Mining Fuzzy Association Rules on Large Numerical Data - A Data Mining System for NAWN

by

Zimpi Komo

A thesis
presented to the University of Manitoba
in partial fulfilment of the
requirements for the degree of
Master of Science
in
Computer Science

Winnipeg, Manitoba, Canada, 2003

THE UNIVERSITY OF MANITOBA

FACULTY OF GRADUATE STUDIES

*****

COPYRIGHT PERMISSION PAGE

MINING FUZZY ASSOCIATION RULES ON LARGE NUMERICAL DATA - A DATA MINING SYSTEM FOR NAWN.

BY

Zimpi Helen Komo

A Thesis/Practicum submitted to the Faculty of Graduate Studies of The University of Manitoba in partial fulfillment of the requirements of the degree of

Master of Science

Signature

Abstract

Mining numerical data has been and still is a burgeoning research area in computer science. This thesis introduces the problem of mining numerical data from very large databases, such as the data obtained from the Niagara Agricultural Weather Network (NAWN), using fuzzy logic and the data mining technique called association rule mining. The numerical data attributes are processed by converting them into categorical data or fuzzy sets using fuzzy logic. The converted numerical data can then be effectively mined using association rule mining algorithms. The first algorithm introduced is referred to as Fuzzy Apriori 1 (FA1), and is a simple implementation that allows each numeric datum to be represented by multiple fuzzy sets. This algorithm was found to take a lot of execution time. An improvement was made, resulting in a new algorithm referred to as Fuzzy Apriori 2 (FA2), which reduced the execution time by almost 60% but generated a lot of similar rules, compared to the rules generated by the FA1 algorithm. Further experiments were conducted using real-life datasets on both algorithms and a standard fuzzy algorithm for comparison. The experiments showed that both FA1 and FA2 generated more rules than the standard algorithm. Further experiments showed the effects of some user constraints on the number of rules generated by both the FA1 and FA2 algorithms. Lastly, the execution times of the FA1 and FA2 algorithms were shown to scale well with the number of records.

Acknowledgements

I would like to thank my supervisors, Dr. Peter Graham and Dr. John Anderson, for all their support, corrections and directions throughout this research. I am really grateful for all the time they sacrificed to help me through the research especially when they were both very busy with other engaging commitments.

I am also grateful to my husband for all the encouragement he gave me when I got tired, and for the strength and support he showed me that made the whole journey easier to bear in his absence.

I am grateful to my parents for their constant support and for making me what I am today. They are a indeed a source of inspiration and always will be, and I am proud to be called their daughter. I would also like to thank my brothers and in-laws, whose constant badgering of "When are you coming home?" showed me how much I am missed and loved.

I would also like to extend my gratitude to my course mates, especially Ye Li, Rajendra Singh and Vishnu Narayanasami, whose support and help have seen me through this research and have also made my stay in Canada a memorable one.

I am also grateful to all my committee members for having provided a good environment during my defense and for all their valuable suggestions.

I am above all grateful to God for the gift of life, family and friends.

Contents

1 Introduction **1**

1.1 Research Domain 3

1.1.1 Problem Definition 4

1.1.2 Problem Solution 5

2 Background and Related Work **8**

2.1 Background 8

2.1.1 Mining Association Rules 10

2.1.2 Fuzzy Logic 12

2.2 Related Work 28

2.2.1 Association Rule Mining 28

2.2.2 Fuzzy Logic 49

2.2.3 Fuzzy-Association Rule Mining 50

2.2.4 Summary of Related Work 59

3 Methodology **60**

3.1 The FA1 Algorithm 61

3.1.1 Fuzzification 62

3.1.2 Learning Fuzzy Association Rules 65

3.1.3 Rule Evaluation 68

3.1.4 Defuzzification 69

3.2 FA2 algorithm 74

4 Evaluation: Mining the NAWN Domain **77**

4.1 Experimental Process 78

4.2 Interest Rule Measure 79

4.3 Algorithm Scalability 89

4.4 Defuzzification 91

5 Conclusions and Future Work **94**

5.1 Conclusions 94

5.2 Future Work 97

5.2.1 System Improvement 97

5.2.2 Possible Extensions 98

List of Tables

2.1 Data items in a Database 11
2.2 Fuzzy Intersection 25
2.3 Fuzzy Union 25
2.4 Fuzzy Complement 26
2.5 1-itemset 30
2.6 2-itemset 30
2.7 3-itemset 31
2.8 Table showing the membership functions of 5 records and their cardinalities using FTDA 57
2.9 Highest cardinalities obtained for each attribute using FTDA 57
2.10 Generating the 2-itemsets by taking the intersections of 1-itemsets cardinalites using FTDA 58

3.1 Sample Database 63
3.2 Sample Membership Functions 64
3.3 Sample Fuzzy Database 64
3.4 Sample Database 74
3.5 Sample Membership Functions using FA2 75
3.6 Sample Fuzzy Database using FA2 75

List of Figures

2.1 Fuzzy Set (Tall) 16
2.2 Fuzzy Sets (Short, Average and Tall) 17
2.3 Line Curves [13] 19
2.4 S Curves [13] 20
2.5 Bell Curve and Overlapping Bell Curves [13] 21
2.6 Triangular Curve and Overlapping Triangular Curves [13] 22
2.7 Trapezoidal Curve and Overlapping Trapezoidal Curves [13] 23
2.8 Triangular Curves with Trapezoidal Curves as shoulders [13] 23

3.1 System Architecture 61
3.2 Center of Gravity Technique 71
3.3 Center of Gravity Calculation 72

4.1 Experimental Results Comparing the Number of Rules generated by Algorithms FA1 and FA2. 80
4.2 Experimental Results Comparing the Number of Rules generated by Algorithms FA1 and FA2 using a different set of 20,000 records. 83
4.3 Experimental Results Comparing the Number of Rules generated by the FA1 algorithm using different support and confidence values. 83
4.4 Experimental Results Comparing the Number of Rules generated the FA2 algorithm using different support and confidence values. 84
4.5 Experimental Results Comparing the Number of Rules generated by the FA1 and FA2 algorithms using different support and confidence values. 84

4.6 Experimental Results Comparing the Number of Rules generated by FA1 and FA2 using 100,000 records. 87

4.7 Experimental Results Comparing the Number of Rules generated by FA1 and FA2 using 100,000 records having the support and confidence values as the x-axis. 87

4.8 Experimental Results Comparing the Number of Rules generated from Algorithms FA1, FA2 and FTDA 88

4.9 Experimental Results Comparing the Time taken by the Algorithms FA1 and FA2 with up to 90,000 records 89

4.10 Experimental Results Comparing the Time taken by the Algorithms FA1 and FA2 with up to 1,000,000 records. 90

List of Algorithms

1 Fuzzy-Apriori Algorithm (FA1) 66

Chapter 1

Introduction

Modern technology provides an effective and efficient means of storing large amounts of data derived from almost every type of human activity including business, education and scientific research. However such large amounts of raw data by themselves are of little or no use to human understanding. They need to be analysed and processed into useful and relevant information to support effective decision-making. Data Mining is an effective method for accomplishing this analysis.

Data mining is a technique used for searching very large data sets to discover relationships among data variables. This procedure is performed to attain relevant information that can be used to make important predictions from such data [42]. Data mining has been shown, from past research, to be an effective tool that can be used to intelligently ascertain vast amounts of useful information from large databases [20, 49, 59].

A database can be considered to be a logically coherent collection of related data, where each datum is a given fact or piece of information with an implicit meaning that can be recorded [21]. For example, a database could contain all personal and professional information on all employees within a firm or a list of all available items in a

grocery store. A database is made up of one or more attribute values known as fields, such as *first name, last name, address, phone number, qualification, salary,* etc. A collection of field values such as:

first name- John,

last name-Cox,

phone number- 9517121,

might make up a record. A record is simply a set of related data attributes. For example, consider a grocery database with the following products for sale: *Apple, Grapes, Oranges, Heineken, Hot chocolate, Rice, Chicken* and *Beef.* The grocery database might store the code, name, price and quantity on hand for each product (i.e. four attributes in a 'product' record). For data mining, the available product in this database example is referred to as an *item.* Collections of items may be grouped to form itemsets and it is the available *itemsets* from a database that are the materials mined to discover useful relationships amongst the items. For example a relationship among the items in the earlier example could be: each time a customer buys a pound of chicken, that same customer is very likely to also buy a pound of beef; or each time a customer buys a bag of apples, the same customer is also likely to buy a pack of Heineken. Such a database could be manually maintained using a file system or, more commonly, be automated using database management systems such as Microsoft Access, SQL server, Oracle, Sybase, DB-2, etc. [21, 56].

Data mining is a cost-effective way of analyzing large amounts of data and as such has become an important field in computer science. The application areas for data mining extend from telecommunications [55], medical science [12] and business ventures [38] to agriculture [1]. In all application areas, the aim of data mining is to generate a 'summary' of a given database and present the results in a form that can be easily understood by a user.

1.1 Research Domain

As database size increases, the ability to manually process the data becomes almost impossible, especially when historical data must be analysed. The ability to acquire relevant information from such volumes of data also requires effective information-retrieval methods. For the purpose of this research, I will use real-world data collected by the Niagara Agricultural Weather Network to define the application domain.

The Niagara Peninsula, where 90 percent of Ontario's grapes and tender fruits are produced has a network of weather stations managed by a local company called the Niagara Agricultural Weather Network (NAWN). Their weather network is composed of 12 stations (sites), each connected by a modem to two data collection systems. The NAWN weather network is used to record long-term numeric climate data to produce a meso-climate map of the region. The information serves as a guide in choosing favourable planting sites for appropriate fruit varieties. The gathered information also needs to be transmitted in various ways, such as by fax or the Internet, to viewers and subscribers on a daily to weekly basis. From the numeric data collected by the NAWN stations, only a fraction of the potentially useful information is currently being extracted. Vastly more useful information, such as determining more precise regions with favourable weather conditions, could be determined if effective information-retrieval techniques and data mining were applied to the NAWN data.

My research is comprised of both mining the numeric data obtained from the NAWN stations and presenting the results in a format that can be understood by users[1]. An important fact to note is that dealing with numeric data introduces some inherent complexities when trying to apply data mining techniques. These will be explained in the

[1]Data mining will be used to determine potentially useful information and provide it in a format that will be human understandable.

next section.

1.1.1 Problem Definition

In general, there are two types of data used in data mining: numeric (or quantitative) data and categorical data. Numeric data is any data that can be represented by quantitative values, which can be observed and specified on a numeric scale. Numeric data are used to represent such things as temperature, speed, volume, voltage, humidity, etc. Numeric data have an implicit order in which the values are represented. Categorical data on the other hand, are represented by symbols and not numeric values, for example *address, postal code, name, title, sex,* etc. Categorical data do not have the characteristic of implicit order like numeric data and as such can not be mapped directly to numeric values. For example if a customer at a grocery store buys a pound of beef and a jar of milk and another customer buys five pounds of beef and a jar of milk, 'categorically' they both bought the same types of items but 'numerically' each customer made different purchases.

Numeric data provides a way of capturing additional details that would be overlooked when using only categorical data. Once numeric data have been efficiently analysed, the results obtained by data mining can be more precise and explicit when compared to those obtained using categorical data. The need for numeric data depends on the interests of the user of the system. For instance if a sales manager is interested in patterns of items being sold in his store rather than the numbers of items being sold, mining his data using categorical algorithms would be just as effective and less costly than using numeric algorithms. However for more numerically sensitive situations, such as scientific experiments that commonly deal with numeric values, categorical data mining techniques would not be as effective as numeric data mining techniques.

Most algorithms available for data mining have been designed to suit categorical data items with no particular attention given to dealing with quantitative or numeric data. Many databases, however, contain a mixture of categorical and quantitative data types (for example, sales made from a grocery store). Available data mining techniques require "clear-cut boundaries", which cannot always be obtained from numeric data. Clear-cut boundaries mean that there should be a clear distinction between items, implying that an item cannot be partly one value and partly another. For example, taking the following possible numeric values in a database: *20, 21, 21.5* and *22*, if one decides to use clear-cut integer boundaries: *20, 21* and *22* , they would have the problem of where to place the value *21.5*. This clear-cut boundary problem means that intervals have to be defined for quantitative data, such as [20, 30], [30, 40], etc. These intervals create sharp boundaries and as such a method of transition from one value to the other has to be provided. If one decides to solve the problem by adding more overlapping intervals, this would create another problem by presenting a wide range of values to be analysed such as [20, 24], [23, 27], [26, 30], etc.

In summary, while categorical data can be mapped to boolean values such as *yes* and *no*, numeric data require interval mappings for a given database attribute and hence may create large numbers of itemsets and therefore take more computational time. This implies that numeric data have to be processed to avoid clear-cut boundaries and also, that computational techniques have to be employed to avoid the generation of large numbers of itemsets.

1.1.2 Problem Solution

In this thesis, I will combine *fuzzy logic* with a data mining technique called *association rule mining* to mine the numeric data collected by the NAWN stations.

Fuzzy logic deals with the clear-cut boundary problem by eliminating hard boundaries and providing a clean conversion from one numeric value to another. Fuzzy logic allows the use of discrete classifications without introducing discontinuities. This means that multiple fuzzy results from different regions in a fuzzy system can be used simultaneously to obtain a single output. Fuzzy logic also deals effectively with the problem of large numbers of numeric intervals by creating fewer itemsets to be mined. Fuzzy logic is therefore a useful tool for systems that are difficult to analyze due to their continuous nature. It also allows numeric values to be represented by linguistic constructs such as *cold, hot, mild*, thus making them easier to understand and easier to both implement and design. Fuzzy logic also has the added advantage of easily being combined with other methods to solve complex problems. Since fuzzy logic converts[2] and reduces the number of itemsets generated, suitable categorical data mining techniques can then be used to mine the required data.

Association rule mining is a classical categorical data mining technique that has been discussed extensively in the data mining research literature. It is a technique that searches the entire database for every possible relationship between each group of data items. Unlike other techniques such as the classification technique, which searches for particular relationships among data items, association rule mining performs an exhaustive search. It is a technique that is easy to understand and implement. It is also a flexible technique that can be modified in a number of ways to improve its performance and results.

Fuzzy logic and the association rule mining techniques have advantages in different areas of research and by combining them I hope to create a data mining system that exploits the advantages of both techniques.

The rest of this thesis is organised as follows: Chapter 2 presents background material and related work. In Chapter 3 the methodologies used in the thesis are described. Chap-

[2]changes numeric values to linguistic terms such as low, medium and high

ter 4 discusses the experimental process used to assess the methodology and presents the results obtained on the real-world NAWN data. The thesis concludes in Chapter 5 by providing conclusions and discussing possible directions for future work.

Chapter 2

Background and Related Work

Data mining techniques have been used extensively over the last decade and provide a range of ways to mine different types of data. The aim of data mining research is to produce efficient algorithms that can effectively mine very large data sets. Data mining combines different techniques such as machine-learning, statistical analysis and database methods to effectively analyse historical data in a database. To understand fully the research presented later in this thesis, an understanding of the basic concepts of data mining and common data mining techniques are required.

2.1 Background

Data mining requires a *processing* technique, which simply mines the available data and a *representation* technique, which represents the results of the mined data in an understandable format for users. Some available techniques that could be used for the *processing* phase include: association rule mining [53] (searches for all the available itemset relationships in a database that satisfy certain given conditions); classification [45, 61]

(searches to determine if some set of pre-defined itemset relationships do exist in a database); decision trees [71] (represents a series of IF...THEN relationships amongst itemsets based on observations obtained from a database that can be used to make predictions); neural networks [25] (an information processing model based on neural information processing, which has the ability to learn how to do tasks based on the data given for training or initial experience) and clustering [57] (the process of partitioning a set of data into meaningful subclasses called clusters with the aim that each cluster contains similar elements yet is substantially different from the other clusters). Two possible *representation* techniques are fuzzy logic [13] (represents numeric values in linguistic terms that can be easily understood by users) and frequent closures [9] (which provides descriptive representations of binary tables for rule mining techniques). I selected *mining association rules* as my processing technique and *fuzzy logic* as the representation technique for the following reasons:

- They are both classical techniques that have been used successfully in the past in different application areas.
- They each have individual advantages in different areas and can easily be combined with other techniques (including one another). For example fuzzy logic has been combined with such techniques as genetic algorithms [14], while association rule mining has been combined with other techniques including clustering techniques [22].
- They both have available and familiar algorithms that can be easily understood and used.
- Fuzzy logic provides an easy way of representing complex numeric data and provides clarity and simplicity to users.

- Association rule mining techniques are easy to understand and implement, while fuzzy logic can easily be modified and changed according to system specifications.

These techniques are integral to the remainder of this thesis and are described in the following subsections.

2.1.1 Mining Association Rules

Mining association rules was first introduced by Agrawal, et al [53]. Mining association rules is a technique that retrieves all possible patterns from very large databases that match some pre-defined conditions, by which the results obtained can be used for future predictions. An example of an association rule is '40% of women who drive do not have jobs, 20% of all entries contain these two results together'. This association rule can be expressed in the form:

Women : drive $\Rightarrow$ women : no jobs [0.2, 0.4]

Generally, an association rule has the form:

$\mathbf{A} \Rightarrow \mathbf{B}$ [c, s] and $\mathbf{A} \cap \mathbf{B} = \emptyset$

Where **A** (women : drive) and **B** (women : no jobs) are items in a database. **A** is known as the *antecedent* and **B** is known as the *consequent*, with support of s (0.4) and confidence of c (0.2). The support value is the number of times the rule appears with respect to the entire dataset, while the confidence value is the number of times the consequent occurs with respect to the antecedent. During the process of mining association rules, the numbers of rules generated are usually many and often uninteresting to the user. Specifying the 'interestingness' of a rule is accomplished by providing support and confidence values. The support and confidence values are specified by the user of the system. The numbers of rules generated from an association rule mining algorithm varies depending

on the support and confidence values. For example, it is possible for a user to be more satisfied with the rules obtained at a higher support value since this shows that those rules occur more often in the database than with a lower support value. Association rule mining focuses on finding rules that have a minimum specified support and confidence level. For example, the support and confidence of the itemsets (**A**) and (**B**) for a rule from a database of **T** records can be calculated as follows:

$$s(A \cup B) = \frac{\#(\mathbf{A} \cup \mathbf{B})}{\mathbf{T}}$$

$$c(A \cup B) = \frac{\#(\mathbf{A} \cup \mathbf{B})}{\#(\mathbf{A})}$$

where '#' is the total number of records containing such itemsets.

Consider the following example based on Table 2.1.

No.	item
1	Milk, Meat
2	Fruit, Rice, Milk, Eggs
3	Fruit, Jelly, Rice, Meat
4	Fruit, Milk, Rice, Meat
5	Jelly, Milk, Rice, Meat

Table 2.1: Data items in a Database

Given the rule **Rice and Meat ⇒ Fruit**, the support (s) and confidence (c) are determined as follows:

$$s = \frac{\#(Rice, Meat, Fruit)}{\#(T)} = \frac{2}{5} = 0.4$$

$$c = \frac{\#(Rice, Meat, Fruit)}{\#(Rice, Meat)} = \frac{2}{3} = 0.66$$

Association rule mining is typically divided into two phases: picking out all items (or itemsets) that have a support level above the minimum required support level, and discovering interesting rules from such itemsets [11, 33, 53, 54].

2.1.2 Fuzzy Logic

Fuzzy logic was first proposed by L. A. Zadeh [65]. It is used to model real-world systems that are continuous, vague, and ambiguous. Fuzzy logic deals with 'fuzzy' events such as 'the temperature is low' or 'the man is tall' [2]. Taking 'the man is tall' as an example, how can we determine when a person is tall? If we infer that a person of 6 feet is tall, do we then imply that a person of 5 feet and 11 inches is short? The purpose of fuzzy logic is to interpret the fuzzy condition in a rule, and provide a continuous range of truth-values within 0 and 1 (i.e. [0, 1]), called the confidence factor or membership function. Fuzzy logic produces results such as 'a person of 5 feet and 10 inches is tall with a confidence factor of *0.7*'. A 'crisp'[1] database query for employees that are old

[1] A crisp value is a precise numerical value such as 2 or 0.3

and earn below $50,000 per year might be based on the following:

Old $\Rightarrow$ age $\geq$ 65years

Low salary $\Rightarrow$ salary $\leq$ $50,000 per year

A crisp search would then fail to identify an employee of age 64 years. Since different users have different ideas of what old should be and some would expect such an employee to be included in the search, this presents a problem. This is one of the complexities involved with analyzing quantitative or numeric data. There are no clear-cut boundaries for analyzing numeric data, since numeric data are continuous. These shortcomings can be handled through the use of fuzzy logic.

Zadeh [65] based his work on the following claims:

- There are classes that cannot be represented by two-valued membership functions because instances of those classes occur at a boundary such as *handsome man* and *short woman*. These classes have different instances such as handsome, man, short and woman.
- Such classes occur in a continuous grade like small, smaller, smallest, etc.
- The sources of these imprecisions occur from the absence of a clearly defined standard rather than a presence of random variables thereby causing the notion of 'fuzzy set' to be non-statistical.

