Not only the borders, but also the number of fuzzy sets for each attribute can be chosen. Fuzzy sets can automatically be discovered from these specifications. It is important that the fuzzy sets cover all values in the attribute, if a value lies outside all of the sets, it can not be mapped to

any of them. For example, a user defines the following borders for four fuzzy sets: $A=\{0,20\}$, $B=\{15,36\}$, $C=\{34,45\}$, $D=\{40,50\}$.

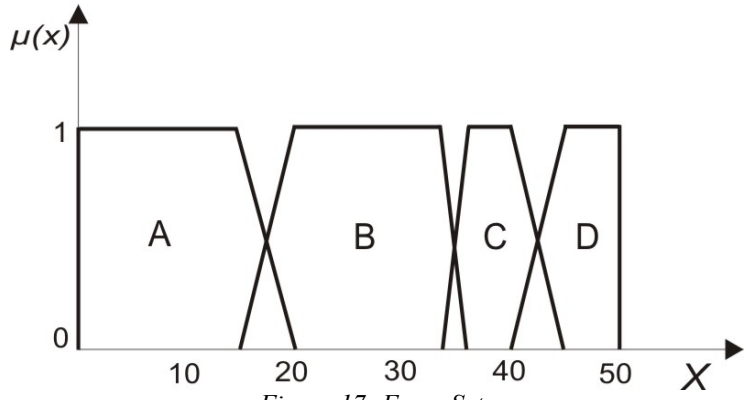

Figure 17: Fuzzy Sets

The resulting fuzzy sets can be seen in Figure 17. We notice that the outer sets A and D both have one crisp border. This is simply due to the fact that they do not have any fuzzy neighbors. The computation of the membership function has already been discussed in chapter 5.1.1.

Should the user not want to specify any fuzzy sets himself, we can easily compute them for him, but without any guarantee for the correctness of the sets for the present data. The approach is to retrieve the fuzzy sets by applying simple statistical measures. In our method, it is only possible to automatically retrieve three fuzzy sets out of any data column. The calculation of the three sets (see Figure 18) works as follows:

- Low set: The lower border of the low set is the minimal value of the quantitative data in the field. The high border of the low set is given by $hb=mean-\frac{sd}{2}+mean\times overlap$.
- Medium set: The lower border of the medium set is calculated by $lb=mean-\frac{sd}{2}-mean\times overlap$,

  the high border by $hb=mean+\frac{sd}{2}+mean\times overlap$.
- For the high fuzzy set, the higher border equals the maximum in the data,

  the low border is $lb=(mean+\frac{sd}{2})-mean\times overlap$.

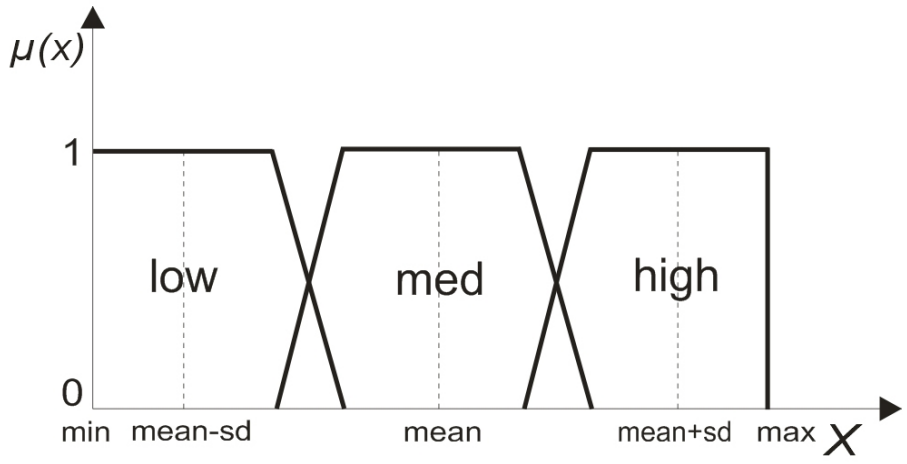

Figure 18: Automatic Set Generation

The variable *overlap* can take any value between 0 and 1. It defines the size of the overlap between the fuzzy sets. A value of 0.1 would make the overlap as big as 0.2 times the mean. Putting in 0 would create crisp sets.

The decision on the right fuzzy sets is crucial for the success of a data mining project, therefore this easy method is not accurate enough and sets should be researched more carefully. However, it will give users a quick start for experimenting with the idea of fuzzy association rules. For use in a real project, fuzzy sets will have to be defined a priori or a more sophisticated algorithm has to be used for finding them.

6.3.2 Constructing a Dataset for Mining

After having defined the fuzzy sets, a new data set enabling the mining of fuzzy association rules has to be constructed out of the original data. This process is rather simple and intuitive, since the values only need to be fitted into the sets. For every fuzzy set that we have previously defined, there is one row in the new database containing the grade of membership of the single items to the specific set. Figure 19 visualizes the process of getting the membership values of a data point to different fuzzy sets.

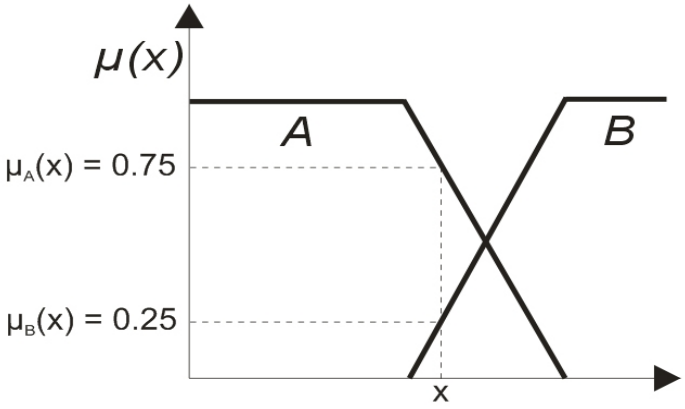

Figure 19: Membership of an Item

As an example, we will look at a sample transaction database representing one row of the original database: $t=\{5,27,12,17,22,16,9,14\}$ Three fuzzy sets have been defined as follows: $A=\{0,15\}$, $B=\{11,23\}$ and $C=\{20,30\}$. The row will be subdivided into three rows, one for each fuzzy set. The new table will only contain the membership values to these fuzzy sets (see Table 18).