These concepts caused Zadeh to propose the concept of a *fuzzy set*. He defined the relations between fuzzy sets to be *equal and containment* and the operations that could be performed on such sets to be *complement, union* and *intersection*. He tried to attain the membership function of a class from the individual membership functions of the instances that compose it by applying fuzzy operations. For example, the *short man*

membership function can be calculated from the intersection of the class *short* and *man*. Zadeh proposed two ways of calculating the membership functions of those constituting classes [40]. He assigned values to instances in a class for categorical attributes and proposed differentiable functions for numeric attributes. In both situations, preconditions have to be satisfied. In the former, for example, the domain for the instances must be finite, and for the latter there must be a measurable property. Even with these, there were still some questions to be answered such as: "How to get the measurable property?" "How to determine which instances have a higher grade than others?", and "How are these grades measured?" Zadeh gave the following suggestions [40, 66, 67, 68]:

- He developed the concept of *the Principle of Incompatibility*, which explains that 'as the complexity associated with a system increases the human ability to make concise remarks about its behaviour decreases'.

- He stated that a membership function simply maps a quantitative value onto a linguistic value, so as to approximate the quantitative value. For example 'He is tall' could be an estimate of 'He is 5 feet 7 inches tall'.

- He also proposed the *possibility/probability consistency principle* which states that the less an event occurs, the less its probability but not the other way around. For example, if the membership function of 'He is tall' is 0.6, this does not imply that this case always occurs 60% of the time.

- Determining the membership functions of instances that make up a compound class is context-dependent, implying that there is no single, general method to determine such functions. This means the fuzzy membership function depends on the type of system, its context and its domain, and not necessarily on a particular scientific work. For example 'she is young' would depend on the domain or con-

text in which it is being used. If the individual in question were in high school or in a retirement home, the results would be very different.

Methods for determining membership functions vary from one system designer to another, and their designs depend on the particular domain in which they operate as reported by Cox [13]. There are different ways of deriving fuzzy rules, some of which are: based on experts' experiences or control engineers' knowledge in the specified field or domain; based on domain operators' control or actions; based on the model of the process itself and based on learning from the domain. Fuzzy rules have been an important alternative in building complex systems that would otherwise likely have been unsuccessful using other tools [13].

Fuzzy logic has also been combined with various AI techniques including neural networks [47], clustering techniques [64], frequency in system identification [16] and belief measure in a system [17]. Fuzzy logic has also been used in other complex systems such as pattern recognition [7, 50], database query processing [58], time-series analysis [60], image databases [48] and meteorological systems [14]. A brief introduction to some of the concepts associated with fuzzy logic is now presented.

Fuzzy Sets

Fuzzy sets such as *tall, short, fat, long,* etc. are linguistic terms that are used to represent a domain or range of truth. For example, if we take the fuzzy set *tall*, the domain for this fuzzy set would represent the range of values that the set *tall* applies to. For example the range of *5 feet and above* could be characterized as *tall* and would represent the range of values for the fuzzy set tall. The degree of truth for any value in this domain will be given by the membership function. For example *5 feet* would have a membership value less than *6 feet* since *6 feet* is obviously taller and more agreeable with the fuzzy set *tall.*

In Figure 2.1, the horizontal axis represents the domain of the fuzzy set *tall*, while the vertical axis represents the membership function.

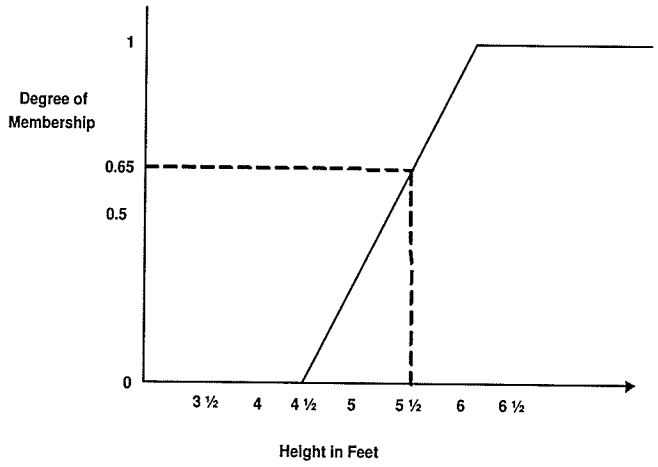

Figure 2.1: Fuzzy Set (Tall)

In Figure 2.1, the membership value of a person of height $5\frac{1}{2}$ feet is given as *0.65*. In general the membership function for any fuzzy set ranges from 0 (not a member of the set) to 1 (full member of the set). A membership value of *0.65* implies that $5\frac{1}{2}$ feet is a fairly strong member of this fuzzy set, though not a complete member.

The membership function is not a *one-to-one* map with the domain. This is because there could be more than one fuzzy set on a graph, which would lead to more than one curve on the graph as shown in Figure 2.2. Multiple curves on a graph lead to membership functions being mapped from several domain values.

A fuzzy set may be represented using three components: a monotonically increasing horizontal axis, which corresponds to the domain, the vertical axis indicating the membership or truth value, and the connection from a domain value to its membership value - the membership function.

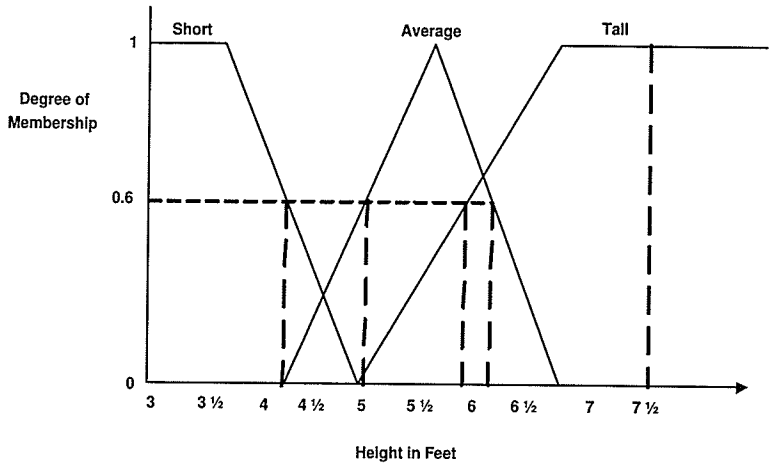

Figure 2.2: Fuzzy Sets (Short, Average and Tall)

The fuzzy domain is a set of real numbers (that can be negative or positive) in increasing order from left to right. The values for a fuzzy domain represent the complete range of the fuzzy set, which depends on the particular system being designed. A fuzzy set for *young* could range from 0 to 2 for a child-care system, while it could range from 60 to 80 for a home care centre for the elderly. A domain usually has an upper and lower limit as shown in Figure 2.1. However this may not always be the case as some fuzzy sets can have unbounded upper or lower limits as will be seen later.

Another term used in fuzzy set theory is the *Universe of Discourse*. This is a domain which consists of the lowest to the highest values in a model for graphs with more than one fuzzy set. For example in Figure 2.2, *Height* is a domain that can be broken down into three fuzzy sets: *short, medium* and *tall*. The universe of discourse for this example is from *3 feet* to $7\frac{1}{2}$ *feet*. The fuzzy sets that make up the universe of discourse may not be symmetric and may overlap. The overlap within these fuzzy sets influences the results obtained from the system and is therefore an important aspect of the system design. The scales used in the various fuzzy sets also affects the result obtained from the system itself

and plays a significant role in generating the required results.

Fuzzy sets and membership functions can be viewed using a graphical representation of the system. These representations vary between systems and also between domain types and are shown as curves.

Fuzzy Curves

According to Cox [13], there are no standard curves for fuzzy set graphs. The curves depend on the particular system being built and the mappings of the domain to the membership functions. However there are a few common curves. These are: lines, **S**-curves, bell, triangular and trapezoid curves.

Line Curve: Represents fuzzy sets that are usually unbounded. The curve is made up of a straight line and can be considered the simplest fuzzy set curve. The fuzzy set curve could be either in increasing order from a low domain value to a high value, or sloping downwards from a low domain value to a higher value as shown in Figure 2.3.

Sometimes a line curve is a straight-line curve at almost 45 degrees, in such cases the membership functions are directly proportional to the distance from origin.

S- Curve: Represents fuzzy sets that are unbounded just as the line curve does, but **S**-curves represent more complex systems such as those arising from time series analysis. The curve is defined by three parameters: its zero membership value, its full membership value and its point of inflexion, which is the point at which 50% of the domain values have a strong truth membership value. An **S**-curve can either curve from complete membership down to the maximum value of the domain or can curve from the lowest value of the domain up to the highest value of the domain. Examples are given in Figure 2.4.

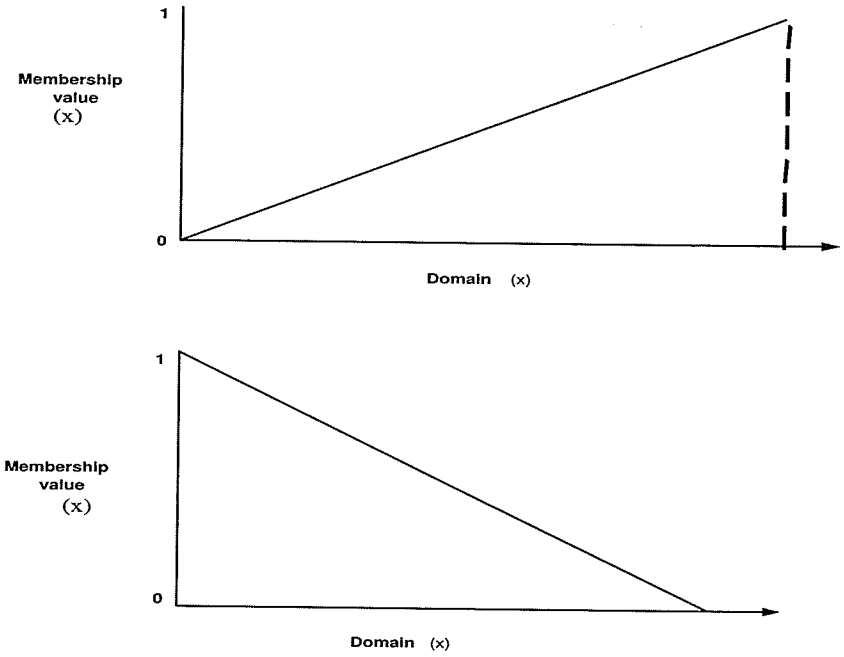

Figure 2.3: Line Curves [13]

Bell Shape Curve: The bell shaped curves have a central point in their domain which has a complete membership function. They represent fuzzy sets that have their domain distributed around a central value. For example values around *medium* height could be in the range from [5, 5.7] feet. The bell-curves are usually two-like S-curves facing each other. The three most common bell shaped curves are the PI, beta and Gaussian curves [13]. The PI curve is a bounded curve and more widely used, while the beta and Gaussian curves are unbounded. S-curves are sometimes placed at the edges of Bell-shaped curves for regions that do not change but have a continuous value.

Triangular Shaped Curve: Triangular curves are simpler to understand and visualize than bell curves. The triangular model has a peak point (at which the membership

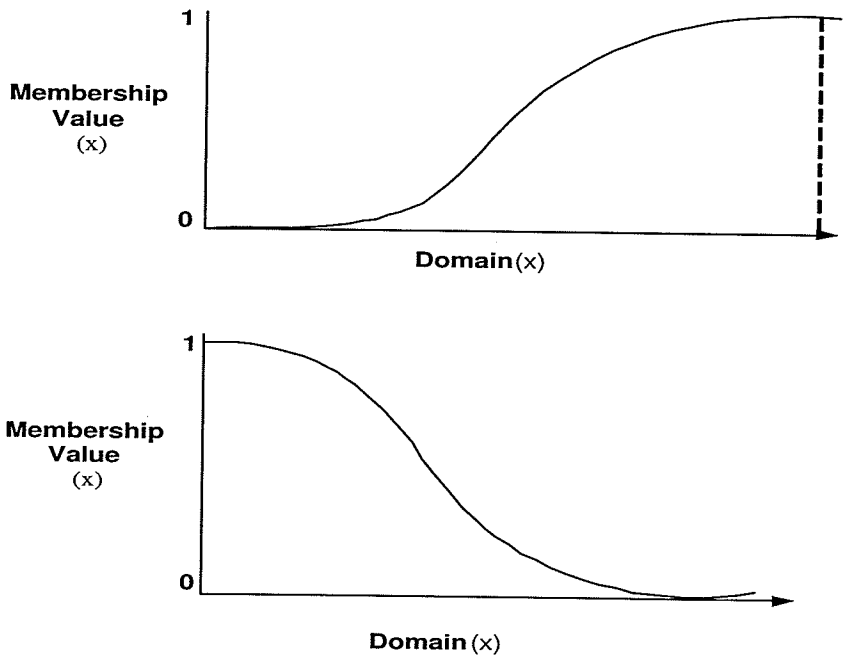

Figure 2.4: S Curves [13]

function is complete) from which there are straight slants downwards on either side of the point to where the membership value is zero, as shown in Figure 2.6.

The triangular curve has three parameters: a, which represents the initial rising point of the triangle curve, b representing the peak of the curve and c, which represents the final descent point of the triangular curve. These points occur in the order ($a < b < c$). The membership function for a point **x** in the fuzzy domain can be determined using the equation [19]:

$$\eta(x) = max\left(min(\frac{x-a}{b-a}, \frac{c-x}{c-b}), 0\right)$$

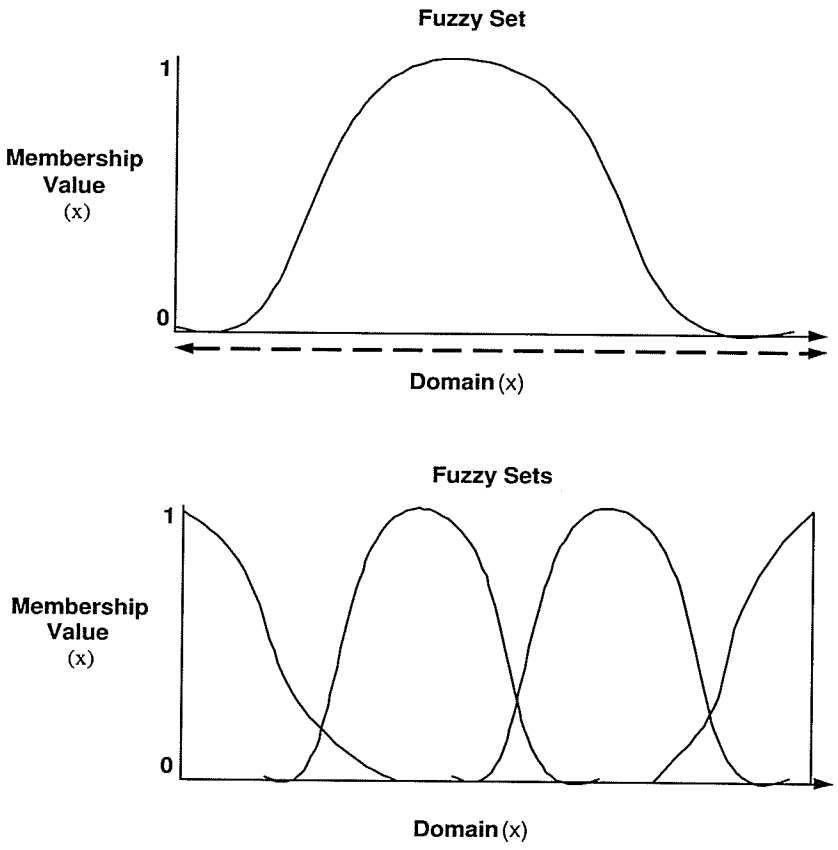

Figure 2.5: Bell Curve and Overlapping Bell Curves [13]

Trapezoid Shaped Curve: The trapezoidal curves have a range of domain values grouped together in clusters thereby giving them a plateau-like curve as shown in Figure 2.7. They are an extension of the triangular fuzzy curve with a plateau instead of a peak.

The trapezoidal graph has four parameters: a represents the start of the upward slope, b the start of the plateau, c the end of the plateau and d represents the end of the downward slope. The order of these values are ($a \leq b < c \leq d$). The membership value of a domain value x can be determined using [19]:

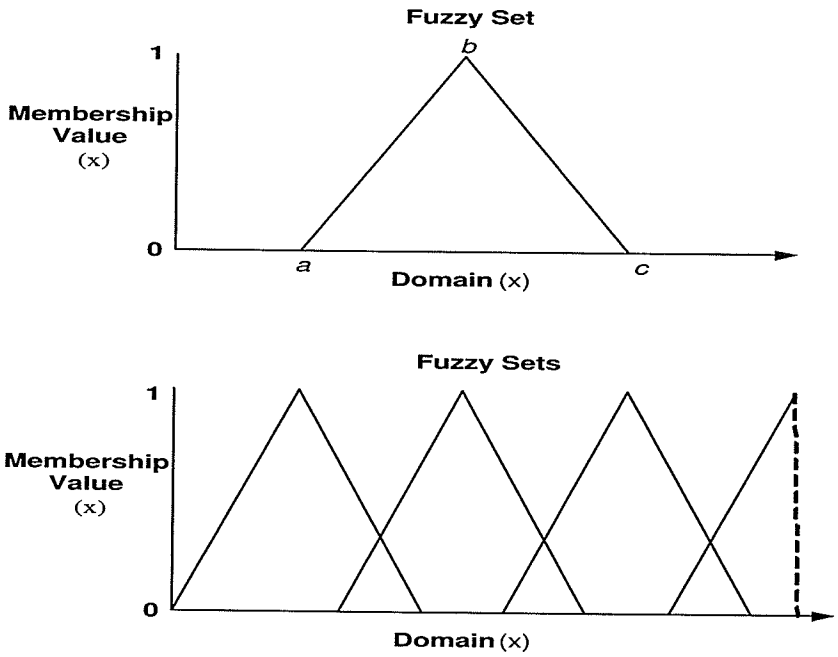

Figure 2.6: Triangular Curve and Overlapping Triangular Curves [13]

$$\eta(x) = max\left(min(\frac{x-a}{b-a}, 1, \frac{d-x}{d-c}), 0\right)$$

Trapezoid curves are sometimes called shoulder-fuzzy sets [13], because they can be placed at the edges of triangular fuzzy sets for regions that do not change but which have a continual value (i.e. unbounded limits in some cases). For example a height of 6 feet could be termed as *tall* with a membership value of 1 and any height greater than this would still have a complete membership value. As such the edge of such fuzzy sets can be represented by a half trapezoidal curve as shown in Figure 2.8.

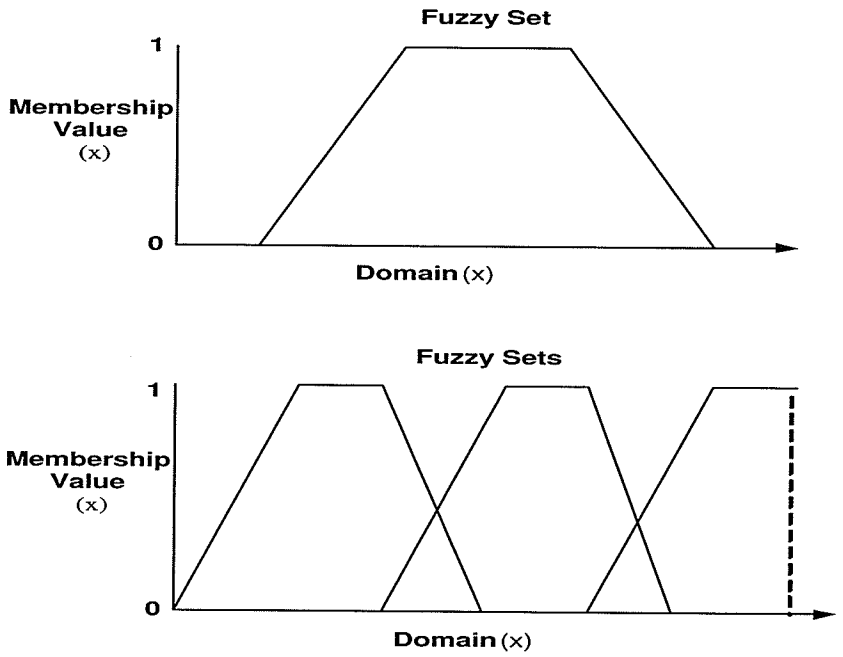

Figure 2.7: Trapezoidal Curve and Overlapping Trapezoidal Curves [13]

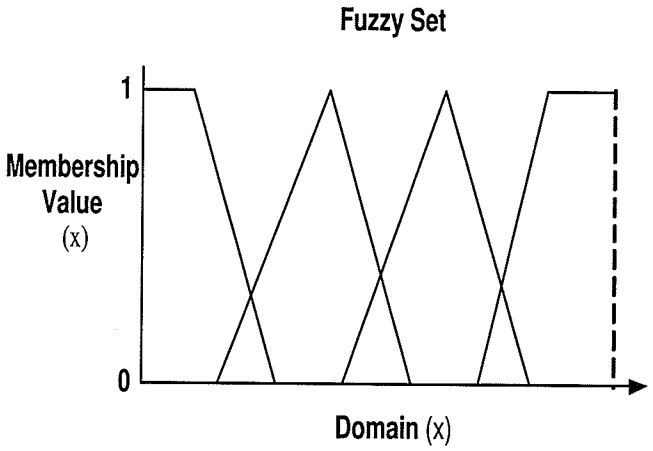

Figure 2.8: Triangular Curves with Trapezoidal Curves as shoulders [13]

After the various fuzzy sets and curves for a system have been determined, fuzzy operators are then applied to produce the required result. The next section briefly discusses these operators.