A	B	C
1	0	0
0	0	1
0.75	0.25	0
0	1	0
0	0.33	0.67
0	1	0
1	0	0
0.25	0.75	0

Table 18: New Database

6.3.3 Calculation of Fuzzy Operations

A decision has to be made on which t-norms should be used for the calculation of the fuzzy support and confidence measures. The calculation is conducted according to chapter 5.2, using the Łukasiewicz logic and the Łukasiewicz t-norm respectively.

$$supp(A \rightarrow B) = \sum_{(x,y) \in D} min(A(x), B(x))$$

$$conf(A \rightarrow B) = \frac{\sum_{(x,y) \in D} min(A(x), B(y))}{\sum_{(x,y) \in D} A(x)}$$

This specific method has been chosen because it appears to give reasonable results and in addition it is the most popular method for calculating fuzzy operations. Also the other fuzzy operations like complement or implication have been designed in this way (see chapter 4.2). Calculating the support is important for the frequent itemset generation, fuzzy confidence for generating the rules.

6.3.4 Frequent Itemset Generation: the FP-Growth Algorithm

For generating the frequent itemsets from the database, the FP-Growth algorithm has been chosen (see chapter 3.4.2). The issue is to transform the original algorithm in a way that it can deal with fuzzy databases. This is quite simple in the case of FP-Growth, because fuzzy data can be incorporated easily into an FP-Tree. Mining the tree then does not differ significantly from the original algorithm.

6.3.4.1 FP-Tree Construction

Constructing an FP-Tree from a fuzzy database is rather straightforward. The only difference to the original algorithm is that it is not enough to count the occurrences of an item , but the value of membership has to be considered as well. This membership value is then simply added to the overall count of the node. The database of fuzzy values in Table 19 is used for FP-Tree construction.

A	B	C	D	E
1	0	0.3	0	1
0	0.6	1	0	0
0.2	1	1	1	0
0	0.4	0	0.9	1
0	0	1	0	0.4
0	0	1	1	0.1
1	1	0	0	0.2

Table 19: Sample Fuzzy Database

Generating the FP-Tree from this database will lead to a tree containing the sums of the fuzzy values in the nodes. The tree created from this database can be seen in Figure 20.

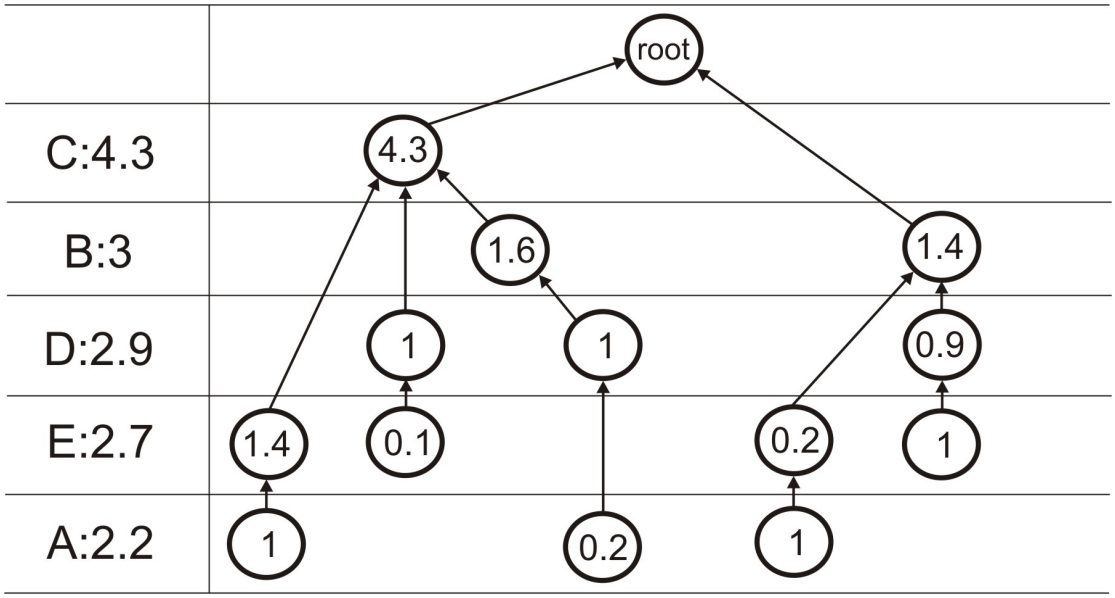

Figure 20: Fuzzy FP-Tree

It is now easy to calculate the support of a path in the tree because it is simply the minimum of the path that is controlled. That tree can be used for conducting the FP-Growth algorithm, described in the following chapter.

6.3.4.2 FP-Growth

The FP-Growth works very similar to the standard algorithm for binary data. The difference is that it uses the fuzzy values for measuring support, constructing the conditional pattern base and building the new conditional FP-Tree. The minimum support can be checked by comparing the minimum of one path with the user-defined threshold. The conditional pattern base for the items in the tree of Figure 20 looks as demonstrated in Table 20.

item	conditional pattern base
A	$\{\langle C, E:1\rangle, \langle C, B, D:0.2\rangle, \langle B, E:0.2\rangle\}$
E	$\{\langle C:1.4\rangle, \langle C, D:0.1\rangle, \langle B:0.2\rangle, \langle B, D:0.9\rangle\}$
D	$\{\langle C:1\rangle, \langle C, B:1\rangle, \langle B:0.9\rangle\}$
B	$\{\langle C:1.6\rangle\}$
C	$\emptyset$

Table 20: Fuzzy Conditional Pattern Base

It is important to notice that, unlike in the classical approach, lower values than in the leaf nodes can appear in the path of a tree. This is why we can not use the values of the leaf nodes for building the conditional patterns. Instead, we have to use the minimum value of the path. Therefore, some data might get lost, but we will accept this distortion in order to enable fuzzy associations mining.