Fuzzy Operators

Zadeh [65] proposed three operations to be performed on fuzzy sets, they are: *complementation, union* and *intersection.* These fuzzy operators are only applied to the membership functions, whose values are above zero.

Intersection: In conventional boolean operations, the intersection of two sets takes common elements from both sets, which is equivalent to the logical *AND* operator. The intersection operator in fuzzy logic takes the minimum of the truth-values of the two membership functions. The intersection of fuzzy sets **A** and **B** is denoted as **A** $\cap$ **B**. The membership function is computed as [65]:

$$\eta_{A \cap B}(x) = min\{\eta_A(x), \eta_B(x)\}, \forall x \in X$$

For example the intersection of the fuzzy membership functions *tall* and *weight* is shown in Table 2.2.

Union: The union of two sets using boolean operations is determined by the logical *OR* operator. The fuzzy union operator is performed by taking the maximum value of two membership functions. The union of fuzzy sets **A** and **B** is denoted as **A** $\cup$ **B**. The membership function is computed as [65]:

$$\eta_{A \cup B}(x) = max\{\eta_A(x), \eta_B(x)\}, \forall x \in X$$

An example using the fuzzy sets *tall* and *weight* is shown in Table 2.3

No.	Tall	Weight	Tall and Weight
1	0.20	0.80	0.20
2	0.00	0.20	0.00
3	0.35	0.45	0.35
4	0.50	0.90	0.50
5	0.80	1.00	0.80

Table 2.2: Fuzzy Intersection

No.	Tall	Weight	Tall or Weight
1	0.20	0.80	0.80
2	0.00	0.20	0.20
3	0.35	0.45	0.45
4	0.50	0.90	0.90
5	0.80	1.00	1.00

Table 2.3: Fuzzy Union

Complementation: The fuzzy complement is different from the boolean complement in that the fuzzy complement inverts the membership value by subtracting it from 1. The complement of a fuzzy set **A** is denoted as $\neg$ **A** and the membership function is computed as [65]:

$$\eta_{\neg A}(x) = 1 - \eta_A(x), \forall x \in X$$

For example taking the fuzzy set *tall* we have the fuzzy complement shown in Table 2.4.

Other fuzzy operators have also been defined (as discussed by Cox [13]), which are somewhat different from Zadeh's work. Cox [13] believes that the operators used for

No.	Tall	Not Tall
1	0.20	0.80
2	0.00	1.00
3	0.35	0.65
4	0.50	0.50
5	0.80	0.20

Table 2.4: Fuzzy Complement

a particular system depend on the model in question. In other words, he claims that the operators should be chosen based on the system domain and the effect of the results obtained when such operators are used. However he agrees that Zadeh's operators are the fundamental tools for fuzzy systems and advises that a system should first be designed with Zadeh's operators and then modified to suit the domain.

Defuzzification

When using a fuzzy system, a user inputs a crisp value and expects a crisp value in return. After the necessary fuzzy operators and processing have been applied to a system, a crisp result must be returned after a process of *defuzzification* has been applied.

There are no fundamental principles for defuzzification that are obtained from fuzzy theory, and so defuzzification is largely ad-hoc. Defuzzification can best be explained with an example. Consider the fuzzy rule: if **A** is **Y** then **B** is **Z**. To determine the actual crisp value of the fuzzy sets **B** and **Z**, one has to employ defuzzification. The fundamental problem with defuzzification is "how do we best represent a single crisp value from a set of fuzzy rules?" There are several techniques for defuzzification including: 'centre of gravity', 'maximum of output', 'hyperspace decomposition comparisons', 'the prepon-

derance of evidence technique', 'average of maximum values', 'average of the support set', 'far and near edges of the support edges' and the 'centre of maximums' [13].

The *Center of gravity* is the weighted average of output membership functions of a fuzzy rule. The centre of gravity method is a technique that determines the balance point of the fuzzy sets by finding its centre of gravity. The centre of gravity technique behaves similar to the Bayesian estimates technique [13] in that the value selected is supported by the knowledge accumulated from each rule. Changes in the fuzzy set's model result in smooth changes in the results.

The *Maximum output* is obtained from the domain value that has the largest membership value. If the value lies on a plateau, then other techniques such as finding the average of the values making up the plateau or finding the centre of the plateau have to be employed to determine the fuzzy set's mid-point. This technique is sensitive to single rules that dominate the fuzzy sets and the results tend to switch abruptly from one fuzzy model to the next as the shape of the fuzzy sets change [13].

The *Average of maximum values* finds the mean value of the fuzzy region. If the value is on a plateau, then the average of the plateau is calculated and obtained. If the value is a single point then that value is returned. For the related techniques, the *Average of the support set* takes the average of the entire fuzzy set's values for a particular fuzzy region, the *Far and near edges of the support set* take values at the right or left of the fuzzy set edge and the *Center of maximums* takes the mid point of the two highest plateaus in a multi-plateau fuzzy region. If the fuzzy region has no plateau then the maximum point in the fuzzy region is used. However if there is a single plateau the centre of maximums technique then switches to the maximum output technique [13].

There are no fundamental principles for defuzzification, but Kosko [35, 36, 37] considered a combination of defuzzification methods such as the *centroid defuzzification*

and the *additive aggregation technique*, which formed the basis for his so-called 'Standard Additive Model' (SAM). His research showed that fuzzy systems employing the SAM model can approximate any continuous function, thereby providing a mathematical foundation for the defuzzification process [13]. There are also other less general defuzzification methods available but they will not be discussed in this thesis.

2.2 Related Work

There are several algorithms commonly used for general data mining. These include: Apriori [52], Partition [20], FP-growth [63], Eclat [43], SPRINT [45], CBA [62], RainForest [61] and DHP [49]. Using association rules to mine numeric data has been a topic of much recent research in computer science [52]. Most research on numeric databases have used classification rules such as in RainForest [61]. A combination of classification rules and association was also done by Liu, et al [62]. A brief description of some of the most important algorithms is now provided.

2.2.1 Association Rule Mining

There are various approaches and algorithms available for mining association rules. Some association rule mining algorithms involve finding all itemsets above a minimum required value and using those itemsets to generate the desired rules. The general model for association rule mining can therefore be divided into two phases: discovering the large[2] itemsets and using the large itemsets to generate the association rules [20, 49, 52]. These algorithms include: Apriori, SETM, DHP, AIS and Partition. Once the large sets

[2]Itemsets having a support level above the minimum support are called large or frequent itemsets, and those below are called small itemsets

have been generated, they have to be converted into useful rules for the users of the system. For example given **WXYZ** and **YZ** as itemsets, a rule such as **YZ** $\Rightarrow$ **WX** could be generated. This rule simply means that at all times **YZ** appears, **WX** is also expected to appear.

Other algorithms such as the one proposed by Zaki, et al [70] use parallel computing techniques to accelerate the computation of the required association rules. The FP-growth, SPRINT, CBA and RainForest algorithms develop a tree structure that is used in mining the data and producing the required association rules.

Apriori

Apriori was the first scalable algorithm designed for association rule mining and was shown to be superior to earlier data mining techniques [27, 49]. Mining association rules using the Apriori algorithm was first introduced by Agrawal, et al [4]. Apriori was an improvement over the earlier AIS [4] and SETM [31] algorithms (which will be reviewed later). The Apriori is presented first since it is the focus of this thesis.

The Apriori algorithm searches for large itemsets during an initial database pass and then uses its result as a seed for discovering other large itemsets during subsequent passes. These are called *candidate* itemsets. At the end of each pass, the *support* for the generated candidate itemsets are determined. Those itemsets with support values less than the given *support* are removed and the ones left become the seed for the next round of candidate set generation [42]. An itemset with one item is known as a *1-itemset* and an itemset with k items is known as a *k-itemset*. The Apriori algorithm requires that, given a set of data, an initial scan of the entire database be made to generate the single itemsets, called the *1*-candidate sets, with a support level equal to or more than the specified minimum support. After generating these itemsets they are then joined together to form

the next collection of candidate sets, called the 2-candidate sets. The database is again scanned to count the support level of the 2-candidate sets. Any set having a support below the specified minimum support is pruned off. This process is repeated for each new collection of candidate sets. The algorithm terminates when no more candidate sets can be generated. This eventually leads to a large k-candidate set or k-itemset. The Apriori algorithm is also known as a bottom-up algorithm because of its mode of operation.

An example based on the itemsets shown in Table 2.5 to Table 2.7 assuming a minimum support of **3** shows the result of the various phases of the Apriori algorithm.

items	**count**
Milk	4
Rice	4
Jelly	2
Meat	4
Fruit	3
Eggs	1

Table 2.5: 1-itemset

items	**count**
Milk, Meat	3
Milk, Fruit	2
Milk, Rice	3
Meat, Fruit	2
Meat, Rice	3
Fruit, Rice	3

Table 2.6: 2-itemset

items	count
Milk, Meat, Rice	3
Meat, Rice, Fruit	2

Table 2.7: 3-itemset

To illustrate the processing, note that in Table 2.6, all pairs (i.e. 2 candidate sets) are shown except those involving 'Jelly' and 'Eggs' (from Table 2.5 which were pruned off because their support in the first pass was less than 3.

The so-called 'join' operations that are performed when advancing to later passes of the Apriori algorithm combine k-itemsets with each other to form the next level's $k+1$-itemsets. They are based on the large itemset property which states that: "subsets of a frequent itemset must themselves be frequent" [43]. An example of a join and prune operation is as follows: given a set of itemsets *(a b c), (a b d), (a c d), (a c e), (b c d)*, after a join operation the following large itemsets are generated *(a b c d)* and *(a c d e)*. After the prune operation *(a c d e)* would be removed since the subsets that can be obtained from it consist of *(a c d), (c d e), (a d e)* and *(a c e)*, and *(a d e)* is not a large subset (or was not in the initial set of large itemsets). This makes *(a b c d)* the only large set left.

Variations on the Apriori algorithm such as AprioriTID [52] and AprioriHybrid [52] have also been proposed. AprioriTID works like Apriori but uses the generated candidate sets to count the support instead of re-scanning the database, thereby avoiding expensive I/O operations. A method of encoding the candidate sets from previous passes is used to count the support. In subsequent passes the size of this encoding becomes much smaller which saves itemset access cost. The AprioriTID algorithm also discards records in subsequent passes that do not contain large itemsets in its current pass, thereby decreasing the number of subsequent records in the database to be scanned. AprioriTID was reported to be more efficient than Apriori whenever the itemsets generated by AprioriTID

were able to fit into memory [20, 52].

AprioriHybrid [52] is a hybrid of Apriori and AprioriTID. It uses Apriori for its initial passes and switches to AprioriTID when it expects that the sets generated will be able to fit into memory. AprioriHybrid was shown to be more efficient than Apriori in most cases except when it switches on the last pass [52]. This is because it incurs an overhead due to switching from one technique to another with no significant gain since no more large itemsets are generated on the final pass. The major problem with AprioriHybrid is determining when a switch should be made from one technique to the other.

AIS

The AIS algorithm [4] generates its itemsets 'on-the-fly' during each pass over the database. This means that candidate itemsets are generated by performing a join operation with large itemsets from previous passes and itemsets in the database, unlike the Apriori algorithm which performs its join operations only amongst generated large sets. The AIS algorithm results in new itemsets that are not large, thus resulting in unnecessary generation of candidate sets and significant overhead. Though Apriori generates small itemsets during its second pass, this is compensated for during subsequent passes, which generate primarily large itemsets. The rules generated by the AIS algorithm also have only one item in the consequent while the Apriori algorithm allows multiple items in its consequent. For example, the AIS algorithm can generate such rules as $\mathbf{YZ} \Rightarrow \mathbf{W}$ and $\mathbf{AB} \Rightarrow \mathbf{C}$, but not $\mathbf{YZ} \Rightarrow \mathbf{WX}$ [4, 52]. AIS has been shown experimentally to perform more efficiently than SETM does [52].

SETM

SETM (Set-Oriented Mining) [31] algorithm operates very similar to the AprioriTID algorithm [32], so it can be inferred that the AprioriTID algorithm might be based the SETM algorithm. The SETM algorithm also generates its itemsets 'on-the-fly' in a fashion similar to the AIS algorithm but it is designed to be used with SQL (Structured Query Language). SETM's major problem occurs from the size of its generated large candidate sets, which requires the sets to be written to a disk and sorted externally. Tests performed showed that SETM often takes too long to execute with respect to the Apriori algorithm. AprioriTID also generates large sets like SETM but the difference is that it has fewer itemsets than SETM and it does not require sorting to count its support [31, 52].

Partition

Sarasere, et al [20] presented another method of mining association rules. Their technique requires that the database be scanned twice. They explored the idea that given a small set of potentially large itemsets, their support could be counted in a single scan of the database and the actual large itemsets could then be generated. This idea is true only if the given set contains all the large sets. Their work was also based on the notion that the reason a database needed to be scanned multiple times was due to the fact that the number of possible large itemsets generated in one scan would be exponentially large.

Their algorithm generates a set of all potential large itemsets in its initial pass, which becomes the superset for all large itemsets. They then count the support for each of the generated itemsets. The algorithm accomplishes this by dividing the database into non-overlapping partitions and generates itemsets on each partition. The database is partitioned in such a way that the large itemsets generated are able to fit into main memory. The generated large itemsets are then merged to create a set of potential large itemsets,

the goal being to find all global[3] large itemsets. In the next phase, the algorithm counts the support for these itemsets and identifies the large itemsets. Finally the algorithm outputs the large itemsets and their support. The correctness of the algorithm is based on the notion that any potential large itemset appears as a large itemset in at least one of the partitions [20]. This is because the algorithm counts the support of every global candidate set thereby discovering all large itemsets.

The 'Partition' algorithm uses the same method as the Apriori algorithm to prune off large itemsets whose subsets are not large and it too generates the next large itemset using a join operation on previous large itemsets. The difference is that the support of each large itemset is counted as the large itemsets are generated.

The Partition algorithm is susceptible to data skew, which means that the degree of data distribution in various partitions affects the support values. For example, if Christmas sales were all kept within a particular partition, there would be high support for that partition. However with regard to other partitions where no Christmas sales were done at any other time, the support for the Christmas partition would be discarded while generating the global candidate set. This shortcoming can be removed by randomly distributing data into the various partitions.

The Partition algorithm is aimed at decreasing both input/output and CPU overheads. The Apriori algorithm performs better than the Partition algorithm when processing requirements are low, but this changes when the processing requirements become high, such as when the minimum support is low. The Partition algorithm is considered to be an algorithm suitable for resource limited environments and can be easily parallelized. Though the algorithm performs relatively well, it does spend some time performing redundant computations on similar itemsets in different partitions, which does not occur with the Apriori algorithm [20, 32].

[3] A global large itemset is a combination of all local large itemsets from the various partitions.

Eclat

Zaki, et al [43] presented an approach to mining association rules that scans the database only once. They explored the idea that given the maximal frequent itemsets, an efficient algorithm could be designed to count the support of all the subsets of the frequent itemsets in a single database pass. They hoped to achieve this by using intermediate results to obtain approximations of the frequent itemsets, called the *potential maximal frequent itemsets*. Since it would not be possible to determine in advance the maximal itemsets. Their algorithm used itemset-clustering techniques to generate the set of potential frequent itemsets and then, applying efficient lattice traversal techniques, the algorithm was made to generate the frequent itemsets within each cluster[4]. Zaki, et al proposed two clustering schemes: *equivalence classes* and *maximal hypergraph cliques*[5]. They also used traversal techniques, which were based on *bottom-up* and *hybrid* search.

Their algorithm, Eclat (Equivalence Class Transformation), first uses an equivalence class technique to generate potential large itemsets. It then applies a bottom up traversal scheme to generate the frequent itemsets within each cluster. Each cluster is then completely processed before proceeding on to subsequent clusters, thereby allowing the database to be scanned only once. Scanning the database once saves input/output overhead and minimizes the amount of memory used. Related algorithms presented included *MaxEclat, Clique* and *MaxClique* [43].

MaxEclat uses a hybrid scheme to generate potential maximal or frequent itemsets. Clique uses a bottom-up scheme and MaxClique uses a hybrid lattice traversal schemes. Comparing experimental results, Clique was shown to provide a finer level of clustering and thereby reduced the number of candidates considered. The MaxEclat and MaxClique techniques were shown to out-perform bottom-up algorithms (such as Apriori) since their

[4]A cluster, in this context, is the largest possible large itemset.

[5]A clique is a graph in which each pair of vertices has an edge.

basic tasks are to find maximal itemsets and they need not scan the database as much as the Apriori algorithm. Overall, Eclat performed ten times faster than the Apriori algorithm and five times faster than the Partition algorithm. The Clique algorithm was more efficient than the Eclat algorithm because it cut down on its join operations by about 25% . MaxClique had a reduction of 75% in join operations thereby making it the best algorithm among the lot. Its efficiency was forty times that of the Apriori algorithm, twenty times that of the Partition algorithm and almost three times that of Eclat [43].

A major disadvantage with the Eclat algorithms is that a number of records have to be kept in memory at the same time and some of the record lists could be very long for a small number of itemsets. Apriori on the other hand easily takes care of this problem since records do not have to remain in memory [8].

DHP

Park, et al [49] proposed a hash based association rule mining algorithm called DHP (Direct Hashing and Pruning). They discovered that the generation of initial large sets in algorithms such as Apriori dominated most of the execution time since a number of the candidate sets generated by Apriori-like algorithms was very large. They came to the conclusion that the candidate set generation, especially for the large *2-candidate sets*, was the key issue to improving the association rule minings' technique execution time. Another issue brought up was the fact that the database had to be scanned every time a candidate set was being generated, and as the candidate sets got larger (from k-candidate sets to $k+1$-candidate sets), the number of generated large candidate sets decreased. They came up with the idea of decreasing the size of the database after each scan to remove records that did not contain large candidate sets to reduce scanning time. Their algorithm thus proposed two methods of improvements: to efficiently generate large itemsets and to reduce the size of the database.

DHP is based on the Apriori algorithm but uses a hash method for generating its itemsets and incrementally prunes the database to reduce its size. The algorithm proceeds as the Apriori algorithm does during its first pass over the database to generate its *1*-itemsets, it then creates a hash table for the *2*-itemsets. It performs the same join operation as the Apriori on its *1*-itemsets to generate its *2*-itemsets. It then determines the large *2*-itemsets by finding the support (checks to see if each item in the hash tree exists in the database, if it does, the item's count which is stored in a vector is increased, if not it is eventually discarded), prunes off the database and creates a hash table for candidate *3*-itemsets. This process is repeated until the final large itemsets are created. The difference between DHP and Apriori is that DHP creates and stores its itemsets in the hash table and only those itemsets that are able to pass through its *hash filtering* are used to generate the next large itemsets. Hash filtering is used to determine potential large sets in advance, to reduce the number of generated large sets. The filtering process reduces the size of the generated itemsets and effectively aids in the trimming of the database. Experimental results show that DHP has a higher execution time during initial itemset generation because of the creation of the hash table but with subsequent passes the execution time is drastically reduced when compared with the Apriori algorithm [49].

Parallel Association Rule Mining

Generating frequent itemsets from data is known to be computationally demanding [70]. It is also known that, given any number of itemsets the number of potentially generated frequent itemsets could be enormous and there is also the possibility of generating a very small number of large itemsets, thus wasting computational effort [70]. Zaki, et al [70] proposed using parallel computing as a way of increasing computational speed. Various parallel algorithms have been proposed to mine data, including the 'Count Distribution algorithm', the 'Data Distribution algorithm' and the 'Candidate Distribution algorithm'

[5].

The Count Distribution algorithm is a parallelized version of the Apriori algorithm that partitions its database and distributes the partitions among the processors evenly. Each processor then generates the support of the candidate sets from its local database. These are then summed to obtain the global support for each candidate set. Each processor then builds the large itemset in parallel and repeats the process until all frequent itemsets are generated. Though the algorithm minimizes communication overhead (only the initial local support values are exchanged) it creates a hash tree that is used to store the generated large candidate sets which is then replicated on every processor, thereby underutilizing the aggregate memory.