If we take the minimum of a certain path in the tree as its support, it might actually lead to a false value. This is due to the fact that elements in higher levels of the tree can contribute with even lower values to the specific path. This results in a possible discovery of itemsets that in reality are infrequent. Therefore, the discovered frequent itemsets should once more be tested for their support before recording them into the final set of frequent transactions.

6.3.5 Generation of Association Rules

The generation of the association rules works as defined in [AgIS93]. The antecedent can consist of any number of items, but in the consequent, there is only one item allowed. Every item in an itemset will be given the role of the consequent, and the confidence will be compared to the specified minimum confidence value. The methods for calculating the fuzzy confidence will be utilized. Interesting rules (i.e. these rules where the confidence exceeds the minimum confidence threshold) will be stored.

6.4 The Program

This chapter introduces the functions of the program and gives a detailed description of how the implementation works. The functions are illustrated with help of a sample data set. The database has three quantitative attributes taken from the database AdultUCI provided by the R-package “arules”. Datasets the user wants to use with this package need some preprocessing. It is important that the database we utilize for discovering fuzzy sets only consists of quantitative attributes. It also contains only the attributes that we intend to use for mining and additionally the table needs a name for each column.

```
> age <- AdultUCI[1:50,1]
> fnlwgt <- AdultUCI[1:50,3]
> hpw <- AdultUCI[1:50,13]
> testdata <- cbind(age, fnlwgt, hpw)
> testdata
      age fnlwgt hpw
 [1,]  39  77516  40
 [2,]  50  83311  13
 [3,]  38 215646  40
 [4,]  53 234721  40
 [5,]  28 338409  40
 [6,]  37 284582  40
 [7,]  49 160187  16
 [8,]  52 209642  45
 [9,]  31  45781  50
[10,]  42 159449  40
...
```

The first functions deal with the construction of the fuzzy sets from each column of the original data set:

The function ***getCenters*** computes centers for the fuzzy sets. Actually, these centers are not real centers, but give an indication for placing the borders of the set. The function calculates three centers for three fuzzy sets. The positioning of the centers has already been demonstrated in chapter 6.3.1, namely $mean - sd$ for the low set, $mean$ for the medium set, and $mean + sd$ for the high set. The three functions ***getLow***, ***getMedium*** and ***getHigh*** lead to the borders for the newly discovered fuzzy sets. All of these methods use *getCenters* for the computation of the borders.

Using the function ***getFuzzySets***, the three fuzzy sets of a vector will be returned in form of a list containing vectors. The functions *getLow*, *getMedium* and *getHigh* are used within this function.

```
> sets <- getFuzzySets(testdata[,1])
> sets
Fuzzy Set:
low 19 36.3
med 28.7 47.9
high 40.2 59
```

These are the borders of the fuzzy sets discovered for one attribute of the database, but we need the sets of all attributes for proceeding with the mining process. Hence, the function ***getAllSets*** provides us with all the fuzzy sets for the database. It simply applies the function *getFuzzySets* to all the columns and returns a list of the sets. Additionally, the sets are presented in a user-readable form.

```
> allsets <- getAllSets(testdata)
> allsets
[[1]]
Fuzzy Set:
low 19 36.3
med 28.7 47.9
high 40.2 59

[[2]]
Fuzzy Set:
low 28887 169798
med 128915 279920
high 239036 544091

[[3]]
Fuzzy Set:
low 13 39.3
med 31 51.5
high 43.2 80
```

The sets are saved in a list, which contains another list for each attribute of the database.

ComputeIndividualMem is a function for calculating the membership for one data point of the original database to the fuzzy sets discovered in the previous step. The method has already been demonstrated in chapter 6.3.1. The function needs the value of the data point and the fuzzy sets as an input.

```
> mem <- computeIndividualMem(30, sets)
> mem
[1] 0.8286 0.1714 0.0000
```

In the above case, the data point shows a membership of 0.83 to the first set and a membership of 0.17 to the second set. If the data point has only a membership in one of the fuzzy sets, all other values in the returned vector will be 0.The function thereby creates one row for the matrix of the attribute.

Now, that we have discovered the membership values for one column, we have to put them in a form that enables association rules mining. The function ***createMembership*** puts together the single rows generated by using the function *computeIndividualMem*. It uses the membership values for a specific attribute and puts them in a matrix with as many columns as fuzzy sets (see Table 21).

Set 1	Set 2	Set 3
1	0	0
0	0.78	0.22
0	0.18	0.82
0	1	0
0	0	1
0	1	0

Table 21: Mapping Table

As an input, it requires the column of the original data set to be examined and the fuzzy sets dedicated to it. The fuzzy sets are required for calling the function *computeIndividualMem*, whose generated vectors are connected making the membership matrix for the specific attribute.

```
> m <- createMembership(testdata[,1], sets)
> m
        [,1]    [,2]   [,3]
...
 [7,] 0.0000 0.00000 1.0000
 [8,] 0.0000 0.00000 1.0000
 [9,] 0.6980 0.30200 0.0000
[10,] 0.0000 0.76699 0.2330
[11,] 0.0000 1.00000 0.0000
[12,] 0.8286 0.17138 0.0000
...
```

The function ***generateMineableMatrix*** combines all the previous functions. It uses *createMembership* in order to form the final matrix that makes mining possible. For the user, this is the most important function because theoretically he will never have to use the former other two in order to generate the matrix for mining.

```
> mm <- generateMineableMatrix(testdata, allsets)
> mm
       [,1]   [,2]  [,3]   [,4]  [,5]   [,6]  [,7]  [,8]   [,9]
 [1,] 0.000 1.0000 0.000 1.0000 0.000 0.0000 0.000 1.000 0.0000
 [2,] 0.000 0.0000 1.000 1.0000 0.000 0.0000 1.000 0.000 0.0000
 [3,] 0.000 1.0000 0.000 0.0000 1.000 0.0000 0.000 1.000 0.0000
 [4,] 0.000 0.0000 1.000 0.0000 1.000 0.0000 0.000 1.000 0.0000
 [5,] 1.000 0.0000 0.000 0.0000 0.000 1.0000 0.000 1.000 0.0000
 [6,] 0.000 1.0000 0.000 0.0000 0.000 1.0000 0.000 1.000 0.0000
 [7,] 0.000 0.0000 1.000 0.2351 0.765 0.0000 1.000 0.000 0.0000
 [8,] 0.000 0.0000 1.000 0.0000 1.000 0.0000 0.000 0.784 0.2165
 [9,] 0.698 0.3020 0.000 1.0000 0.000 0.0000 0.000 0.177 0.8227
[10,] 0.000 0.7670 0.233 0.2531 0.747 0.0000 0.000 1.000 0.0000
[11,] 0.000 1.0000 0.000 0.0000 0.000 1.0000 0.000 0.000 1.0000
[12,] 0.829 0.1714 0.000 0.6971 0.303 0.0000 0.000 1.000 0.0000
```

The next important point is to save the corresponding columns of the new data set and the original set (i.e. which fuzzy set belongs to which original column of the data set). This task is performed by a function called ***getMatrixIndex***. As an input, it needs the original data set as well as the fuzzy sets. It is important that the columns of the original set have names (any name will be enough, it can be as simple as “field1”, “field2” etc.) otherwise the function gives back an empty list. The list contains the names of the fields and the starting column in the new data set. As we do not know how many fuzzy sets each attribute has, it is necessary to know which fields in the new data set belong to which column in the old database. Otherwise, the discovered rules do not make sense because they can not be mapped to the original attributes. The index list looks as follows:

```
> index <- getMatrixIndex(testdata, allsets)
> index
[[1]]
[1] "age" "1"

[[2]]
[1] "fnlwgt" "4"

[[3]]
[1] "hpw" "7"
```

After generating the results of the mining process, this list enables the user to identify which sets belong to which original attributes. The first value names the original attribute name, the secaond value defines the starting column in the transformed database.

Having performed this task, we can try to put in some rules and generate their support and confidence values. For generating the support, we do not primary need a rule, a set of items is enough. A vector containing the column numbers of the itemset is needed to call the function ***generateSupport***. The function sums up the minimum values of the specified itemsets in each row of the fuzzy database. As defined before, the support of an itemset is determined by summing up the minimum value of each tuple.

```
> sup14 <- generateSupport(c(1,4), mm)
> sup14
[1] 4.134083
```

The function gives back the support in an absolute value. But the user might as well be interested in the relative support. For computing this, we just need to

divide the support by the number of transactions in the database. By default, the function returns the absolute value. By passing `relative=TRUE` to the function, the relative support value can be retrieved:

```
> relsup14 <- generateSupport(c(1,4), mm, relative=TRUE)
> relsup14
[1] 0.0827
```

In this case, the relative support of the itemset {1,4} would be 8.27%. To generate the confidence, an itemset is not enough. We now need a rule containing an antecedent and a consequent. For creating a rule, the function ***makeRule*** can be used, where it is necessary to put in the antecedent and the consequent in form of a vector. The function *makeRule* returns a list with the antecedent and the consequent of the rule. Alternatively, it is also possible to define the list without using this function.

```
> rule <- makeRule(1, 4)
> rule
Association Rule:
1 ---> 4
support:
confidence:
```

Obviously, the rule does not have a support or a confidence yet. Those values can be accessed with `rule$sup` and `rule$conf`. A rule generated by this method can then be used as an input for the function ***generateConfidence*** returning the confidence of the rule. Of course, the function as well needs the data as input. It calculates the confidence of a rule according to the method in chapter 6.3.3, returning a relative value.

```
> generateConfidence(rule, mm)
[1] 0.2385930
```

6.4.1 Mining Frequent Itemsets

As mentioned in chapter 6.3.4, the FP-Growth algorithm has been chosen for the mining of fuzzy frequent itemsets. Before building the initial FP-Tree, some preprocessing is necessary in order to optimize the performance of the algorithm. The core of the preprocessing task is the function ***preProcessFP***, which is also the function applied by the user. It will return a sorted dataframe that contains all columns meeting the minimum support in decreasing order. The input for this function is the fuzzy data set and a minimum support value specified by the user.

The function ***getSingleItemSupport*** returns a data frame that contains the original position of the column and its relative support value. The original position has to be stored in order to be able to trace back the columns after having recombined them.

```
> sisup <- getSingleItemSupport(mm)
> sisup
  place   support
1     1 0.3465386
2     2 0.3098267
3     3 0.3436347
4     4 0.3368870
5     5 0.3763121
6     6 0.2868009
7     7 0.1468253
8     8 0.6724630
9     9 0.1807117
```

In the subsequent step, the next function ***getPreProcessedIndex*** is processed which removes the items from the frame that do not meet the minimum support and sorts the remaining items by support in a decreasing order. This index is the basis for constructing the preprocessed data set that optimizes mining frequent itemsets with an FP-Tree. The input is the previously retrieved data frame and the minimum support.

```
> preind <- getPreProcessedIndex(sisup, 0.3)
> preind
  place   support
8     8 0.6724630
5     5 0.3763121
1     1 0.3465386
3     3 0.3436347
4     4 0.3368870
2     2 0.3098267
```