The Data Distribution algorithm also has the database partitioned and shared as in the Count Distribution algorithm but is designed to generate disjoint candidate sets on each processor, thereby utilizing the total system memory since replicated copies of large candidate sets are not kept on every processor. A flaw associated with this algorithm is that the entire database has to be scanned to generate the global support during each iteration. This causes I/O overhead to be high and, from experiments, it was shown to perform worse than the Count Distribution algorithm.

The Candidate Distribution algorithm performs the same form of database partitioning as was done by the previous algorithms but also allows each processor to work independently by selectively replicating the database in each partition. The selection is made as a trade-off between allowing processor dependence and balancing the workload. Each processor broadcasts the local frequent set to every other processor during every iteration to ensure that each processor has the same copy of frequent sets. The algorithm requires rescanning of the local databases during every iteration and has been shown to perform worse than the Count Distribution algorithm.

Another algorithm proposed by Zaki, et al is the CCPD (Common Candidate Partitioned Database) algorithm [69] which is similar to the Count Distribution algorithm and also has the database partitioned into the various processors' local drives. The difference between the CCPD algorithm and the Count Distribution algorithm is that the CCPD algorithm uses an optimization technique to balance its hash tree. The candidate sets are generated in parallel and stored in hash-trees shared by all processors. Each processor updates the counts of candidates in the shared hash tree after scanning its local database. The algorithm does not require a global count to be made, but instead performs a *barrier synchronization*[6] to ensure that all processors have updated their counts after every iteration. Previous algorithms required that the local partitioned databases be repeatedly scanned and a barrier synchronization be used after generating the required frequent sets, thereby incurring I/O overheads. Previous algorithms also required counts of candidate sets or database partitions to be communicated during iterations, which incurred communication overheads. A flaw with the CCPD algorithm is that it requires complicated hash structures, which incurs additional overheads [69].

FP-Growth

Generating candidate sets in long-patterned[7] data is costly and tedious for algorithms that repeatedly scan databases such as Apriori. It is also costly for algorithms such as Apriori to handle large numbers of candidate sets. For example, if there were about ten thousand *1*-itemsets, the Apriori algorithm would need to generate over fifty thousand *2*-itemsets. This is because a join operation of all *1*-itemsets would generate:

$$\frac{10,000!}{(10,000-2)!*2!} where(! = factorial)$$

[6] A barrier is a method of ensuring that all processors have reached a particular point during processing. Early processes must wait for all other processors to arrive at the barrier.

[7] Long-patterned data are data that contain very long chains of patterns or prolific numbers of patterns.

initial 2-itemsets, after which the support count and pruning phase of the 2-itemsets would have to be done. Han, et al [63] were motivated by the idea of avoiding such costs to consider constructing a data structure to reduce computational time especially in databases that contain both long and short patterns. The FP-growth (Frequent Pattern-growth) algorithm uses a data structure called the *frequent pattern tree* (FP-tree), which is an extended prefix-tree structure that stores information about frequent patterns. The nodes of the tree are arranged in such a way that frequently occurring tree nodes have better chances of sharing their frequent patterns than other non-frequently occurring tree nodes. From experiments performed, it was shown that the tree structure was highly compact, thereby providing a smaller data set to mine. Han, et al also developed an FP-tree based pattern growth[8] mining technique, which starts from a frequent length-1 pattern (or "itemset" using earlier terminology). That technique then examines its sub-database consisting of frequent items occurring with the initial pattern, and constructs its FP-tree recursively. The pattern growth is accomplished by joining the initial pattern with the generated frequent patterns from the FP-tree. Finally, Han, et al applied a partition-based divide and conquer search technique, which partitions the mining tasks into smaller tasks and thereby reduces the size of the search space. The tasks are then translated from searching for long patterns to short ones followed by performing a join operation on them.

The search cost is substantially reduced using FP-Growth when compared to the Apriori-like algorithms which apply bottom-up generation of frequent itemsets. FP-growth was shown to be faster than Apriori and the speed advantage increases with the length of the frequent patterns. FP-growth has also been shown to scale effectively with data size and decrease in support [63] thus making it a scalable algorithm. The Apriori algorithm, on the other hand, has memory requirements that are exponential in the number

[8]Pattern growth is a system of generating and storing patterns from a database one step at a time.

of candidate sets or inversely in the size of the support [63]. The FP-growth algorithm generates only new itemsets found in the database, unlike the Apriori-like algorithms which generate new itemsets and then test the itemsets' subsets to determine if they are contained in the initial large itemset. Though the FP-tree is compact it does require that the database be initially scanned twice during the FP-tree's construction phase. Also, the FP-tree created may still exceed the main memory size for large databases, which does not help the algorithm's claim of compactness [32, 63].

SPRINT

Classification is a different type of data mining technique that has been shown to be effective [45]. Classification techniques (of which SPRINT is an example) aim at discovering a small number of pre-determined rules, while association rule mining techniques (which Apriori is an example of) aim at discovering all available rules with no pre-determined targets [62]. As such, Apriori and other association rule mining techniques can discover rules that SPRINT and other classification techniques will overlook.

Classification algorithms require that some, if not all, of the datasets reside in memory. This limits such algorithms' ability to mine data from very large databases. The advantage of having larger datasets is that the accuracy of the classification model would be improved. Shafer, et al [45] addressed this problem by developing the SPRINT (Scalable PaRallelizable INduction of decision Trees) algorithm. Their algorithm used decision trees because they can be easily constructed and understood. Decision trees have the added advantage of being easily converted to SQL statements for efficient database access. A decision tree classifier was built in two phases. Shafer, et al began by recursively dividing the training set until each portion consisted of itemsets from a particular class (or set). The non-leaf nodes of the decision tree were test points for one or more attributes, and they determined which training sets belonged to a particular leaf. The next

phase involved pruning. The tree was pruned to remove inconsistencies that were due to statistical noise or other forms of errors particular to the given training sets.

Shafer, et al focused on building a decision-tree that was both scalable and easily parallelizable. SPRINT was able to handle data sets too large for other algorithms such as SLIQ (Supervised Learning In Quest) [44], although SPRINT's scalability is considered expensive due to its use of hash-join and sorting. SPRINT was also designed to avoid any form of centralized structural dependence that was memory-resident, so that the algorithm could be efficiently and easily parallelized [61].

RainForest

Rainforest [61] is a framework for decision tree classification techniques that are aimed at improving the quality of decision tree algorithms. Rainforest tries to bridge the gap between main memory limitations and data scalability in data-mining. Gehrke, et al [61] presented four algorithms under the RainForest framework that scale with the size of the database and can adapt to any given main memory size. This was done by focusing on the tree growth phase, since it accounts for most of the computational overhead of previous algorithms[9]. RainForest tries to present a unifying framework that can be applied to most decision tree algorithms to improve their scalability. The authors accomplished this by isolating the AVC-set (The AVC-set is a set of categorical attributes with a very small domain that have been aggregated), which enables scalability of the classification tree algorithms by picking an appropriate splitting criterion for the itemsets to enable them fit into memory. This was also done because they observed that previous algorithms based their splitting criteria at a tree node on the AVC-group (an AVC-group is a group of

[9]Gehrke, et al showed that SPRINT almost triples the size of its database during its tree growth phase and incurs a high computational cost while trying to keep its attribute list sorted at each node.

AVC-sets) of that particular node thereby incurring significant computational overheads. Four algorithms were presented: RF-Write, RF-Read, RF-Hybrid and RF-Vertical [61].

RF-Write assumes that the AVC-group will fit into memory and creates it after the database is scanned. The child nodes of the AVC-group are then created and partitioned after the classification algorithm is applied. An additional scan of the database is made from which each tuple[10] is written into one of the partitions. The algorithm repeats itself until each partition is written into.

The RF-Read algorithm reads the database directly (instead of writing partitions as the RF-Write algorithm) when it is estimated that the entire AVC-group will not fit into memory at some later point. Thus the database is read several times and AVC-groups are constructed at each read for a subset of new child nodes in the tree. This increases the I/O overhead.

The RF-Hybrid algorithm is a combination of the RF-Write and RF-Read algorithms. It starts with the RF-Write algorithm until all the AVC-groups can no longer be fit into the main memory and then it switches to RF-Read.

The RF-Vertical algorithm is designed for AVC-groups that are unable to fit into memory, but where each of their individual AVC-sets are able to fit into memory. The RF-Vertical algorithm proceeds like RF-Hybrid, however instead of dealing with the AVC-group as a whole, it is broken into its various AVC-sets which are computed separately.

From experiments, Gehrke, et al's algorithms were shown to have higher speed-ups when compared to the SPRINT algorithm and this result improved further whenever the ACV-groups were able to fit into main memory.

Rainforest is based on classification and, as such, also misses out on determining all

[10]The members of a node n is the set of tuples in the database that follows the path from the root to n when being classified by the tree [61].

the rules that would be discovered using, for example, the Apriori algorithm.

CBA

While association rule mining is aimed at searching out all rules within a given database that satisfy the given minimum support and confidence, classification rules try to discover a small set of rules in the given database that form a correct classifier. A classifier is a model of the classifying attribute (CA), which is based on other attributes and determines which class each data entry belongs to. Since association rule mining and classification techniques have both been shown to be good tools in mining data, Liu, et al [62] proposed the idea of integrating the two techniques to gain the advantages of each in a single package. CBA (Classification Based on Associations) combines both association rule mining and classification rules to produce a complete set of classification rules. The combination of the association rule mining and the classification techniques occurs using a subset of the association rules, which have characteristics of being limited to the classification class attributes. These subsets are termed *Class Association Rules* (CARs). The CBA algorithm proceeds as follows. First, all continuous (numeric) attributes are separated into distinct groups (i.e. continuous data are partitioned into intervals e.g. [1, 6], [30, 67]). Then the Apriori algorithm is used to mine all the CARs that satisfy the desired support and confidence levels. This is done because classification rules tend to contain large numbers of association rules, and therefore mining the CARs alone tends to reduce the number of rules generated. Also classification datasets contain numeric attributes, which are not easily dealt with in mining association rules (association rules are typically used to mine categorical or binary data). Finally the CBA algorithm builds a classifier based on the CARs generated.

One of the advantages associated with generating the classifier is that it also generates the set of rules required to form the classifier. These rules however, were not very

interesting or useful to the user, while other interesting and useful rules lay unexplored. With the CBA algorithm, Liu et al were able to find interesting rules since the association rule-mining algorithm initially generates the required rules specified by the user's support value. In building the classifier, the earlier applied method was called *M1* and was based on the C4.5 method [51]. This method was inefficient since the database was not resident in memory and it was necessary to make multiple passes over the database. An improvement was made, called *M2* which did not have to make as many passes over the database [62]. However *M2* still has the flaw of searching for a small number of pre-determined rules, unlike Apriori which discovers all available rules.

Mining using Relational Algebra Operations

Databases have simple operations that are used for data processing. If it is possible to use these same operators to find meaningful relationship amongst data items, it would save researchers the trouble of using other machine learning techniques. Holsheimer, et al [27] introduced the idea of using only database operations for data mining. They proposed a way of efficiently determining association rules from binary (or categorical) databases using only a general-purpose database management system and the basic relational algebra operators that are supported by such database systems. Their goal was to compare the use of only simple database operators for finding association rules with specialized techniques such as the Apriori algorithm. Their algorithm uses only the relational algebra *intersection* and *union* operators. A retail market was used as their test domain and items were grouped in categories (for example, Heineken and Stout might have been grouped into a category called *beer*). This grouping was done because rules generated from such groupings produced supports larger than the given minimum support. For example, a combination of Stout and apples might not constitute a large set but beer (which comprised of Stout and Heineken) and fruit (which might have been

comprised of Apples, Oranges and Grapes) probably would. To count the support for *beer* they performed a *union* operation on all rows containing the items in the group (e.g. performing a union of all rows containing Stout and Heineken).

Holsheimer, et al [27] first converted the database into a *decomposed storage structure*, which is a structure that has each record containing a unique record identifier (which they called TID) stored as a set of items or columns. To generate the *1*-itemset, the authors selected columns whose number of items were above the minimum support. To generate the *2*-itemset of items i and j for example, the authors took the intersection of i and j. The result was a new column which contained the TIDs of both i and j which was then stored in the database. To compute the support of a triple of items $(i\,j\,k)$, given that $(i\,j)$, $(i\,k)$ and $(j\,k)$ are large, they simply took the intersection of $(i\,j)$ and (k), since the result of $(i\,j)$ was already available from previous operations. This method reduced the number of database scans and, furthermore, since $(i\,j)$ had been stored in the database there would be no need to access columns i and j anymore, so these items could be removed from the database to decrease storage requirements.

The algorithm, however, had a problem during its second pass. Since many candidate sets were generated and only a few were large, much processing time was unnecessarily consumed. For example, it was possible to generate about 200,000 candidate sets with only about 40 being large sets. The authors tried to reduce the number of database operations by generating information on groups of candidate sets rather than on single candidates. For example, if they assumed that A, B, C and D were large *1*-itemsets, and given that (A B) was also a large set, if they took the union of A and B, it would contain all the TIDs in both sets. If the intersection of (A $\cup$ B) $\cap$ (D $\cup$ C) was small, they inferred that none of (A C), (A D), (B C) and (B D) were large and thus decreased the computational overhead to about three fourths that of the original processing time since the pairs (A D), (B C) and (B D) were not processed. One limitation of their algorithm is

that it can not be used with very large datasets and very low minimum support. Another flaw is that the algorithm is suited only for categorical databases.

Caching Mining Rules

A major problem associated with mining association rules is the large processing time required when answering even simple queries. This causes association rules to normally be mined in batch mode rather than interactively, thus limiting the techniques effectiveness and usability. Data mining research has showed that for maximum system utilization (having multiple users simultaneously), the system has to be interactive and thus should ideally have processing time proportional to the size of the result and not the size of the database [46]. Nag, et al [46] proposed accomplishing this by creating a cache-based interactive mining system with quick query response that could support several users simultaneously. As an example, the authors computed that if two queries **A** and **B** were requested having a support of 3 and 4 respectively, it would take about 330 seconds to accomplish the first query from a database of about a million records and about 300 seconds for the next query. But applying the results gained from the first query, the second query could easily be run in about 0.04 seconds. They based their caching strategy on the notion of *guaranteed support*, which implied that if they could guarantee that their cache contained all the required frequent itemsets, there would be no need to access the database for further queries. By caching and re-using the results of previously submitted queries from other users their system can rapidly answer subsequent queries. This concept was based on similar ideas from on-line analytic processing (OLAP) systems [18]. To assess the effectiveness of caching frequent itemsets Nag, et al compared three caching algorithms: *No Replacement* (NR), *Simple Replacement* (SR) and *Benefit Replacement* (BR).

The NR algorithm is the simplest of the three algorithms. The cache was divided

into sectors with the *k-itemset* stored in the k^{th} sector. During the k^{th} pass of the Apriori algorithm only the k^{th} sector of the cache is accessed. The "Apriori-generator" initially generates the candidate sets and the cache is accessed to find the support of the itemsets. During the search, if there are new candidate sets whose support is not found in the cache, a hash-tree of these itemsets is created and the support is determined from the database and added to the cache. Itemsets with a support less than the minimum support are discarded and the remaining itemsets are used to generate the association rules. For each query, all itemsets counted are added to the cache until it becomes full and no more itemsets can be added. This method saves the support count computation and once an itemset is placed in the cache it is never removed. Hence, the name - "No Replacement".

The SR algorithm takes into account the fact that not all itemsets in the cache are of equal importance. Those with larger support are more valuable than the others since they are used in more queries. For example, itemsets with support of about 60% can be used to answer queries that require support of 60% or less. Since these valuable itemsets will be used over and over it is therefore more profitable to store them in the cache and sort all the cache entries in ascending order of support. The SR algorithm operates like the NR algorithm but allows itemsets with the lowest support that are stored in the cache to be replaced by other more valuable frequent itemsets when the cache becomes full.

The BR algorithm uses a tree based data structure to store the cache contents within the cache. The advantage that this provides is that the cache contents are indexed by support level, thus avoiding the need to rescan the entire cache to count the support.

Experiments were performed using the various algorithms (and a 'no cache' system based on the simple Apriori algorithm) and it was found that all the caching algorithms performed better than the 'no cache' algorithm and that the BR algorithm performed five times better than Apriori. The NR algorithm was also found to be the least efficient among the caching algorithms [46]. Essentially, the work by Nag, et al simply imple-

ments the Apriori algorithm with a *cache* attached to it. Caching, of course, may also be useful in other algorithms.

I have briefly discussed some of the algorithms available for data mining. I will now review my selected representation technique, fuzzy logic. I will also overview how fuzzy logic has been used in a number of data mining approaches to deal with data representation issues.

2.2.2 Fuzzy Logic

In this section, I briefly discuss fuzzy logic research that is related to my thesis before moving on to consider the use of fuzzy logic in association rule mining.

Fuzzy Sets and CBR

Hansen and Riordan [14] analysed numerical data from a meteorological centre by combining fuzzy logic and Case-Based Reasoning (CBR) [14]. Meteorological data, such as that produced by the Niagara Area Weather Network (NAWN), are continuous, large and growing. CBR, which uses past information to make decisions on present events, is a useful technique is searching historical records for similar patterns. Hansen and Riordan designed a case-based system that would find an optimal way of indexing cases[11] for effective retrieval by representing the cases using fuzzy sets.

The authors integrated the advantages of both techniques (fuzzy logic and CBR) thus developing a hybrid fuzzy logic-CBR system. Fuzzy logic-CBR was aimed at incorporating grammatical expressions into the system. The algorithm was called the *fuzzy k-nn algorithm* and measured similarities between past and present data. The algorithm

[11] In this situation, a case is a representation of the available data in the domain.

was constructed by configuring a similarity-measure function, which traverses the case base to find similar patterns and make necessary predictions. The technique was able to identify one hundred applicable cases from a library containing over 300,000 cases. The system was used by meteorologists to predict visibility at airports [14, 26].

Other methods of generating rules using fuzzy logic are also possible and include neural networks [34] and uncertain reasoning [6].

2.2.3 Fuzzy-Association Rule Mining

This section briefly discusses previous research that used both association rule mining and fuzzy logic.

Mining Association Rules Using Intervals

Srikant and Agrawal [54] considered the problem of mining large relational databases with both quantitative and categorical attributes. They mapped categorical attributes to a set of consecutive integers. Quantitative attributes whose domains were too large were partitioned into intervals and then mapped to consecutive integers. Other quantitative attributes that were not partitioned might be mapped directly to consecutive integers. These mappings were performed in such a way that the orders of values or intervals were preserved regardless of whether the data was partitioned or not. For example, the intervals [1, 10], [11, 20] and [21, 30] were mapped to 1, 2 and 3 respectively. This was to allow the records to be treated as sets of (attribute, integer) pairs.

There were problems with this approach. First, when the number of intervals for the quantitative attributes was large, the support for a given interval could be rendered low, and without using large intervals, some rules might not be found at all due to a lack of

sufficient support. Second, some information was lost when attributes were partitioned. For example, given a rule such as *number of houses owned = 1 ⇒ people married* with a support of 100%, when the data is partitioned such that *number of houses owned= [1, 2] ⇒ people married* might give a support of only 70%. If the minimum support was 72% then the rule may not be considered. This problem tends to increase with the interval size. This means that if the intervals are too large, information will be lost and some rules may not have the required minimum support. Also, if the intervals are too small some rules may not be able to make up the support at all.

Srikant and Agrawal tried increasing the number of intervals by considering all possible ranges without affecting the minimum support. This allowed intervals to overlap, such as (20, 25) and (22, 30), but it created new problems. Higher execution times resulted, and many unnecessary rules were discovered, such as *if age is [20, 25] ⇒ married* and *if age is [20, 30] ⇒ married.* To reduce the execution time, they restricted the number of adjacent intervals by introducing a user-specified maximum-support. Thus, intervals whose combined support exceeded this value were not created. To avoid having many unnecessary rules, they introduced an *interest measure* that helped prune unwanted rules. Their algorithm operates as follows:

- They first determine the number of partitions necessary, and then map the various attributes to the partitions depending on the types and range of intervals.
- They then find the support for both attribute types (categorical and quantitative) and determine the frequent itemsets. To reduce execution time, adjacent attributes are combined as long as their support is less than the user specified maximum support.
- They then use the Apriori algorithm to generate the frequent itemsets.

- Finally they determine the rules of interest.

The problem with their algorithm is its use of sharp boundaries in partitioning the database, which results in itemsets at the boundaries being either ignored or over-emphasized. To illustrate this problem, consider taking a range of values from 1 to 100. If the database is partitioned into ten parts, it is possible that the value 11, which is in the second partition, could play an insignificant role in the second partition but play a significant role in the first partition [54].