In this specific case, all of the items below a support value of 0.3 are deleted and the rest sorted. The previous two functions are then used in the function *preProcessFP*. It needs the fuzzy data and the minimum support threshold as an input and returns the new sorted data set in form of a matrix. The previously constructed index is the guideline for doing this because it tells the function in which position the column has to be put in.

```
> premm <- preProcessFP(mm, 0.3)
> premm
       [,1]  [,2]  [,3]  [,4]   [,5]   [,6]
 [1,] 1.000 0.000 0.000 0.000 1.0000 1.0000
 [2,] 0.000 0.000 0.000 1.000 1.0000 0.0000
 [3,] 1.000 1.000 0.000 0.000 0.0000 1.0000
 [4,] 1.000 1.000 0.000 1.000 0.0000 0.0000
 [5,] 1.000 0.000 1.000 0.000 0.0000 0.0000
 [6,] 1.000 0.000 0.000 0.000 0.0000 1.0000
 [7,] 0.000 0.765 0.000 1.000 0.2351 0.0000
 [8,] 0.784 1.000 0.000 1.000 0.0000 0.0000
 [9,] 0.177 0.000 0.698 0.000 1.0000 0.3020
[10,] 1.000 0.747 0.000 0.233 0.2531 0.7670
[11,] 0.000 0.000 0.000 0.000 0.0000 1.0000
[12,] 1.000 0.303 0.829 0.000 0.6971 0.1714
...
```

After having preprocessed the data, the initial FP-Tree can be constructed. This is done in the function ***makeFPTree***. This function only requires the accordingly preprocessed data set as an input and constructs the tree as demonstrated in chapter 6.3.4.1. The function first generates root elements for each column of the data set, giving them the value 0. These values can be added later on. Having performed this, the function loops through all the tuples of the database. For the actual tuple, it is first being checked if the itemset contains only a single item. If this is the case, the function goes on to the next tuple.

The function puts all other tuples which are not a single item itemset in the tree. This is performed as follows: The first item in the tuple that is not 0 is searched. For this item, a root element entry is made (i.e. the value of the item is added to the previously constructed root node). For every subsequent item, the algorithm checks if this same path already exists or not, and if it does exist, the value of the item is simply added to the existing node. If it does not exist, a new node is created in the tree.

The result of the function *makeFPTree* is a tree in form of a matrix with three columns. The first column contains the number of the column in the data set it corresponds to, the second one shows a reference number to the ancestor of the current node, and the third column saves the accumulated value of each node.

```
> tree <- makeFPTree(premm)
> tree
      depth parent support
 [1,]     1      0  35.623
 [2,]     2      0   3.765
 [3,]     3      0   1.000
 [4,]     4      0   1.000
 [5,]     5      0   0.000
 [6,]     6      0   0.000
 [7,]     5      1   5.000
 [8,]     6      7   3.000
 [9,]     5      4   1.000
[10,]     2      1  15.051
[11,]     6     10   1.000
[12,]     4     10   7.233
[13,]     3      1   5.098
...
```

The first six nodes are the root nodes, one for each column of the dataset. It is easy to spot the root nodes because they have 0 in the second column which represents their ancestor. Obviously, no itemsets did start with the items 5 and 6. This tree comprises the basis for conducting the FP-Growth algorithm.

The next step is to mine the tree, using the function ***mineFPTree***. This function generates all frequent itemsets contained in the data from two inputs, namely the previously constructed tree and the minimum support. It simply cycles through all the depth levels from the tree, starting at the bottom, and executes the function ***FPGrowth*** for each level. This function is the most complex one of the whole package and will be discussed in the following section.

As inputs, the *FPGrowth* function needs an FP-Tree, the minimum support, the current level of depth at which the FP-Growth should be performed and a counter that indicates if it is the first execution of the function for the current level. The function is performed recursively, returning a list of discovered frequent itemsets. In the following paragraphs the mode of operation of the *FPGrowth* function is described.

The first thing it does is counting the number of paths a tree contains. It does this by simply counting the nodes which do not have any children, we call these nodes endnodes. If the tree contains only one endnode, it is a single-path tree, if it contains more than one it has multiple paths and has to be treated differently.

If it is a single path tree, the only task is to build every possible combination of the items in the tree and to compare each of these combinations with the minimum support. For building all the possible combinations, a function called ***pow-***

erSet is used. The combinations showing a greater support than the minimum are added to the list of frequent itemsets. The mining process ends here with returning this list of frequent itemsets..

```
> itemset
[1] 1 3 4
> powerSet(itemset)
[[1]]
[1] 1

[[2]]
[1] 3

[[3]]
[1] 1 3

[[4]]
[1] 4

[[5]]
[1] 1 4

[[6]]
[1] 3 4

[[7]]
[1] 1 3 4
```

It gets a little bit more complicated if we are dealing with a tree made up of more than one path. Then, further growing of the tree becomes necessary. Here, the counter has got an important role. For each level of depth, all possible combinations of items have to be discovered and added together. This is done by going up every level of the tree and executing the function recursively on each of the levels. The counter signals the function if this is the first iteration for a specific level of depth or not. Different actions are performed in either case.

In case of the first iteration, the conditional pattern base for the current level is constructed which is a list containing the items in each path of the tree and their corresponding support. The itemsets in this list meeting the minimum support are then added to the list of frequent itemsets, but sets not meeting the minimum support are not deleted from the conditional pattern base.

If it is not the first iteration, the function behaves differently. Again, the conditional pattern base is constructed in the beginning. But this time, the level is decreased by one so that the function moves up in the tree in order to grow the frequent patterns. For this next level, the conditional FP-Tree is constructed out of the conditional pattern base. Using this new tree, the decreased level and a

count of 0, the function is executed again for treating the next level. The frequent itemsets returned from the function are added to the already discovered ones.