Determining Fuzzy Association Rules Using Correlation

Kuok, et al [39] focused on the shortcomings of Srikant and Agrawal [54]. They used fuzzy logic to try and overcome the problems described in the previous section. The fuzzy sets used for the attributes in the database were manually specified. The membership functions of each record for an itemset were multiplied, and then divided by the total number of records to determine if the itemset was a frequent itemset or not. For example, if we had an itemset called *low* and the following membership functions from various records as: *0.3, 0.1, 0.8, 0.65* and *0.7*, with a total record size of 10, the result would be

$$\frac{0.3 * 0.1 * 0.8 * 0.65 * 0.7}{10} = \frac{0.01092}{10} = 0.001092$$

Depending on the cut off value given, the *low* itemset would be added if it was above the cut off value or dropped if it was below. The fuzzy *mul* operator was used instead of the other fuzzy operators because it always gives values that are less than or equal to 1. This was acceptable because the authors believed that all records should have a general maximum contribution of 1 or less, since the highest membership function an item can

have is 1 and the other fuzzy operators tend to give a contribution of more than 1 in some cases. To generate a rule from the frequent itemsets, they applied a correlation technique that was different from statistical methods in that their calculations took a user specified membership threshold into account. The authors were thus able to produce association rules in the form of 'fuzzy association rules'.

Determining Fuzzy Association Rules Using Clustering Techniques

Fu, et al [3] proposed an algorithm to generate fuzzy-association rules on quantitative data using clustering techniques. They also focused on the shortcomings of Srikant and Agrawal [54] and felt that it was difficult to know in advance the fuzzy sets that would be applicable for a given application. They also believed that experts for such domain applications were not a reliable source for obtaining the required fuzzy sets for quantitative datasets. As a result, they chose to explore creating the required fuzzy sets automatically from the available data using clustering techniques. They first transformed the database into a format in which clustering techniques could be applied. A clustering technique was then used to determine the medoids[12] of the k-clusters. As an example, if we have k-medoids of 5, that means k is 5. That is there exist 5 fuzzy sets for each quantitative attribute (which could be represented as *very low, low, medium, high and very high*). The fuzzy sets were then created according to the medoids of the k clusters. They then derived the corresponding membership functions for the fuzzy sets. The algorithm was shown to be relatively efficient at mining large databases.

[12]A medoid is a representative object in each cluster that is the most centrally located point in the cluster.

Determining Fuzzy Association Rules Using Normalization

Gyenesei [24] proposed that if quantitative attributes could be mapped to "boolean rules", then algorithms such as Apriori could then be used to find quantitative association rules. Boolean rules are another way of representing association rules derived from categorical attributes. Gyenesei built on the work of Srikant and Agrawal [54] but instead of using intervals to partition the quantitative attributes he replaced the intervals with fuzzy sets. Gyenesei used two methods to mine the fuzzy quantitative rules: one with normalization and the other without normalization. Suppose a quantitative attribute had the value 24, with a membership function of 0.8 in a given fuzzy set and 0.4 in another fuzzy set. The total support contributed by the value 24 from the two fuzzy sets would be more than 1 thus making the value 24 more important than other values. Gyenesei applied *normalization* to bring the total contributed support of the value to a maximum of 1. The *without normalization* method makes no changes to the support, even though it means the itemset is contributing more than the others. His basic algorithm operates as follows:

- The database is transformed into a *fuzzy* database by user specified fuzzy sets and the candidate *1*-itemsets are generated.
- The new database is scanned and the fuzzy support is counted (the fuzzy support is obtained by adding the number of itemsets within each record and dividing the result by the total number of records).
- The rest of the algorithm follows that of Srikant and Agrawal [54].

Gyenesei's sample database consisted of both categorical and quantitative attributes. From the experiments performed it was shown that the fuzzy method *with normalization* gave more interesting frequent itemsets than *without normalization*. When compared with Srikant and Agrawal [54], Gyenesei's technique was shown to give similar itemsets.

Determining Mining Fuzzy Association Rules by Adding Weights

In [23], Gyenesei extended his earlier work by adding *weights* to each attribute-fuzzy pair. His work was similar to that of Cai, et al [10], but he calculated his weights using the products of membership functions and not the sum. He also incorporated confidence and support thresholds, which was similar to his earlier work. Thus he redefined his frequent itemset definition as "an itemset is frequent if the weighted fuzzy support is greater than or equal to the user specified support". The frequent itemsets also had the property that all subsets of such itemsets were frequent, as such Gyenesei applied a bottom-up algorithm suited for fuzzy sets to mine the fuzzy-association rules. He also applied the techniques of normalization and non-normalization, as he did in his previous work.

Through experimentation, Gyenesei realised that subsets of his frequent itemsets might not be large and therefore could not generate candidate sets from their seeds. He applied a 'z-potential frequent subset' technique to combat this problem. A k-itemset is said to be a z-potential frequent subset if the product of the weights of k-itemset and the remaining itemsets with maximum weights is greater than the user specified support. Apart from these differences, the algorithm operates just as that his earlier one does.

Mining Fuzzy Association Rules Using Trees

De Graaf, et al [15] grouped the attributes in their algorithm into sets (such as *Heineken, Fruit*) to form the itemsets. To determine the support of such compound items, they took the product of the membership values of the different attributes of that item and added them. Their goal was to mimic the fuzzy *and* operator. For example, to calculate the support of *(Heineken, Fruit)* given membership functions for the different attributes such as *(0.1, 0.9)* and *(0.2, 0.4)*, the product of each pair of membership functions would be

calculated as:

$0.1 * 0.2 = 0.02$

$0.1 * 0.4 = 0.04$

$0.9 * 0.2 = 0.18$

$0.9 * 0.4 = 0.36$

giving *0.02, 0.04, 0.18* and *0.36* which are then added to give 0.6. The result was that a large number of rules were generated, so they tried to reduce this to only the *interesting* rules by creating a tree-like taxonomy, where the leaves were the various itemsets. Every parent in the taxonomy is a sum of its children which depicts the result of applying the fuzzy *or* operator to the children. An itemset is then considered interesting if its first generation ancestors have a support that is higher or lower than the real support by a significant amount called the 'interestingness threshold'. Their algorithm was based on Apriori with the necessary changes to the support to include the 'interestingness threshold'.

Fuzzy Transaction Data-mining Algorithm

Hong, et al [28, 29] used a modified version of the Apriori algorithm and fuzzy logic to generate association rules from a quantitative database. Their algorithm was called Fuzzy Transaction Data-mining Algorithm (FTDA). They manually generated their fuzzy sets and calculated the membership functions of the data. For example, taking *low*, *medium* and *high* as fuzzy sets and applying the sets to an attribute such as *humidity* with value 86, they would obtain three membership functions for that value as *(Low = 0.0), (Medium = 0.0), (High = 0.7)*. They applied this to all the data in the database and calculated the sum of the attributes which they called the cardinality. For example, they might have obtained Table 2.8 after transforming their database to fuzzy membership functions.

No.	Humidity			Temperature			Rainfall		
	Low	Med	High	Low	Med	High	Low	Med	High
1	0.0	0.0	0.5	0.0	0.0	0.8	0.1	0.5	0.0
2	0.8	0.0	0.0	0.0	0.4	0.2	0.9	0.0	0.0
3	0.4	0.2	0.0	0.0	0.5	0.1	0.5	0.1	0.0
4	0.0	0.5	0.4	0.0	0.0	0.9	0.0	0.2	0.4
5	0.0	0.8	0.0	0.7	0.0	0.0	0.4	0.2	0.0
Cardinality:	1.2	1.5	0.9	0.7	0.9	2.0	1.9	1.0	0.4

Table 2.8: Table showing the membership functions of 5 records and their cardinalities using FTDA

From the table, the cardinalities for *Humidity* would be 1.2, 1.5 and 0.9. The highest fuzzy cardinality would be picked (in this case it would be 1.5). Therefore the corresponding fuzzy set *medium* would be used as the fuzzy set for *Humidity*. From Table 2.8, the fuzzy set *High* would be used for the attribute *Temperature* and the fuzzy set *Low* would be used for the attribute *Rainfall*, since the cardinalities of these fuzzy sets were higher than the others. This process is done for every attribute in the database to cut down on processing time by almost 70%. The attributes and fuzzy sets selected by FTDA were then put into separate tables. Cardinality counts above the specified support were used to generate the required *1*-itemsets, as shown in Table 2.9.

itemset	cardinality
Humidity	1.5
Temperature	2.0
Rainfall	1.9

Table 2.9: Highest cardinalities obtained for each attribute using FTDA

Those attributes whose counts are below the support are pruned. A modified version of the Apriori algorithm was then applied to the remaining attributes. The algorithm proceeded as follows: the 2-itemsets were generated by taking the intersection of all the pairs of attributes. For example taking the fuzzy intersections of (*Humidity.medium, Temperature.high*), (*Humidity.medium, Rainfall.low*) and (*Temperature.high, Rainfall.low*). The intersection applied to *Humidity* and *Temperature* is shown in Table 2.10.

No.	Humidity	Temperature	Humidity ∩ Temperature
1	0.0	0.8	0.0
2	0.0	0.2	0.0
3	0.2	0.1	0.1
4	0.5	0.9	0.5
5	0.8	0.0	0.0

Table 2.10: Generating the 2-itemsets by taking the intersections of 1-itemsets cardinalites using FTDA

The cardinalities are then calculated for all the 2-attribute sets, and the attribute sets with cardinalities less than the support are pruned. This process is continued until the final k-attribute sets are obtained. For example if we have *Humidity.medium, Rainfall.low* as our *k-attribute sets*, the association rule will be either

if Humidity = *Medium*, then Rainfall = *Low*

or

if Rainfall = *Low*, then Humidity = *Medium*.

To determine the correct rule, the intersection of *Humidity* and *Rainfall* is taken. Whichever intersection gives a value greater than the *confidence* (and not support) is considered to be the correct rule.

Hong, et al also proposed a modified version of the FTDA algorithm [30] using "all important"[13] fuzzy linguistic terms instead of picking the fuzzy set with the highest cardinality value alone. As such, only fuzzy cardinalities above the support were used. Though this increased their processing time, they believed the trade-off was an increased number of correct rules.

2.2.4 Summary of Related Work

From the research discussed, it is clear that the Apriori algorithm has been used extensively with success and that a number of other algorithms have been built upon Apriori. Fuzzy logic has been used successfully in much research as well and it has also been shown to integrate well with other systems to handle the clear-cut boundary problem effectively. By combining fuzzy logic and the Apriori algorithm, I hope to combine the advantages of both techniques into a Fuzzy-Apriori association rule mining system.

The next section discusses the implementation of my Fuzzy-Apriori systems and the modifications that were made to the Apriori algorithm.

[13] Fuzzy sets with cardinality values higher than the support.

Chapter 3

Methodology

My research goal is to effectively mine the numeric data obtained from NAWN and present the results in a format that can be understood by users. In this section, I discuss in detail the methodological steps taken to build the Fuzzy Apriori systems that were designed to achieve my research goal.

My system operates in five steps: I initially define the various fuzzy sets applicable to a given data set, such as *high, medium, low, very low,* etc. I then apply these fuzzy sets to the available crisp data during the *fuzzification* process. The modified version of the original Apriori algorithm, introduced by R. Agrawal and R. Srikant in [52] is then applied to the fuzzified data. The resulting large set is then converted to fuzzy-association rules of the form:

$$\mathbf{A : Y \Rightarrow B : Z}$$

where **Y** and **Z** are the generated fuzzy sets, and **A** and **B** are the data attributes. The resulting rules are then converted back into crisp values during the process of *defuzzification* after evaluating the correctness each rule and presented to the users of the data mining system.

3.1 The FA1 Algorithm

The following sections give an explanation of the methodology in the first system (FA1) developed. The FA1 system was created according to the system architecture shown in Figure 3.1.

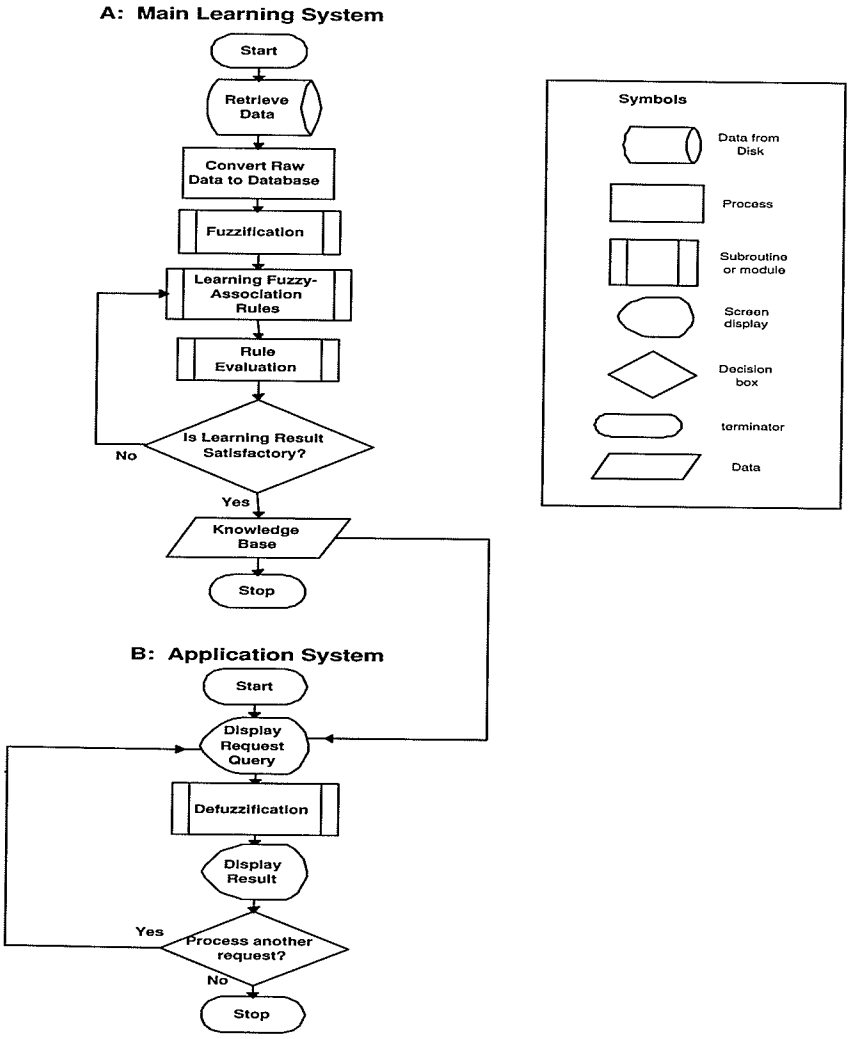

Figure 3.1: System Architecture

As shown in Figure 3.1, the data is first retrieved and then converted into a database. This was necessary since the test data I used for the system was not originally stored in a database. The data from the database is then converted into fuzzy data during the process of fuzzification (shown in the Fuzzification module in Figure 3.1). The system continues by applying the modified Apriori algorithm to the fuzzified data to generate the required fuzzy association rules. This process is done within the Learning Fuzzy-Association Rules module. Finally, the Rule Evaluation module checks each generated fuzzy-association rule to ensure correctness. After the learning process is complete the new rules are stored in a file called the knowledge base, from which the application system can query based on users' requests.

3.1.1 Fuzzification

In converting data into fuzzy sets, I first determined the types of fuzzy sets to be used. For example, the attribute *Temperature* in the database was divided into three simple sub-groups: *low* Temperature, *medium* Temperature and *high* Temperature. Each attribute was converted into fuzzy sets (i.e. from numeric to linguistic terms) using either the *triangular* or the *trapezoidal* fuzzy curves. These curves were chosen for two reasons. First, the curves are simple to implement, since they use linear functions (with three or four parameters depending on the system's symmetry) unlike more complex systems such as time-series data, which use *bell*-shaped curves. Second, the fuzzy operations themselves are elementary and as such require only simple fuzzification techniques [19]. The trapezoidal curves were placed at the edges of triangular fuzzy sets for regions that did not change but had continuous values. For the traingluar or trapeziodal membership curves using three fuzzy sets, example of the boundary values (as discussed in Section 2.1.2) inlcude:

Humidity:

Low : a=0, b= 0, c=30, d=50 with a trapezoidal shoulder fuzzy curve;

Meduim : a = 35, b =50, c = 75 with a triangular fuzzy curve;

High: a = 40, b =80, c = 100, d=100 with a trapezoidal shoulder fuzzy curve;

Radiation:

Low : a=0, b= 0, c=5, d=10 with a trapezoidal shoulder fuzzy curve;

Meduim : a = 9, b =15, c = 21 with a triangular fuzzy curve;

High: a = 10, b =25, c = 35, d=35 with a trapezoidal shoulder fuzzy curve;

Those values greater than the d boundary would be given a membership function of 1. However these values for the fuzzy curves would be provided for by an expert. The triangular and trapezoidal curves were also selected by me, however other fuzzy curves could also be used in future research. Taking the data in Table 3.1 as an example, the conversion of the shown database entries[1] into fuzzy membership functions using three fuzzy sets is shown in Table 3.2.

No.	Humidity	Temperature	Rainfall
1	103.00	10.80	0.20
2	49.00	8.59	0.00
3	51.00	7.82	0.15
4	85.20	16.57	0.20
5	64.80	2.13	0.10
6	53.30	6.18	0.10

Table 3.1: Sample Database

[1]The units associated with the NAWN data were not provided and therefore have not been specified in this thesis

No.	Humidity			Temperature			Rainfall		
	Low	Med	High	Low	Med	High	Low	Med	High
1	0.0	0.0	0.5	0.0	0.0	0.8	0.1	0.5	0.0
2	0.8	0.0	0.0	0.0	0.4	0.2	0.9	0.0	0.0
3	0.4	0.2	0.0	0.0	0.5	0.1	0.5	0.1	0.0
4	0.0	0.5	0.4	0.0	0.0	0.9	0.0	0.2	0.4
5	0.0	0.8	0.3	0.7	0.0	0.0	0.4	0.2	0.0
6	0.2	0.6	0.0	0.4	0.5	0.3	0.7	0.3	0.0

Table 3.2: Sample Membership Functions

Fuzzy sets with membership functions equal to zero were removed in each record, since such membership functions showed the data was not a member of the set. A new database was then created using the fuzzy sets as itemsets for each record. This was done to create a database that could enable the application of categorical association rule mining algorithms. An example of converting Table 3.1 into fuzzy linguistic values is shown in Table 3.3.

No.	Humidity	Temperature	Rainfall
1	High	High	Low, Med
2	Low	Med, High	Low
3	Low, Med	Med, High	Low, Med
4	Med, High	High	med, High
5	Med	Low	Low, Med
6	Low, Med	Low, Med, High	Low, Med

Table 3.3: Sample Fuzzy Database

Most research done using fuzzy logic uses various computational methods such as normalization [24], normalization and weights [15], etc. to aggregate membership functions for each database attribute to get a single fuzzy set under each attribute record. I decided not to aggregate the membership functions, to allow each available fuzzy set to play a role in generating the required fuzzy association rules, in the hope that this technique would create additional reliable rules. The intention was that if each datum in a record was expressed in every fuzzy set available to it, the algorithm could discover new hidden information.

3.1.2 Learning Fuzzy Association Rules

After the database was converted into a form that could easily be categorically mined, the system then used the modified-Apriori algorithm shown in Algorithm 1.

The algorithm operates as follows. The initial *1*-itemsets are generated by taking all items in each database record. The support for each itemset is counted and those with support less than the minimum are pruned. The remaining itemsets make up the large *1*-itemsets and are used as a seed to generate the *2*-itemsets using the routine *fuzzy-apriori-gen*[2]. The *2*-itemsets are obtained by performing a join operation on all large *1*-itemsets. All subsets of the newly generated *2*-itemsets are checked to ensure they are also large itemsets. Any itemset whose subset is small[3] is removed. The support for the remaining *2*-itemsets are then counted. Any *2*-itemset less than the required minimum support is pruned. The database is also pruned of records that do not contain large itemsets. The pruning of the database was done to minimise the database scanning time. The entire process continues until the final large k-itemsets are obtained.

[2]The *fuzzy-apriori algorithm* generates new large itemsets using previous large itemsets as seeds.

[3]Itemsets with support values lower than the given support are said to be small.

Algorithm 1 Fuzzy-Apriori Algorithm (FA1)

Input: NAWN weather database

Output: Fuzzy-association rules

D := $FuzzifiedData$

L_1 :(large 1-itemsets);

for ($k := 2$, L_{k-1} != 0, $k := k + 1$)

do

$C_k := fuzzy\_apriori\_gen(L_{k-1})$;

do forall items $t \in D$

$C_t :=$ subset(C_k, t);

do forall candidates $c \in C_t$

$c.count{+}{+}$; count for support

end

$L_k :=$ (c $\in C_k$ || $c.count \geq$ minsup)

end

end

Output Result

The itemsets are stored as strings in temporary files after they are generated. Each time a k-itemset is generated, it is directly written into a text file to decrease pressure on the main memory. When the itemsets are required to generate the next set of large itmesets, they are accessed from file and stored in memory in a format similar to a linked list structure. The next set of k+*1*-itemsets are generated by accessing the k-itemsets from the file, storing them in a java vector and performing the join operations on them

there. To perform the join, each itemset that makes up the original k-itemset is obtained and added to the end of the k-itemset. For example, if we had the 2-itemsets: *(a b), (a c), (a d), (b c), (b e), (b f)* and *(b d)*, the itemsets to be used for the join operation would be *a, b, c, d, e* and *f*. These are called the *join-itemsets*. For the join operation to be made, each join-itemset is compared to the sets that make up the 2-itemsets to avoid repetitions such as *(a b a), (b c c)*. After repetitions have been avoided and a join is done between the 2-itemsets and the join-itemsets, the results would be *(a b c), (a b d), (a c d),(b c d) , (b c f)* and *(b e f)*. The support count is then taken and the process continues until the final large itemset is obtained which could be similar to *(a b c d e f)*. The final large itemsets are then converted into fuzzy association rules.

The fuzzy association rules are generated by finding the *confidence* of each large itemset. For example given ABCD and CD as large itemsets, to determine if the rule AB $\Rightarrow$ CD holds, the confidence is obtained by finding the ratio of support(ABCD) and support(CD). If the result is greater than the minimum required confidence then the rule holds. For example, if the final large k-itemsets obtained consist of the following itemsets: *temperature low, humidity high, rain low, radiation low* and *air pressure medium.* To determine the fuzzy association rules, the confidence of the final large set must be found. To determine which of the following rules hold, the rules whose values are larger than the confidence are selected. For example, we might have to assess all the following rules:

Temperature *low* $\Rightarrow$ **Humidity** *high* and **Rain** *low* and **Radiation** *low* and *Air pressure medium*,

Temperature *low* and **Humidity** *high* $\Rightarrow$ **Rain** *low* and **Radiation** *low* and **Air pressure** *medium*,

Temperature *low* and **Humidity** *high* and **Rain** *low* $\Rightarrow$ **Radiation** *low* and **Air pressure** *medium*,

Temperature *low* and **Humidity** *high* and **Rain** *low* and *Radiation low* $\Rightarrow$ **Air pressure** *medium*

This procedure was done for every large itemset discovered. The generated fuzzy association rules were then stored in a temporary file for input to the Rule Evaluation process.

3.1.3 Rule Evaluation

The generated fuzzy rules from the temporary file must be tested for consistency with itemsets found in a test database. A test database consists of records that were not used to generate the fuzzy association rules in the previous stage, and as such serves as a control measure to determine the accuracy of the generated rules. For example, given the rule:

If **Temperature** is *High* and **Humidity** is *Med* $\Rightarrow$ **Rainfall** is *Low* and **Radiation** is *Med* and **Relative_Air_Temperature** is *Low*

The system has to make certain that for each record in the test database containing *Low* Rainfall, *Med* Radiation and *Low* Relative_Air_Temperature, *High* Temperature and *Med* Humidity also appear. If a record in the test database proves the fuzzy rule wrong, the system assumes there is an error with the fuzzy rule and checks the number of times the rule is proved both wrong and right. If the number of records in the test database showing the fuzzy rule to be true exceeds the number of records showing it is false, the rule is kept, otherwise it is discarded. The accepted rules are then stored into the Knowledge Base, from which the application system makes requests as shown in Figure 3.1.

3.1.4 Defuzzification

Fuzzy association rules need to be defuzzified to generate crisp values, so they can provide meaningful responses when user requests are made. To get the crisp value of a rule in response to a request, the 'rule strength' of the fuzzy rules have to be determined since there are usually similar fuzzy association rules in the knowledge domain. An example of similar rules are:

if **Humidity** is *low* and **Rainfall** is *high* $\Rightarrow$ **Temperature** is *low*
if **Humidity** is *low* and **Rainfall** is *med* $\Rightarrow$ **Temperature** is *low*.

For instance if we are given a record (or domain) value for the database attributes **Temperature** and **Rainfall** and there exists the following fuzzy rules:

if **Temperature** is *low* and **Rainfall** is *medium* $\Rightarrow$ **Humidity** is *high*
if **Temperature** is *low* and **Rainfall** is *high* $\Rightarrow$ **Humidity** is *high*

To get the crisp value, the membership functions of the domain values for the **humidity** attribute have to be determined from the fuzzy rule antecedents. Assume, that after calculations, the membership functions for the database record are given as follows:

if **Temperature** is *low* (0.5) **And Rainfall** is *medium* (0.3) $\Rightarrow$ **Humidity** is *high*
if **Temperature** is *low* (0.65) **And Rainfall** is *high*(0.9) $\Rightarrow$ **Humidity** is *high*

The rule strength is then obtained by applying the fuzzy *Intersection* or the logical and operator. The result would be as follows:

For the first rule: rule strength = min(0.5, 0.3) = 0.3

For the second rule: rule strength = min(0.65, 0.9) = 0.65

The fuzzy membership function value for fuzzy set *high* for the attribute **Humidity** is determined from both rules by using the fuzzy *Union* or the logical or operator. The result is therefore:

max (0.3, 0.65) = 0.65

To obtain the domain value for the **Humidity** attribute, a defuzzification technique must be applied.

To convert fuzzy association rules into crisp values, I defuzzified the rules by applying the *Center of gravity* (COG) technique. The COG technique was used because of its similarity to the Bayesian estimates. The similarity is in the fact that the value selected is supported by the knowledge accumulated from each executed rule [13], and because it provides more precise responses [41]. The COG technique involves taking the weighted average of the membership functions of a fuzzy rule. The equation is given as:

$$\frac{\Sigma(x).x}{\Sigma(x)}$$

where: x is a domain value, (x) is the membership function of x

The COG technique requires that when a membership function is mapped onto a domain, the area above the membership function (or fuzzy curve) of the particular fuzzy set be truncated[4] as shown in Figure 3.2.

The weighted average of the values from the truncated fuzzy set to the end of the Universe of Discourse are taken. Theoretically the center of gravity is calculated over

[4]The COG technique is used in papers such as [13], however this example was obtained from a non publicized source

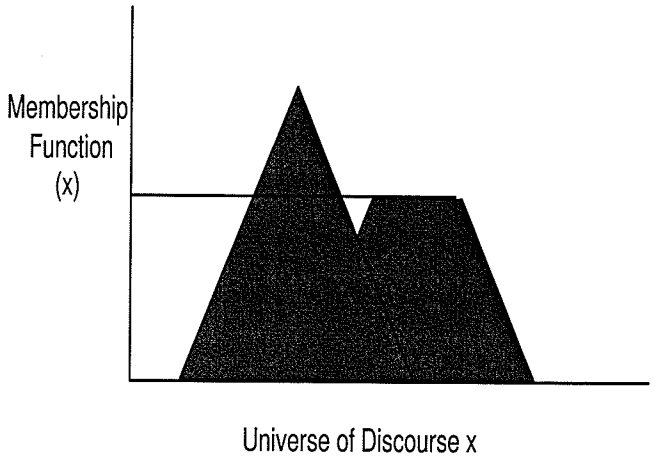

Figure 3.2: Center of Gravity Technique

a continuous succession of points. For the sake of simplicity I divided the remaining domain values into five parts and mapped their various membership functions. However, the domain could have been divided into fewer or many parts. There would, of course, be differences in the results obtained when the domain is mapped into different number of parts. Since I did not want the mapping to be too small, I picked five hoping that the difference in the result obtained for larger numbers of divisions would not be too significant. When I tried an example using five and seven parts respectively, I got a domain value of about 70 using five parts and 72 using seven parts. So the point of having a significant difference is relative, for some systems this difference might be termed as 'not so much' while other more sensitive situations might take the difference in results to be quite significant. This would be another area for future research.

The weighted average of the five parts was then calculated by applying the COG formula, which gave the crisp result for the fuzzy rule. For example, assuming a *consequent* membership value (rule strength) of 0.65 for the fuzzy set *medium*, the mapping would be made as shown in Figure 3.3.

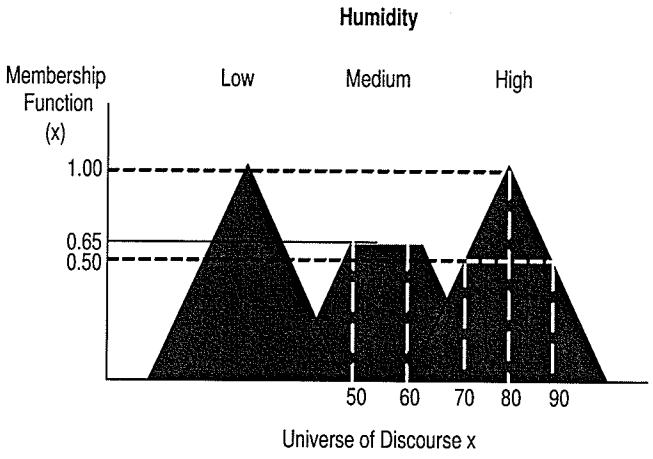

Figure 3.3: Center of Gravity Calculation

The weighted average, would be:

$$\frac{(0.65)*(50)+(0.65)*(60)+(0.50)*(70)+(1)*(80)+(0.50)*(90)}{0.65+0.65+0.50+1+0.50}=\frac{231.5}{3.3}=70.15$$

In this example, the domain was divided into five parts, as stated earlier. The division starts from the fuzzy set with the given rule strength and works down to the end of the universe of discourse as shown in Figure 3.3. The first domain value, 50, has a membership function value of 0.65 (seen in Figure 3.3). The next domain value, which is 60, has a membership value of 0.65. 70 has a membership value of approximately 0.50, 80 has a membership value of 1 and 90 has a value of 0.50. Each domain value and its corresponding membership function are multiplied. The sum of the results, is 231.5. The membership functions of all five domain values are then added to get 3.3 which is then used to divide 231.5. According to the computation, the crisp value for the fuzzy set *medium* is therefore 70.15, which would be the result given back in response to a user query.

In applying the modified-Apriori algorithm to the fuzzy database, I realised that the number of itemsets generated was very large and the procedure was therefore computationally costly. This was because for almost every attribute the algorithm had doubled the number of generated itemsets. Also, due to memory limitations most processing was done out of the system's main memory using temporary files that stored the generated itemsets. This process eased the pressure on the main memory but the I/O cost was great since the system had to read and write the file often. Although the database size was reduced to only records containing large itemsets, which reduced the scanning time, the overall execution time was still relatively high because the generated itemsets were large. The pruning of the database at each pass also contributed to the total execution time of the program. I observed that the number of records pruned at certain passes, such as the initial and final passes, gave minimal results since each pass over the database tried to count the support and prune the database as well. I then came up with the idea of limiting the number of times the database was pruned. I picked strategic passes that would make a difference in pruning the database and allowed the system to prune the database during those passes. The system was configured to prune the database at an interval of five passes. This was based on the notion that by these passes the itemsets would be large enough to make a difference in the database. This reduced the execution time considerably. I also realised that some of the itemsets had very low membership functions and were also part of the itemsets used to generate the rules. This also added to the execution time. I therefore decided to consider other methods to attempt to reduce the number of generated itemsets and thereby decrease the execution time. This resulted in an alternative system, which I call the Fuzzy-Apriori 2 (FA2) algorithm.

3.2 FA2 algorithm

The FA2 algorithm works just like the FA1 algorithm, except for the number of generated fuzzy sets. I used the *triangular* or the *trapezoidal* fuzzy curves in determining the various fuzzy membership functions as was done in FA1. I however picked the membership function with the highest truth factors to be the fuzzy sets that represents each numeric value, instead of allowing multiple fuzzy sets to represent each crisp value. For example, instead of having two or three fuzzy sets on each data attribute for each record, the records were assigned a single fuzzy set to represent them. Taking Table 3.4 as an example, the membership functions are shown in Table 3.5.

No.	Humidity	Temperature	Rainfall
1	103.00	10.80	0.20
2	49.00	8.59	0.00
3	51.00	7.82	0.15
4	85.20	16.57	0.20
5	64.80	2.13	0.10
6	53.30	6.18	0.10

Table 3.4: Sample Database

I used the fuzzy **union** operator to pick the fuzzy set with the highest membership value, which reflects a higher truth value. My aim was to reduce the number of generated itemsets and choose fuzzy sets that best described the numeric data. Table 3.6 shows the fuzzy sets picked for Table 3.4.

After fuzzifying the data, the FA1 algorithm (described in Algorithm 1) was then applied. The algorithm again generated new large itemsets using the previous large itemsets as seeds (as discussed earlier). This process was continued until the final large k-itemsets

No.	Humidity			Temperature			Rainfall		
	Low	Med	High	Low	Med	High	Low	Med	High
1	0.0	0.0	0.5	0.0	0.0	0.8	0.1	0.5	0.0
2	0.8	0.0	0.0	0.0	0.4	0.2	0.9	0.0	0.0
3	0.4	0.2	0.0	0.0	0.5	0.1	0.5	0.1	0.0
4	0.0	0.5	0.4	0.0	0.0	0.9	0.0	0.2	0.4
5	0.0	0.8	0.3	0.7	0.0	0.0	0.4	0.2	0.0
6	0.2	0.6	0.0	0.4	0.5	0.3	0.7	0.3	0.0

Table 3.5: Sample Membership Functions using FA2

No.	Humidity	Temperature	Rainfall
1	High	High	Med
2	Low	Med	Low
3	Low	Med	Low
4	Med	High	High
5	Med	Low	Low
6	Med	Med	Low

Table 3.6: Sample Fuzzy Database using FA2

were obtained. These itemsets were then used to generate the fuzzy association rules. The database was also pruned at intervals of five passes as done in FA1 to reduce the total execution time.

Fuzzy rule evaluation was done as in FA1. In converting fuzzy association rules into crisp values, I defuzzified the rules by again applying the *Center of gravity* technique, as was done in FA1.

Though FA2 had a lower execution time than FA1, it generated a large number of

similar rules. Addressing the problem of large numbers of similar rules is an area of potential future work.

The aim of both the FA1 and FA2 algorithms is to mine the data obtained from the NAWN stations and hopefully develop techniques that will also be applicable elsewhere. In the next section I evaluate the performance of the FA1 and FA2 algorithms in the application domain.

Chapter 4

Evaluation: Mining the NAWN Domain

For the purpose of evaluating this research I used the numeric data available from the Niagara Agricultural Weather Network (NAWN) to assess the performance of the two developed algorithms. Considering the size of data available from NAWN (collected from 1995 to 2002), and the complexities involved in dealing with numeric data, it was felt that the data would serve as a suitable real-life test bed for assessing the algorithms.

The NAWN network collects 24-hour weather readings from its different stations in the region. Data readings are obtained for data attributes such as *temperature-in-vines, radiation, rainfall, air-temperature, relative humidity* etc. There were a total of about 2 million records available for the experiments.

The experiments were performed on a Windows based platform with a Pentium III-500 processor and 512MB of main memory. The data were stored in text files having a total size of about 3.79GB which was stored on a local ultra DMA66, EIDE hard disk.

4.1 Experimental Process

The running time for the algorithms can be divided into three parts:

1. *Large Itemset Generation.* The time taken to generate the next large itemsets using the previous large itemsets as seeds. The large itemset generation time is independent of the number of records in the database. This is because it deals with joining previous large itemsets, that have been saved, to produce the next set of large itemsets. The generated large itemsets with small subsets are also pruned off at this phase. The database is not accessed during the large itemset generation process since only the generated itemsets are required. During certain stages of the generation of the large itemsets, the seeds tend to create exponential numbers of potential large itemsets as compared to the number of seed sets used to generate them. The time taken to initially create these large sets tends to be large and as such adds significantly to the total execution time. As the numbers of generated itemsets get smaller and as the algorithm proceeds towards the final large itemset(s), the numbers of generated itemsets tends to decrease and the system's execution time improves.

2. *Support Counting.* The time taken to count the support of generated potential large itemsets is directly dependent on the number of records available in the database and the number of potential large itemsets. As the number of records in the database increases, the support counting time increases. Though I tried to remove records without large itemsets at each pass, I observed that the number of records pruned during certain passes, such as the initial and final passes, were minimal in both the FA1 and FA2 systems (as stated in Chapter 3). I picked strategic passes that would make a difference in pruning the database and allowed the system to prune the database during those passes. The system was configured to

prune the database at an interval of five passes. This reduced the execution time considerably as will be shown.

3. *Rule Evaluation.* The time taken to evaluate each rule generated depends on the number of large itemsets discovered and the size of the test database given to evaluate the rules. The number of rules generated also depends on the confidence value given. The test database consisted of 10,000 records, and they were used for every experiment performed.

The portions of data used from the NAWN data for the experiments were in chunks. This may have also affected some of the results since i

4.2 Interest Rule Measure

Figure 4.1 shows the number of rules obtained using both algorithms (FA1 and FA2) at a confidence and support level of 30% and 20% respectively as the number of fuzzy sets was increased from 3 to 11 on 20,000 records. A low support and confidence value were given in this experiment to determine the number of rules that could be generated and also as a starting point from which later comparisons with higher support and confidence values could be made.

As shown in Figure 4.1, the number of rules generated by the FA1 algorithm was much less than the number of rules generated by the FA2 algorithm. This was because there were so many itemsets generated since each crisp value can be represented by as many fuzzy sets as possible. This necessarily leads the FA1 algorithm to have such rules as:

If **Temperature** is *High* and **Temperature** is *Med* and **Humidity** is *Med*

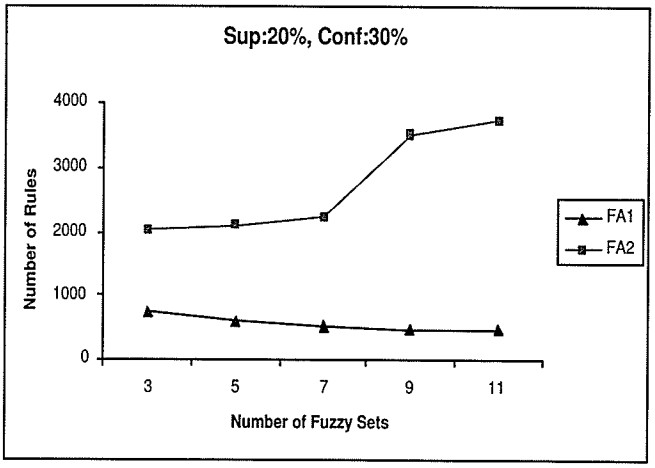

Figure 4.1: Experimental Results Comparing the Number of Rules generated by Algorithms FA1 and FA2.

> and **Humidity** is *Low* $\Rightarrow$ **Rainfall** is *Low* and **Rainfall** is *Med* and **Radiation** is *High* and **Radiation** is *Med* and **Radiation** is *low* and **Relative_Air_Temperature** is *Low*

Also because of the multiple fuzzy sets available for each database attribute, the final large itemsets eventually obtained tend to be very few since the multiple fuzzy sets do not cover every record in the database. Simply put, as long as a join operation occurs and the new itemset generated appears in the database, the number of potential final large itemsets is reduced. This is because the large itemsets are combined with a join-itemset after each join operation and therfore changed. For example, while FA2 might have a final large itemset as:

> **Temperature** *low*, **Humidity** *med*, **Radiation** *high*

FA1 might also generate the above itemset but on performing a join operation, it might generate:

Temperature *low*, **Humidity** *med*, **Radiation** *high*, **Temperature** *med*

which might also be contained the database. However not all records in the FA1 system have the newly generated itemsets and as such a lot of the generated itemsets would be pruned off, leaving a small number of large itemsets.

The FA2 algorithm on the other hand, would not be able to generate

Temperature *low*, **Humidity** *med*, **Radiation** *high*, **Temperature** *med*,

because none of its database records have multiple fuzzy sets and as such would rather generate similar itemsets which are larger in number than the number generated by FA1. The FA2 algorithm leads to such similar fuzzy rules as:

> If **Temperature** is *High* and **Humidity** is *Med* $\Rightarrow$ **Rainfall** is *Low* and **Radiation** is *Med* and **Relative_Air_Temperature** is *Low*

and

> If **Temperature** is *Med* and **Humidity** is *Med* $\Rightarrow$ **Rainfall** is *Low* and **Radiation** is *Med* and **Relative_Air_Temperature** is *Low*

and not[1]

> If **Temperature** is *High* and **Temperature** is *med* and **Humidity** is *Med* $\Rightarrow$ **Rainfall** is *Low* and **Radiation** is *Med* and **Relative_Air_Temperature** is *Low*

[1]Please note that these are fuzzy rules and not boolean predicates.

The number of rules generated by the FA1 algorithm decreased as the number of fuzzy sets increased for the 20,000 records in the experiment. I hypothesized that this was due to the itemset dispersion, it was possible that only a few itemsets dominated the entire data used for the experiment, thereby resulting in fewer rules. FA2 would also have its support decreased for each fuzzy set under such conditions but the number of generated similar rules tends to over ride the support decrease.