Having performed either of the two previous procedures, the function generates the conditional FP-Tree out of the conditional pattern base which varies depending on which procedure has been used. It increases the counter and calls *FPGrowth* again, using the new conditional FP-Tree, the level and the increased counter as an input. Table 22 shows how the implementation works.

```
FPGrowth(tree, minsup, level, count) {
  if(tree is single path) {
    generate all possible combinations of items
    add itemsets with support > minsup to frequent sets list
  }
  if(tree has more than one path) {
    if(it is the first iteration for current level) {
      generate cpb
      add itemsets in cpb to frequent itemsets if support > minsup
    }
    if(it is not the first iteration for current level) {
      newlevel = level-1
      generate cpb (of newlevel)
      generate conditional fp-tree (condfptree)
      call FPGrowth(condfptree, minsup, newlevel, 0)
      add returned itemsets to frequent itemsets
    }
    generate conditional fp-tree
    count = count + 1
    call FPGrowth(condfptree, minsup, level, count)
  }
  return frequent itemsets
}
```

Table 22: FP-Growth Implementation

Executing the function will return all frequent itemsets (according to the minimum support) involving the defined level of depth. The function *mineFPTree* will perform this on all the levels of the tree returning the complete set.

```
> freqsets<-FPGrowth(tree, 1, 5, 0)
> freqsets
[[1]]
Itemset:
attributes: 4 5
support 1

[[2]]
Itemset:
attributes: 1 3 5
support 4
```

```
[[3]]
Itemset:
attributes: 3 5
support 1
...
```

All frequent itemset can be retrieved using the function *mineFPTree*:

```
> freqsets <- mineFPTree(tree, 1)
> freqsets
[[1]]
Itemset:
attributes: 1 5 6
support 3

[[2]]
Itemset:
attributes: 1 2 6
support 1

[[3]]
Itemset:
attributes: 1 3 5 6
support 1.6
...
```

As mentioned before, the discovered frequent itemsets should be evaluated again and compared to the minimum support in order to avoid discovery of infrequent itemsets. This is done by the function ***verifySupport***. Only sets meeting the minimum support requirement are returned in a list:

```
> verfreq <- verifySupport(freqsets, premm, 1)
> verfreq
[[1]]
Itemset:
attributes: 1 5 6
support 6.12

[[2]]
Itemset:
attributes: 1 2 6
support 3.26

[[3]]
Itemset:
attributes: 1 2 3 6
support 1.13

[[4]]
Itemset:
attributes: 2 6
support 4.52
...
```

The same function can also be used for itemsets that have not been evaluated at all.

6.4.2 Generating Association Rules

After having generated the frequent itemsets, generating association rules out of them is a rather easy task. In the function ***evaluateRules***, all rules are investigated that only have one single item in the consequent. The input to this function are all possible rules derived from the frequent itemsets and discovered with the FP-Growth method with a minimum confidence threshold.

For discovering all candidate rules from the itemset, another function is needed called ***generateRules***. With this function all possible rules with a single item consequent are built from the frequent itemsets and returned in a list. This list is the input for the function *evaluateRules* that will return only the rules having a confidence bigger than the minimum confidence defined by the user. The list contains the elements of the set and the support, which has to be verified as mentioned above.

```
> rules <- generateRules(verfreq)
> rules
[[1]]
Association Rule:
5 6 ---> 1
support: 6.12
confidence:

[[2]]
Association Rule:
1 6 ---> 5
support: 6.12
confidence:

[[3]]
Association Rule:
1 5 ---> 6
support: 6.12
confidence:

[[4]]
Association Rule:
2 6 ---> 1
support: 3.26
confidence:
...
```

Every rule does now have a support value, but the confidence is still missing. The above described function *evaluateRules* is now needed to calculate the confidence for each rule. It returns a list containing all relevant attributes of a rule: the antecedent, the consequent, the support and the confidence. Additionally, a minimum confidence can be put in that the evaluated rules are compared

with. If the computed confidence shows a value above the minimum confidence threshold, it is incorporated in the list.

```
> evalrules <- evaluateRules(rules, premm, 0.1)
> evalrules
[[1]]
Association Rule:
5 6 ---> 1
support: 6.12
confidence: 0.866

[[2]]
Association Rule:
1 6 ---> 5
support: 6.12
confidence: 0.542

[[3]]
Association Rule:
1 5 ---> 6
support: 6.12
confidence: 0.477

[[4]]
Association Rule:
2 6 ---> 1
support: 3.26
confidence: 0.722
...
```

Finally, it is relevant to know which columns in the original database the discovered rules belong to. This functionality is provided by the function ***traceBack***. After putting in the rules, the original data set and the fuzzy data set, the function returns the rules with their original columns of the database.

```
> finalrules <- traceBack(evalrules, index, preind)
> finalrules

[[1]]
Association Rule:
fnlwgt 1 age 2 ---> hpw 2
support: 6.12
confidence: 0.866

[[2]]
Association Rule:
hpw 2 age 2 ---> fnlwgt 1
support: 6.12
confidence: 0.542

[[3]]
Association Rule:
hpw 2 fnlwgt 1 ---> age 2
support: 6.12
confidence: 0.477
```

```
[[4]]
Association Rule:
fnlwgt 2 age 2 ---> hpw 2
support: 3.26
confidence: 0.722
...
```

The function exchanges the items with the name of the column and the number of the fuzzy set in the original database. This final step creates rules which can be used and understood by the user. The process of discovering fuzzy association rules ends here.

6.5 Discussion

There is some more work to do, especially on the FP-Growth algorithm. In the first place, its quality has to be investigated by comparing its results to the results of other fuzzy frequent itemset discovery methods. For example, it would be possible to compare it to an Apriori-like approach. Unfortunately, the author did not find any available implementations to do so. Comparing these algorithms would make it possible to evaluate the quality of the implemented FP-Growth algorithm by finding out whether it discovers all relevant itemsets or not. Also, the speed of different methods is still to be compared because it is not known whether the FP-Growth algorithm is as efficient as other algorithms in a fuzzy context.

Another task is to find out if the FP-Growth algorithm is ideal for mining fuzzy frequent itemsets because it might also rate itemsets as frequent while they are infrequent in reality (see chapter 6.3.4.2). The so arising necessity to evaluate the itemsets again slows down the algorithm. It is a big disadvantage of the fuzzy algorithm compared to the original one that the support values can not be directly taken from the FP-Tree. Generally speaking, an important future task will be to evaluate whether the FP-Tree structure is adequate for mining fuzzy frequent itemset in general or not. At least, an improvement of the algorithm will be necessary in order to prevent it from finding many sets that in consequence have to be discarded.