To test the hypothesis, a similar experiment to the one shown in Figure 4.1 was also conducted (results shown in Figure 4.2), using a different set of records for the experiment. I again used 20,000 records but they were taken from a different portion of the NAWN data. The experiments were again performed with a confidence and support level of 20% and 30% respectively and the rules were again obtained as the number of fuzzy sets was increased from 3 to 11. In this experiment the rules generated by the FA1 algorithm increased slowly but steadily as the number of fuzzy sets increased. This was because the records were not dominated by a small number of itemsets but instead had a nearly even dispersion. This resulted in a larger number of final large itemsets that could then be used to generate the fuzzy association rules. FA2, on the other hand, still behaved much as it did in Figure 4.1.

The salient conclusion that can be drawn from the experimental results shown in Figures 4.1 and 4.2 is that the data dispersion may significantly affect the number of rules generated by FA1. Though FA2 might be affected by the data dispersion as well, this change does not show clearly since the numbers of rules obtained tends to always increase due to the number of similar rules generated.

Experiments were also conducted to compare the changes in the number of rules generated as the support and confidence values were changed. The experiments were conducted using the same 20,000 records that were used in the experiment presented in Figure 4.2. The results are shown in Figures 4.3, 4.4 and 4.5 (which is a combination of

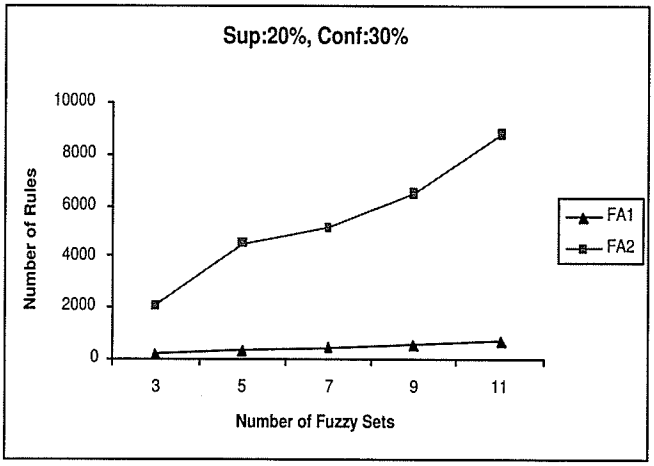

Figure 4.2: Experimental Results Comparing the Number of Rules generated by Algorithms FA1 and FA2 using a different set of 20,000 records.

4.3 and 4.4).

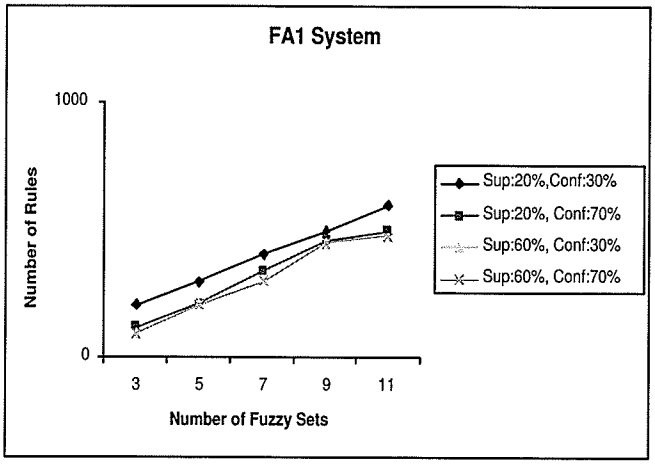

Figure 4.3: Experimental Results Comparing the Number of Rules generated by the FA1 algorithm using different support and confidence values.

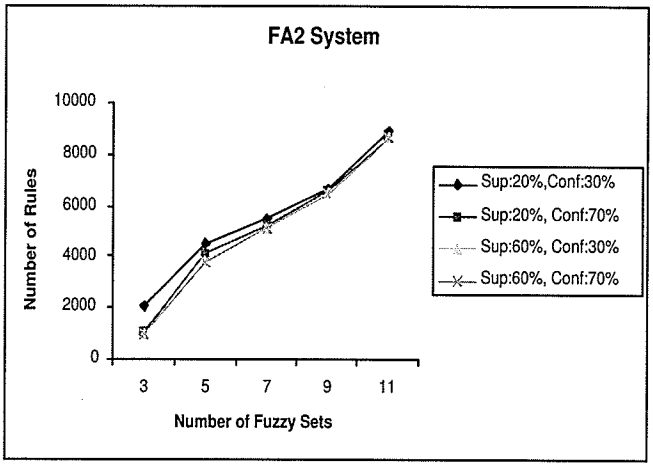

Figure 4.4: Experimental Results Comparing the Number of Rules generated the FA2 algorithm using different support and confidence values.

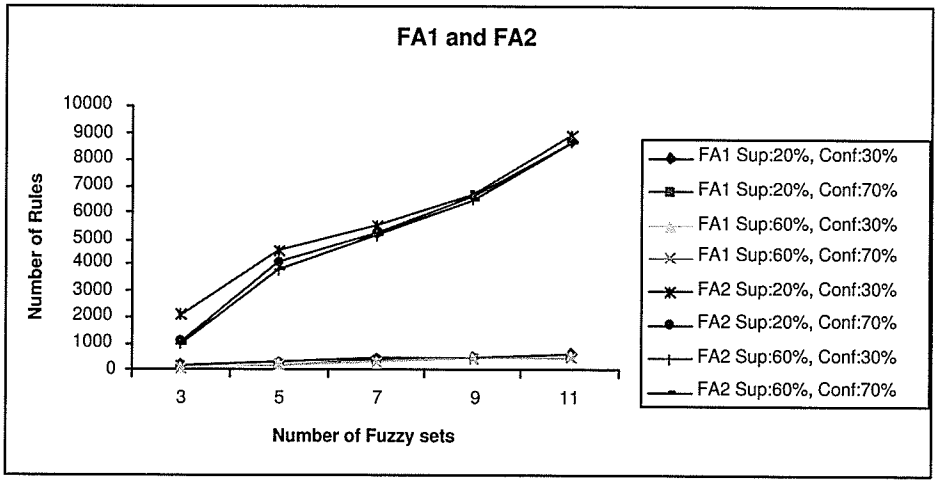

Figure 4.5: Experimental Results Comparing the Number of Rules generated by the FA1 and FA2 algorithms using different support and confidence values.

From Figures 4.3, 4.4 and 4.5, it can be seen that as the support value increases from 20% to 60% and the confidence from 30% to 70%, the number of rules generated by both the FA1 and FA2 algorithms tends to decrease. I noticed that at a support value of 60% the number of rules generated at a confidence value of 30% and 70% were the same for both the FA1 and FA2 algorithms, and so the difference between the graphs of support 60% and the varying confidence values are not clearly seen. This implies that the generated rules were so prominent in the data that no matter how much change was made to the confidence, the same rules were still generated. The figures also showed that at support of 20%, the number of rules generated at a confidence of 30% and 70% differed considerably between the algorithms. The graph showed that the difference in the number of rules generated by FA2 using 11 fuzzy sets was not very significant. This was not due to any particular reason other than that the difference in the number of rules generated was very small. The number of rules generated with support of 20% and confidence of 70% and higher were less than those generated with support of 20% and confidence of 30% but only by about ten rules. This explains why the graphs are so close to each other.

In brief, these experiments show that at low support values the confidence value affects the number of rules generated while at higher support values the effect of the confidence tends to be less. In these experiments there was no difference at all in the number of rules generated at a support of 60% and with different confidence values. This is why the 'Sup:60%, Conf:70%' graphs in both FA1 and FA2 seem to be missing. I expected that if I used a larger amount of data that it would be possible to get a considerable difference in the number of rules generated, since it is possible that the earlier data used might have limited number of rules.

Experiments were also performed using 100,000 records but on only 3 and 5 fuzzy sets. I chose 3 and 5 fuzzy sets because they had considerable differences in the number

of rules generated in earlier experiments compared to other number of fuzzy sets, so I felt they would reveal even the slightest change and therefore serve as a good, small test bed. The specific aim of the experiment was to determine if there could be any change in the number of rules generated at a moderate support level of 60% given differing confidence values. The results are given in Figures 4.6 and 4.7(which is an inverted version of Figure 4.6, which shows that there were essentially no changes in the number of rules generated at a support of 60% for different confidence values. I again speculated that it was possible that the data dispersion might have been the cause. The itemsets that made up the generated rules seem to consistently appear together in the database. For example, to generate different numbers of fuzzy rules at different confidence values, one is assuming, for example, that itemsets such as the following exist:

Humidity *low*, **Temperature** *high*, **Temperature** *med*, **Rainfall** *low*
Humidity *low*, **Temperature** *high*, **Radiation** *med*, **Rainfall** *med*
Humidity *low*, **Temperature** *high*, **Temperature** *med*, **Rainfall** *med*

If this is the case, then by the time the confidence is counted at about 30%, it is possible that **Radiation** occurs sufficiently enough to be counted and thus is added to the rules generated. At a 70% level of confidence, however, it is possible that the number of times **Radiation** occurs with the rest of the itemsets in the database is reduced so that it would be pruned off, thereby reducing the number of rules generated. It seems that almost every final itemset generated at a considerable support level tends to have significant numbers of itemsets in the data. Also considering the fact that the data domain is from weather readings, it is typical to have values that appear only when certain other weather conditions occur. So one can simply conclude that the data dispersion does indeed have an impact. It is, of course, also possible for the algorithms (FA1 and FA2) to be applied to some other database, such as a grocery database, to see if different result from those

seen in the experiments are obtained. This is an area for future research.

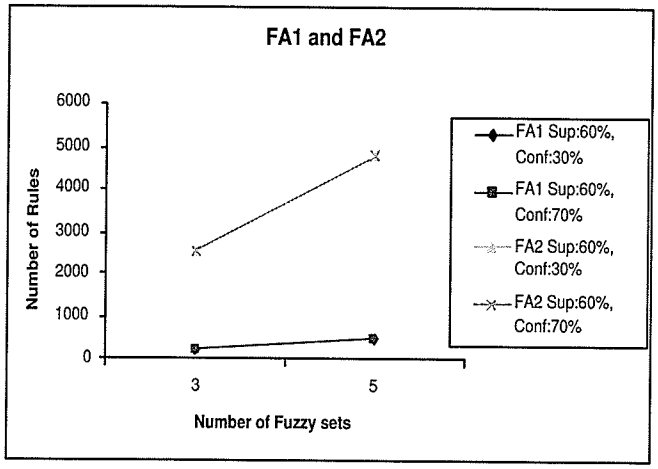

Figure 4.6: Experimental Results Comparing the Number of Rules generated by FA1 and FA2 using 100,000 records.

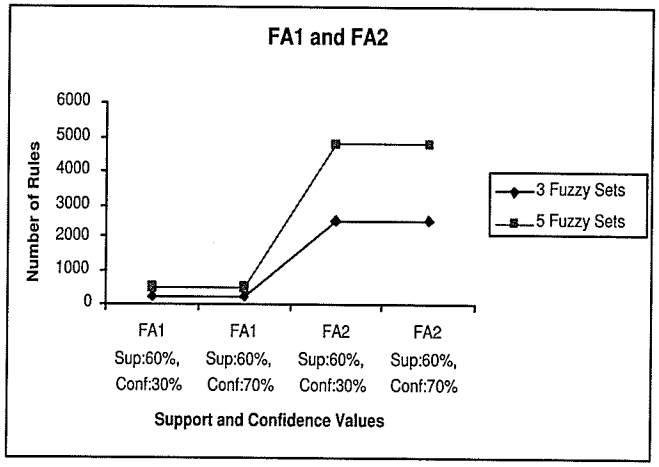

Figure 4.7: Experimental Results Comparing the Number of Rules generated by FA1 and FA2 using 100,000 records having the support and confidence values as the x-axis.

I also compared results from my algorithms with the Fuzzy Transaction Data-mining

(FTDA) algorithm [28, 29]. Three fuzzy sets were used for each algorithm as the number of record was increased from 10,000 to 100,000 records. Three fuzzy sets were chosen first for simplicity and second because, using three fuzzy sets seems to be typical in fuzzy systems. Figure 4.8 shows the results.

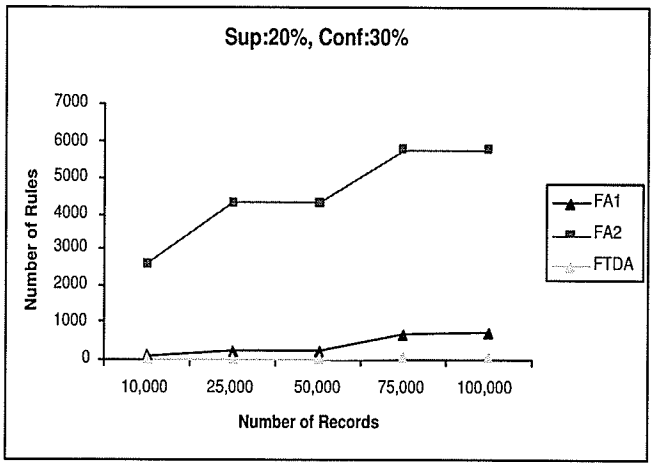

Figure 4.8: Experimental Results Comparing the Number of Rules generated from Algorithms FA1, FA2 and FTDA

The FTDA algorithm was considered because it too is a fuzzy based Apriori system. However the algorithm generates its large sets based on attributes' cardinalities and not directly on the available data. The FTDA algorithm also initially removes fuzzy sets with low cardinality values thereby limiting the number of fuzzy sets that can later be used in the algorithm. The FA1 and FA2 algorithms allow every possible fuzzy set to be represented and used. FTDA also takes a union of the cardinality values of its database attributes in generating its k-itemsets and prunes off all attribute combinations with membership values lower than the support. This process reduces the number of large sets generated considerably. Hence, FTDA should generate fewer rules than those obtained by both the FA1 and FA2 algorithms. This is confirmed in Figure 4.8. Higher

number of rules tend to give the notion that more itemset relationships have been found in a database.

4.3 Algorithm Scalability

A set of larger experiments (processing more records) were also done to assess the scalability of the algorithms with respect to execution time. The results of these experiments are shown in Figures 4.9 and 4.10. Again, three fuzzy sets were used for each algorithm.

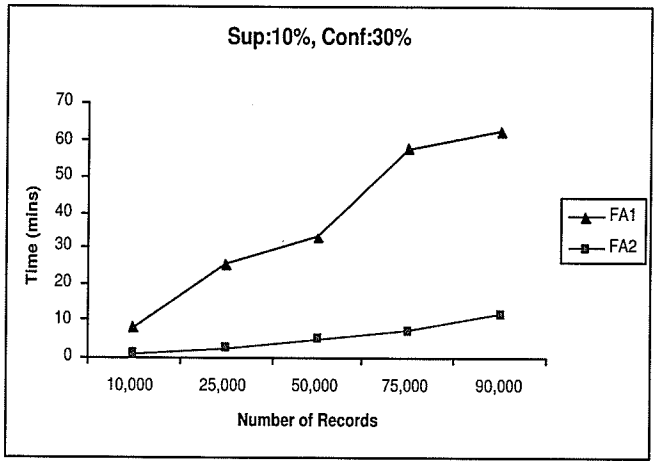

Figure 4.9: Experimental Results Comparing the Time taken by the Algorithms FA1 and FA2 with up to 90,000 records

I expected a near-linear scale-up in execution time as the number of records increased. This was because the algorithms had more data to deal with and as the number of records increased, the longer the time needed to complete a database scan. Also, larger numbers of large itemsets were expected to be generated as the record size increased, and so

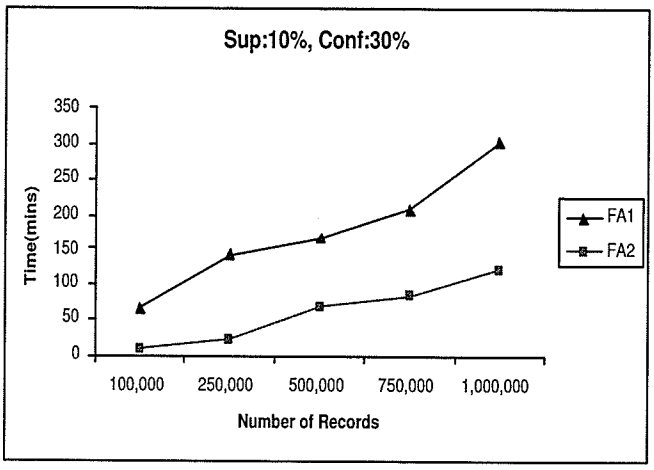

Figure 4.10: Experimental Results Comparing the Time taken by the Algorithms FA1 and FA2 with up to 1,000,000 records.

the algorithms would have more itemsets to count, and this would inevitably increase its execution time. The number of generated rules were expected to increase as well with record size, and so evaluating the rules for correctness would also add to the execution time. From my scalability experiments, the support value was set to 10% and the confidence was kept at 30%. I lowered the support value so that the algorithms would generate more rules and therefore take longer to execute. The aim was to observe how scalable the algorithm would be under near worst case conditions.

The FA2 algorithm continued to be more effective than the FA1 algorithm due to the reduced number of joins that had to be performed. The FA1 algorithm had to deal with almost three times the number of fuzzy sets that the FA2 algorithm did thereby causing its execution time to be about three times that of the FA2 algorithm.

The portions of data used from the NAWN data for the experiments were taken in chunks. This may have also affected some of the results obtained, however it showed

that there is a pattern amongst the data itemsets especially during the experiments in which varying support and confidence values were used. For future research I hope to try using random data from various areas of the NAWN data and compare the results with those obtaned here.

4.4 Defuzzification

The application system which contains the Defuzzification module shown in Figure 3.1, was used with both the FA1 and FA2 algorithms with no changes made to the defuzzification technique. When queries were given to rules obtained from FA1 and FA2, the results obtained were sometimes quite similar and at other times quite different. For example if FA1 gave the rule:

> **Temperature** is *low*(0.85) and **Temperature** is *med*(0.55) and **Humidity** is *med*(0.65) $\Rightarrow$ **Rainfall** is *high*

while FA2 gave the rule:

> **Temperature** is *low*(0.85) and **Humidity** is *med*(0.65) $\Rightarrow$ **Rainfall** is *high*

Applying the rule strength calculation discussed in Section 3.1.4 to the rules just presented, the results would be:

> **FA1**
>
> **rule strength = min(0.85, 0.55 0.65) = 0.55**
>
> **FA2**
>
> **rule strength = min(0.85, 0.65) = 0.65**

This leads to different results from the two systems. Since FA2 may have a number of similar rules it is possible that it might discover the following:

Temperature is *med*(0.55) and **Humidity** is *med*(0.65) $\Rightarrow$ **Rainfall** is *high*

Temperature is *low*(0.85) and **Humidity** is *med*(0.50) $\Rightarrow$ **Rainfall** is *high*

computing the rule strengths for these rules we get:

FA2

rule strength = min(0.55 0.65) = 0.55

rule strength = min(0.85, 0.50) = 0.50

Applying the fuzzy *Union* operator to determine the membership function of the attribute **Rainfall**, the result would be:

FA2: max(0.55, 0.50) = 0.55

This is the same result obtained in FA1.

The difference in defuzzification results in some instances could be a basis to determine which system an organisation might choose to use. For organisations that would not be affected by a small number difference, for example, an organisation that would consider 80 and 82 or 0.04 and 0.06 in the same category and have a large amount of data to mine, the FA2 system would be most suitable. However for numeric sensitive environments that might consider 0.04 and 0.06 as two completely different results and would prefer every fuzzy set available to an itemset to have an effect in its result, FA1 might be a better option. FA1 would probably be best chosen if the organisation had only a small amount of data to mine since FA1 takes a considerable amount of time to do the mining and that the time is proportional to the size of the database.

Due to the fuzzifying and defuzzifying process, it can not be claimed that the precise crisp result obtained is the precise result. Therefore a range of values would be given back to the user based on the result obtained from the defuzzification technique. For example, for a defuzzified result of 70.15, the user might get the range [69, 71] or more depending on the system use.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

In this thesis, I have presented two algorithms, Fuzzy-Apriori 1 (FA1) and the Fuzzy-Apriori 2 (FA2), for discovering fuzzy association rules in large numeric databases. I dealt with quantitative data by converting them into fuzzy sets and hence making them suitable for the application of categorical data mining algorithms. I then applied a modified version of the Apriori algorithm to generate the required fuzzy association rules. I also developed a defuzzification technique based on the center of gravity technique to convert fuzzy results to crisp values for display to users.

The FA1 algorithm is a simple implementation that uses every fuzzy set that can be represented by each numeric value. This causes some values to be represented by one, two or more fuzzy sets depending on the number of fuzzy sets used in the system. The FA1 algorithm generates a considerable number of rules and takes a significant amount of execution time due to the large number of itemsets it has to deal with. The FA2 algorithm optimizes the FA1 algorithm by reducing the number of fuzzy sets assigned to

each numeric value to one and thereby reducing the overall execution time. The reduction in the number of fuzzy sets assigned to each crisp value in FA2, however, has the effect of increasing the number of rules generated by the algorithm.