Another improvement to the program could be the incorporation of an algorithm to mine multi-consequent association rules. This part should not be too difficult because it simply requires a different generation of candidates from the

frequent itemsets. The confidence for these multi-consequent association rules can easily be evaluated with the existing package.

7 Conclusion/Review

A lot of differing work has been done on fuzzy association rules, using different methods and different approaches. This thesis tried to put the different approaches together in order to develop an understanding what the term “fuzzy association rules” can mean. Depending on how it is interpreted, it can be used in different domains. The most common usage, though, is to overcome the sharp boundary problem when mining association rules from quantitative data. It is easy to compute membership values here, the critical task is to find the right amount and range of the fuzzy sets in order to lose as little information as possible and still be able to discover some interesting rules. This goal is facing a trade-off with the computation time, because the more sets we choose the longer the generation of fuzzy association rules will take.

The developed tool allows experimenting with the concepts of fuzzy association rules. It implements the most important functionalities needed to conduct the mining of the rules. Fuzzy sets can be discovered and the data can be put in a mineable format. For the generation of frequent patterns, the FP-Growth algorithm has been chosen. Although it is not clear whether this algorithm is ideal for fuzzy associations mining, it can discover some fuzzy frequent itemsets that will further be used for mining. It is difficult to evaluate whether the algorithm is good because there is no implementation easily available for comparison. Still, the user is able to discover some itemsets. That can help the user developing an understanding for the whole topic and enables him to mine his own rules from any quantitative dataset.

Concluding, the idea of mining fuzzy association rules is very interesting, but not yet mature. Algorithms to enable mining have to be developed and compared in order to find out whether rules can be discovered that are valuable for a business. Due to the fact that most databases contain quantitative data, fuzzy association rule mining methods might achieve wide acceptance in the future.

Index of Figures

Index of Tables

Bibliography

[Adam00] Adamo, Jean-Marc: Data Mining for Association Rules and Sequential Patterns: Sequential and Parallel Algorithms. Springer, 2000.

[AgIS93] Agrawal, Rakesh; Imielinski,Tomasz; Swami, Arun: Mining Association Rules between Sets of Items in Large Databases. Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, 1993.

[AgSr94] Agrawal, Rakesh; Srikant, Ramakrishnan: Fast Algorithms for Mining Association Rules. Proc. 20th Int. Conf. Very Large Data Bases, VLDB, 1994.

[Borg05] Borgelt, Christian: An Implementation of the FP-growth Algorithm. ACM Press, New York, NY, USA, 2005.

[BuKZ95] Butnariu, Dan; Klement,Erich Peter; Zafrany, Samy: On Triangular Norm-Based Propositional Fuzzy Logics. Fuzzy Sets and Systems Volume 69, 1995.

[Cant95] Cantor, Georg: Beiträge zur Begründung der transfiniten Mengenlehre. Springer Berlin / Heidelberg, Mathematische Annalen, Volume 46, Number 4 / November 1895, 1895.

[CCKK99] Chapman, Pete; Clinton, Julian; Kerber, Randy; Khabaza, Thomas; Reinartz, Thomas; Shearer, Colin; Wirth, Rüdiger: CRISP-DM 1.0: Step-by-step data mining guide. CRISP-DM consortium, 1999.

[CeRo06] Ceglar, Aaron; Roddick, John F.: Association mining. ACM Computing Surveys (CSUR), Volume 38 Issue 2, 2006.

[ChAu98] Chan, Keith C.C.; Au, Wai-Ho: An Effective Algorithm for Discovering Fuzzy Rules in Relational Databases. Fuzzy Systems Proceedings. IEEE World Congress on Computational Intelligence, 1998.

[ChWe02] Chen, Guoging; Wei, Qiang: Fuzzy association rules and the extended mining algorithms. Information Sciences-Informatics and

Computer Science: An International Journal, 2002.

[CoCK03] De Cock, Martin; Cornelis, Chris; Kerre, Etienne E.: Fuzzy Association Rules: a Two-Sided Approach. Proceedings Int. Conf. on Fuzzy Information Processing - Theories and Applications, 2003.

[Dalg04] Dalgaard, Peter: Introductory Statistics with R. Statistics and Computing, Springer, 2004.

[Delg03] Delgado, Miguel: Fuzzy Association Rules: an Overview. BISC Conference, 2003.

[DMSV03] Delgado, Miguel; Marin, Nicolas; Sanchez, Daniel; Vila, María-Amparo: Fuzzy association rules - general model and applications. IEEE Transactions on Fuzzy Systems, Vol. 11, No. 2, 2003.

[DuHP03] Dubois, Didier; Hüllermeier, Eyke; Prade, Henri: A Note on Quality Measures for Fuzzy Association Rules. Proceeding of the 10th International Fuzzy Systems Association World Congress on Fuzzy Sets and Systems, Springer, 2003.

[DuHP06] Dubois, Didier; Hüllermeier, Eyke; Prade, Henr: A Systematic Approach to the Assessment of Fuzzy Association Rules. Data Mining and Knowledge Discovery, Volume 13 , Issue 2, 2006.

[DuPS03] Dubois, Didier; Prade, Henri; Sudkamp, Thomas: A Discussion of Indices for the Evaluation of Fuzzy Associations in Relational Databases. Springer Berlin / Heidelberg, 2003.

[DuPS05] Dubois, Didier; Prade, Henri; Sudkamp, Thomas: On the Representation, Measurement, and Discovery of Fuzzy Associations. IEEE Transactions on Fuzzy Systems, Volume 13, Issue 2, 2005.

[FaPS96a] Fayyad, Usama; Piatetsky-Shapiro, Gregory; Smyth, Padhraic: Knowledge Discovery and Data Mining: Towards a Unifying Framework. In Proceeding of The Second Int. Conference on Knowledge Discovery and Data Mining, pages 82--88, 1996.

[FaPS96b] Fayyad, Usama; Piatetsky-Shapiro, Gregory; Smyth, Padhraic: The KDD Process for Extracting Useful Knowledge from Volumes of Data. Communications of the ACM, Volume 39, Issue 11 Pages: 27 - 34, 1996.

[FaPS96c] Fayyad, Piatetsky-Shapiro, Smyth: From Data Mining to Knowledge Discovery in Databases. AI MAgazine, 1996.

[FrPM92] Frawley, William J.; Piatetsky-Shapiro, Gregory; Matheus, Christopher J.: Knowledge Discovery in Databases: an Overview. AAAI/MIT Press, 1992.

[FWSY98] Fu, Ada Wai-chee; Wong, Man Hon; Sze, Sui Chun; Wong, Wai Chiu; Wong, Wai Lun; Yu, Wing Kwan: Finding Fuzzy Sets for the Mining of Fuzzy Association Rules for Numerical Attributes. In Proceedings of the First International Symposium on Intelligent Data Engineering and Learning (IDEAL'98), 1998.

[Gott06] Gottwald, Siegfried: Universes of Fuzzy Sets and Axiomatizations of Fuzzy Set Theory. Studia Logica Volume 82, Number 2 / März 2006, Springer, 2006.

[GrKW01] DeGraaf, Jeannette M.; Kosters, Walter A.; Witteman, Jeroen J.W.: Interesting fuzzy association rules in quantitative databases. Lecture Notes in Computer Science, Volume 2168, 2001.

[Gyen00] Gyenesei, Attila: A Fuzzy Approach for Mining Quantitative Association Rules. Turku Centre for Computer Science Technical Reports, 2000.

[HaNe01] Hansen, Hans Robert; Neumann, Gustaf: Wirtschaftsinformatik I. Lucius & Lucius, 2001.

[HaPY99] Han, Jiawei; Pei, Jian; Yin, Yiwen: Mining Frequent Patterns without Candidate Generation. 2000 ACM SIGMOD Intl. Conference on Management of Data, ACM Press, 1999.

[HiGN00] Hipp, Jochen; Guentzer,Ulrich; Nakhaeizadeh, Gholamreza: Algorithms for Association Rule Mining - A General Survey and

Comparison. ACM SIGKDD Explorations Newsletter, Volume 2 , Issue 1, 2000.

[HoKC99] Hen, Tzung-Pei, Kuo, Chan-Sheng; Chi, Sheng-Chai: A Fuzzy Data Mining Algorithm for Quantitative Values. Knowledge-Based Intelligent Information Engineering Systems, Third International Conference, 1999.

[KlFo88] Klir, George J.; Folger, Tina A.: Fuzzy Sets, Uncertainty, And Information. Prentice Hall, Englewood Cliffs, New Jersey, 1988.

[KuFW98] Kuok, Chan Man; Fu, Ada; Wong, Man Hon: Mining Fuzzy Association Rules in Databases. SIGMOD Record Volume 27, 1998.

[Liu07] Liu, Bing: Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data. Springer, 2007.

[MaTV97] Mannila, Heiki; Toivonen, Hannu; Verkamo, A. Inkeri: Discovery of Frequent Episodes in Event Sequences. Data Mining and Knowledge Discovery, Volume 1, Issue 3, 1997.

[MeNa99] Mesiar, Radko; Navara, Mirko: Diagonals of continuous triangular norms. Fuzzy Sets and Systems 104, 1999.

[Nels06] Nelson, Roger B.: An Introduction to Copulas. Springer Science+Business Media, 2006.

[RoGe03] Roiger, Richard J.; Geatz, Michael W.: Data Mining: A Tutorial-Based Primer. Addison Wesley, 2003.

[Rpro] Unknown Author: The R Project for Statistical Computing. http://www.r-project.org/, last call July 17th, 2007.

[SrAg95] Srikant, Ramakrishnan; Agrawal, Rakesh: Mining Generalized Association Rules. In Proc. of the 21st Iternational Conference on Very Large Databases, Zurich, Switzerland, 1995.

[SrAg96] Srikant, Ramakrishnan; Agrawal, Rakesh: Mining Quantitative Association Rules in Large Relational Tables. Proceedings of the 1996

ACM SIGMOD International Conference on Management of Data, 1996.

[SuSi06] Sumathi, S.; Sivanandam, S. N.: Introduction to Data Mining and its Applications. Springer, 2006.

[WaBK05] Wang, Xiaomeng; Borgelt, Christian; Kruse, Rudolf: Mining Fuzzy Frequent Item Sets. 11th Int. Fuzzy Systems Association World. Congress (IFSA 2005, Beijing, China), 2005.

[WeCh99] Wei, Qiang; Chen, Guoqing: Mining Generalized Association Rules with Fuzzy Taxonomic Structures. Fuzzy Information Processing Society, 1999. NAFIPS, 1999.

[XieD05] Xie, Dong (Walter): Fuzzy Association Rules discovered on Effective Reduced Database Algorithm. The 14th IEEE International Conference on Fuzzy Systems, 2005.

[YaFB00] Yang, Cheng; Fayyad, Usama; Bradley, Paul S.: Efficient Discovery of Error-Tolerant Frequent Itemsets in High Dimensions. Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, 2000.

[Zade65] Zadeh, Lofti A.: Fuzzy Sets. Information and Control 8 (3) 338--353, 1965.

[Zade88] Zadeh, Lofti A.: Fuzzy Logic. University of California, Berkeley, 1988.

[Zade96] Zadeh, Lofti A.: Fuzzy Logic = Computing with Words. IEEE Transactions on Fuzzy Systems, Vol. 4, No. 2, 1996.

[ZeZy96] Zembowicz, Robert; Zytkow, Jan M.: From Contingency Tables to Various Forms of Knowledge in Databases. American Association for Artificial Intelligence, 1996.