It was shown experimentally that the number of rules generated by the FA1 algorithm was much less than the number generated by the FA2 algorithm. This was because the FA1 algorithm had to keep performing join operations on its generated itemsets until they no longer existed in the database. Further, since the FA1 algorithm had a lot of itemsets to deal with, it had a higher number of join operations as well. After a certain stage in its join operations, the number of generated itemsets did decrease considerably thereby resulting in fewer generated rules. The FA2 algorithm on the other hand, had a limited number of join operations to perform and therefore had a larger number of itemsets that were finally used to generate its rules. Though FA1 was significantly slower when compared to FA2, the FA2 algorithm generated a lot of similar rules.

I also compared the number of rules generated by the FA1 and FA2 algorithms with the number generated by the FTDA algorithm. The results obtained showed that both the proposed algorithms generated more rules than FTDA. This is because FTDA was designed exclusively to minimize execution time even at the expense of the number of generated rules. The FA1 and FA2 algorithms are aimed at generating as many rules as possible. The large number of rules generated give the notion of generating a lot of relationships between itemsets, that is of course not including the similar rules generated by FA2.

The results of other experiments showed that, as the support and confidence values got higher, the numbers of rules generated by both algorithms tended to be lower. However the experiments also showed that at a relatively high support value, the effects of the change in confidence values was not as significant. Even when further experiments were conducted on a larger set of data, the same results were obtained. I believe that

this result likely occurred because of the data dispersion, since our domain comprised of weather readings, which usually have specific patterns. It is possible that there might be a considerable difference in the number of rules generated using data from other databases.

The rules generated by both algorithms are often quite similar but at other times they differed. This similarity and difference is clearly seen in the output generated after defuzzification has been applied.

The performance of the FA1 and FA2 algorithms was also shown to scale approximately linearly with the number of records processed. Experiments showed that the FA2 algorithm consistently performed better than the FA1 algorithm decreasing the total execution time by almost sixty percent.

The FA2 algorithm is the preferred algorithm for use with the Niagara Agricultural Weather Network (NAWN), since it runs faster than the FA1 algorithm and the size of the NAWN data is very large. For other, more numerically sensitive areas and with smaller amounts of data, the FA1 algorithm might be a better option since it uses multiple fuzzy sets and generates a more limited number of rules than the FA2 algorithm.

In summary, in this thesis, I introduced two new algorithms with different characteristics for mining association rules using fuzzy logic to handle numeric data. I also conducted a series of experiments to assess and compare the characteristics and effectiveness of the two algorithms relative to each other and to another association rule mining algorithm that uses fuzzy logic (the Fuzzy Transaction Data-mining algorithm). The organisation doing the mining and its requirements, would inevitably determine which of the two algorithms (FA1 and FA2) would be most suitable for use.

5.2 Future Work

There are many opportunities to extend and improve on the research work presented in this thesis. These are either in the area of improvements to the existing system or possible extensions to the algorithms.

5.2.1 System Improvement

In my algorithm implementations, I reduced the pressure on system memory by storing and accessing the generated itemsets using temporary files. This increased the execution time of the system significantly. Future work might include improving both the FA1 and FA2 algorithms in the area of memory management and I/O processing, thereby reducing execution time. For example, pre-fetching itemsets into memory just before they are actually needed could provide significant improvements

Another important area of future work would be to improve the data structures used. To implement the algorithms I did not use a near-optimal search technique and as such could easily improve on it to minimize execution time. Also, for this research, the database and itemsets were not ordered, so it would be interesting to determine if sorting the database and itemsets could reduce execution time and by how much.

Performing experiments using different levels of memory for storing various data items would also be interesting. This would allow the determination of cost-performance trade-offs that would be useful to potential users of the system.

Reducing the number of similar rules generated by the FA2 algorithm would also be an improvement. If the similar rules generated by FA2 can be compacted into non-similar rules, then the problem of generating so many rules can be reduced. Also determining a way of reducing the execution time of FA1 can allow the algorithm to be used on very

large databases as well without taking too much time.

Determining the effects of domain division during the application of the center of gravity technique for defuzzification can be another interesting area of further research. If it can be determined that the number of divisions used on a domain can significantly affect the type of results expected for organisations, the number of divisions could then be made to suit organisational requirements.

Further research can also be done to determine if it is possible to generate different amounts of fuzzy rules at a considerable support value with differing confidence values using a different test domain, such as data obtained from a grocery store.

5.2.2 Possible Extensions

Some possible extensions include combining fuzzy logic with other association rule mining algorithms such as the Partition or DHP algorithms to mine numeric data and then to compare the results with those obtained in my thesis research. It may also be possible to integrate fuzzy logic with different types of data mining techniques such as those that use clustering techniques. It would be interesting to see what trade-offs would arise in such systems.

Parallel techniques have been shown in the research to significantly reduce the execution time in mining data. Fuzzy logic could also be combined with parallel data mining algorithms such as the Equivalence Class Transformation algorithm (ECT) and applied to numeric databases. This would be particularly interesting to explore since ECT has been tested on categorical data alone.

A final possible extension would be to determine other means of obtaining the required fuzzy rules such as using clustering techniques, instead of using an expert's ad-

vice. This would avoid human errors from experts and also provide fuzzy sets that might better suit the particular data to be mined.

Bibliography

[1] S. K. Harms and S. Goddard and S. E. Reichenbach and W. J. Waltman and T. Tadesse. Data Mining in a Geospatial Decision Support System for Drought Risk Management. National Conference on Digital Government Research, Digital Government Research Center, Lincoln, USA, August 2001.

[2] U. Straccia. A Fuzzy Description Logic. In *Proc. of AAAI-98, 15th Conference of the American Association for Artificial Intelligence, Madison, USA*, pages 594–599. AAAI Press/MIT Press, July 1998.

[3] A. W. Fu and M. H. Wong and S. C. Sze and W. C. Wong and W. L. Wong and W. K. Yu. Finding Fuzzy Sets for the Mining of Fuzzy Association Rules for Numerical Attributes. In L. Xu, L. W. Chan, I. King, and A. Fu, editors, *Proc. of 1st Intl. Symposium on Intelligent Data Engineering and Learning (IDEAL'98)*, pages 263–268. Springer-Verlag, October 1998.

[4] R. Agrawal, T. Imielinski, and A. N. Swami. Mining Association Rules between Sets of Items in Large Databases. In P. Buneman and S. Jajodia, editors, *Proc of the 1993 ACM SIGMOD International Conference on Management of Data*, volume 22, pages 207–216, Washington, D.C., May 1993.

[5] R. Agrawal and J. C. Shafer. Parallel Mining of Association Rules. *IEEE Transactions On Knowledge And Data Engineering*, 8(6):962–969, 1996.

[6] A. D. C. Bennett, J. B. Paris, and A. Vencovska. A New Criterion for Comparing Fuzzy Logics for Uncertain Reasoning. *Journal of Logic, Language and Information*, 9(1):31–63, January 2000.

[7] J. C. Bezdek. *Pattern Recognition with Fuzzy Objective Function Algorithms*. Plenum Press, New York, 1981.

[8] C. Borgelt and M. R. Berthold. Mining Molecular Fragments: Finding Relevant Substructures of Molecules. In *Proc. of IEEE Data Mining, Japan*, December 2002.

[9] J.-F. Boulicaut and A. Bykowski. Frequent Closures as a Concise Representation for Binary Data Mining. In *Proc. of the Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 62–73, 2000.

[10] C. H. Cai, A. W. Fu, C. H. Cheng, and W. W. Kwong. Mining Association Rules with Weighted Items. In *Proc of the International Database Engineering and Application Symposium*, pages 68–77. IEEE Comput. Soc., 1998.

[11] S. Chen, B. Liu, W. Hsu, and Y. Ma. Analyzing the Subjective Interestingness of Association Rules. *IEEE Intelligent Systems*, 15(5):47–55, September 2000.

[12] K. J. Cios and J. Kacprzyk, editors. *Medical Data Mining and Knowledge Discovery*, volume 60 of *Studies in Fuzziness and Soft Computing*. Physica Verlag, 1st edition, January 2001.

[13] E. Cox. *The Fuzzy Systems Habndbook*. AP Professional, Chappaqua, New York, second edition, 1998.

[14] D. Riordan and B. K. Hansen. Fuzzy Case-Based Prediction of Cloud Ceiling and Visibility. In *3rd Conference on Artificial Intelligence, American Meteorological Society*, pages 118–123. American Meteorological Society, August 1998.

[15] J. M. de Graaf, W. A. Kosters, and J. J. W. Witteman. Interesting Fuzzy Association Rules in Quantitative Databases. *Lecture Notes in Computer Science, Springer Verlag*, 2168:140–151, 2001.

[16] M. Delgado and Gonzalez. An Introductive Learning Procedure to Identify Fuzzy Systems. *Fuzzy Sets and Systems*, 55:121–132, 1993.

[17] M. Delgado and A. Gonzalez. A Frequency Model in a Fuzzy Environment. *International Journal of Approximate Reasoning*, 11:159–174, 1994.

[18] P. M. Deshpande, K. Ramasamy, A. Shukla, and J. F. Naughton. Caching Multidimensional Queries using Chunks. In *Proc. of the 1998 ACM SIGMOD International Conference on Management of Data*, volume 27, pages 259–270. ACM, June 1998.

[19] D. Dubois and H. Prade. *Fuzzy Sets and Systems: Theory and Applications*. Academic Press, 1980.

[20] E. Omiecinsky and A. Sarasere and S. Navathe. An Efficient Algorithm for Mining Association Rules in Large Databases. In *Proc. of the 21st International Conference on Very Large Data Bases Conference Zurich, Switzerland*, pages 432–444. M. Kaufmann, September 1995.

[21] R. Elmasri and S. Navathe. *Fundamentals of Database Systems*. The Benjamin/Cummings Publishing Company, Inc., Redwood City, CA, USA, 1994.

[22] G. Gupta, A. Strehl, and J. Ghosh. Distance based clustering of association rules. In *Intelligent Engineering Systems Through Artificial Neural Networks (Proceedings of ANNIE 1999)*, volume 9, pages 759–764. ASME Press, November 1999.

[23] A. Gyenesei. Mining Weighted Association Rules for Fuzzy Quantitative Items. In *Principles of Data Mining and Knowledge Discovery. 4th European Conference, PKDD 2000*, pages 416–423. Springer–Verlag, September 2000.

[24] A. Gyenesei. A Fuzzy Approach for Mining Quantitative Association Rules. In *Second Conference on Computer Science*, volume 15, pages 305–320, July 2001.

[25] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers, August 2000.

[26] B. K. Hansen. Weather Prediction Using Case-Based Reasoning and Fuzzy Set Theory. Master's thesis, Department of Computer Science, Dalhousie University, 2000.

[27] M. Holsheimer, M. L. Kersten, H. Mannila, and H. Toivonen. A Perspective on Databases and Data Mining. In *1st International Conference on Knowledge Discovery and Data Mining*, pages 150–155. Centrum voor Wiskunde en Informatica (CWI), 1995.

[28] T. Hong, C. Kuo, and S. Chi. A Fuzzy Data Mining Algorithm for Quantitative Values. In L. C. Jain, editor, *Third International Conference on Knowledge-Based Intelligent Information Eng. Systems*, pages 480–483. IEEE, August 1999.

[29] T. Hong, C. Kuo, and S. Chi. Mining Association Rules from Quantitative Data. *Intelligent Data Analysis*, 3(5), November 1999.

[30] T. Hong, C. Kuo, and S. Chi. Trade-Off Between Computation Time and Number of Rules for Fuzzy Mining from Quantitative Data. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 9(5):587–604, October 2001.

[31] M. A. Houtsma and A. N. Swami. Set-Oriented Mining for Association Rules in Relational Databases. In P. S. Yu and A. L. P. Chen, editor, *Proc of the Eleventh International Conference on Data Engineering*, pages 25–33. IEEE Computer Society, March 1995.

[32] J. Hipp and U. Guntzer and G. Nakaeizadeh. Algorithms for Association Rule Mining - A General Survey and Comparison. *SIGKDD Explorations*, 2(1):58–64, July 2000.

[33] J. L. Han and A. W. Plank. Background for Association Rules and Cost Estimate of Selected Mining Algorithms. In K. Barker and M. T. Ozsu, editors, *Proc. of the Fifth International Conference on Information and Knowledge Management, CIKM '96, Rockville, Maryland, USA*, pages 73–80. ACM, November 1996.

[34] K. Grabczewski and W. Duch and R. Adamczak. Extraction of Crisp Logical Rules using Constrained Backpropagation Networks. In M. Verleysen, editor, *Proc. of European Symposium on Artificial Neural Networks. ESANN '97*, pages 109–114, April 1997.

[35] B. Kosko. Fuzzy Systems as Universal Approximators. *IEEE Transactions on Computers*, 43(11):1329–1333, November 1994.

[36] B. Kosko and S. Mitaim. The Shape of Fuzzy Sets in Adaptive Function Approximation. *IEEE Transactions on Fuzzy Systems*, 9(4):637–656, August 2001.

[37] B. Kosko and P. J. Pacini. Adaptive Fuzzy Frequency Hopper. *IEEE Transactions on Communications*, 43(6):2111–2117, June 1995.

[38] S Kudyba and R. Hoptroff. *Data Mining and Business Intelligence: A Guide to Productivity.* Idea Group Publishing, February 2001.

[39] C. M. Kuok, A. W. C. Fu, and M. H. Wong. Mining Fuzzy Association Rules in Databases. *SIGMOD Record*, 27(1):41–46, March 1998.

[40] L. A. Zadeh. Outline of a New Approach to the Analysis of Complex Systems and Decision Processes. *IEEE Transaction on System, Man, and Cybernetics*, SMC-3(1):28–44, January 1973.

[41] A. Iriarte Lanas, M.A. Pacheco, M.M. Vellasco, and R. Tanscheit. Neuro-Fuzzy Control of a Multivariable Nonlinear Process. In *The 7th European Congress on Intelligent Techniques and Soft Computing. EUFIT '99 - Magdeburg, Germany*, pages 13–16, 1999.

[42] M. Chen and J. Han and P. Yu. Data Mining: An Overview from Database Perspective. *IEEE Transactions on Knowledge and Data Eng.*, 8:866–883, December 1996.

[43] M. J. Zaki and S. Parthasarathy and M. Ogihara and W. Li. New Algorithms for Fast Discovery of Association Rules. In D. Heckerman, H. Mannila, D. Pregibon, and R. Uthurusamy, editors, *Proc. of the 3rd International Conference Knowledge Discovery and Data Mining, AAAI Press, Menlo Park, California*, pages 283–286, August 1997.

[44] M. Mehta, R. Agrawal, and J. Rissanen. SLIQ: A Fast Scalable Classifier for Data Mining. In S. Apers, M. Bouzeghoub, and G. Gardarin, editors, *Extending Database Technology*, pages 18–32. Springer–Verlag, 1996.

[45] M. Mehta, R. Agrawal, and J. Shafer. SPRINT: A Scalable Parallel Classifier for Data Mining. In *Proc. of the 22nd International Conference Very Large Data Bases, Bombay, India*, pages 544–555. Morgan Kaufmann, September 1996.

[46] B. Nag, P. Deshpande, and D. J. DeWitt. Using a Knowledge Cache for Interactive Discovery of Association Rules. In *Proc. of the International Conference on Data Mining and Knowledge Discovery (KDD'99)*, pages 244–253, 1999.

[47] D. Nauck, F. Klawonn, and R. Kruse. Combining Neural Networks and Fuzzy Controllers. In Klement, P. Erich, and W. Slany, editors, *Fuzzy Logic in Artificial Intelligence (FLAI93)*, pages 35–46. Springer–Verlag, 1993.

[48] S. Nepal, M.V. Ramakrishna, and J.A. Thom. A Fuzzy Object Query Language (FOQL) for Image Databases. In *6th International Conference on Database Systems for Advanced Applications (DASFAA '99)*, page 117, April 1999.

[49] P. S. Yu and M.-S. Chen and J.-S. Park. An Effective Hash Based Algorithm for Mining Association Rules. In *Proc. of the 1995 ACM SIGMOD Conference, San Jose, California, USA*, volume 24, pages 175–186, June 1995.

[50] S. K. Pal and S. Mitra. *Neuro-Fuzzy Pattern Recognition: Methods in Soft Computing*. John Wiley and Sons Inc, New York, 1999.

[51] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, CA, USA, 1993.

[52] R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules. In J. B. Bocca, M. Jarke, and C. Zaniolo, editors, *Proc. of the International Conference on Very Large Databases, Santiago, Chile*, pages 487–499, September 1994.

[53] R. Agrawal and T. Imielinski and A. Swami. Mining Association Rules between Sets of Items in Large Databases. In *Proc. SIGMOD '93. 1993 ACM SIGMOD. International Conference on Management of Data*, volume 22, pages 207–216, June 1993.

[54] R. Srikant and R. Agrawal. Mining Quantitative Association Rules in Large Relational Tables. In *Proc. of the ACM SIGMOD Conference on Management of Data (SIGMOD'96), Montreal, Canada*, volume 25. ACM, June 1996.

[55] R. Sasisekharan, V. Seshadri, and S. M. Weiss. Data Mining and Forecasting in Large-Scale Telecommunication Networks . *IEEE - Intelligent Systems*, 11(1):37–43, February 1996.

[56] A. Silberschatz, H. F. Korth, and S. Sudarshan. *Database System Concepts, Fourth Edition*. McGraw-Hill, July 2001.

[57] M. Steinbach, G. Karypis, and V. Kumar. A Comparison of Document Clustering Techniques. In *KDD Workshop on Text Mining*, 2000.

[58] Y. Takahashi. Fuzzy Database Query Languages and Their Relational Completeness Theorem . *IEEE - Knowledge and Data Engineering*, 5(1):122–125, February 1993.

[59] H. Toivonen. Sampling Large Databases for Association Rules. In T. M. Vijayaraman, Alejandro P. Buchmann, C. Mohan, and Nandlal L. Sarda, editors, *Proc. of the 1996 International Conference Very Large Data Bases*, pages 134–145. Morgan Kaufman, September 1996.

[60] R. Tsaur, H. Wang, and J. Yang. Fuzzy Regression for Seasonal Time Series Analysis. *International Journal of Information Technology and Decision making*, 1(1):165–175, 2002.

[61] V. Ganti and R. Ramakrishnan and J. Gehrke. RainForest: A Framework for Fast Decision Tree Construction of Large Datasets. In A. Gupta, O. Shmueli, and J. Widom, editors, *Proc. of the 24th Annual International Conference on Very Large Data Bases, New York*, pages 416–427. Morgan Kaufmann Publishers Inc, August 1998.

[62] W. Hsu and B. Liu and Y. Ma. Integrating Classification and Association Rule Mining. In R. Agrawal and P. Stolorz, editors, *Proc. of the Fourth International Conference on Knowledge Discovery and Data Mining KDD-98, New York, USA*, pages 80–86. AAAI Press, August 1998.

[63] Y. Yin and J. Pei and J. Han. Mining Frequent Patterns without Candidate Generation. In *Proc. 2000 ACM-SIGMOD International Conference Management of Data (SIGMOD'00), Dallas, Texas*, volume 29, pages 1–12. ACM, May 2000.

[64] Y. Yoshinari, W. Pedrycz, and Hirota K. Construction of Fuzzy Models through Clustering Techniques. *Fuzzy Sets and Systems*, 54(2):157–165, March 1993.

[65] L. A. Zadeh. Fuzzy Sets. *Information and Control*, 8(3):338–353, 1965.

[66] L. A. Zadeh. The Concept of a Linguistic Variable and its Application to Approximate Reasoning. *Information Science*, 8(4):301–357, 1975.

[67] L. A. Zadeh. Fuzzy Sets as a Basis for a Theory of Possibility. *Fuzzy Sets and Systems*, 1(1):3–28, 1978.

[68] L. A. Zadeh. A Theory of Approximate Reasoning. *Machine Intelligence*, 9:149–194, 1979.

[69] M. J. Zaki, M. Ogihara, S. Parthasarathy, and W. Li. Parallel Data Mining for Association Rules on Shared-Memory Multiprocessors. In *Proc. Supercomputing'96*, pages 17–22, November 1996.

[70] M. J. Zaki, S. Parthasarathy, and W. Li. A Localized Algorithm for Parallel Association Mining. In *Proc. of the ACM Symposium on Parallel Algorithms and Architectures*, pages 321–330. ACM, June 1997.

[71] H. Zantema. Decision Trees: Equivalence and Propositional Operations. In *Proc. 10th Netherlands/Belgium Conference on Artificial Intelligence (NAIC'98)*, pages 157 – 166. H. L. Poutr'e and J. van den Herik, November 1998.