A Multi-Tiered Genetic Algorithm for Data Mining and Hypothesis Refinement

BY

Christopher M. Taylor

Submitted to the graduate degree program in Computer Science and the Graduate Faculty of the University of Kansas in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

_____
Arvin Agah, Chairperson

_____
Jerzy Grzymala-Busse

_____
Nancy Kinnersley

_____
Xue-Wen Chen

_____
Elizabeth Friis

Date Defended _____

The Dissertation Committee for Christopher M. Taylor certifies
that this is the approved version of the following dissertation:

A Multi-Tiered Genetic Algorithm for Data Mining and Hypothesis Refinement

Committee:

_____
Arvin Agah, Chairperson

_____
Jerzy Grzymala-Busse

_____
Nancy Kinnersley

_____
Xue-Wen Chen

_____
Elizabeth Friis

Date approved _____

Acknowledgements

First of all, I would like to thank my mentor and advisor, Dr. Arvin Agah. Without his patience and determination, I am not sure that I could have finished. He deserves the credit for any of my academic success. All failures, of course, are my own.

Thank you to all of my committee members: Dr. Jerzy Grzymala-Busse, for introducing me to data mining; Dr. Xue-Wen Chen, for introducing me to genetic algorithms and bioinformatics; Dr. Elizabeth Friis, for being an inspiration and giving me big ideas for the future; and Dr. Nancy Kinnersley, for believing in me and for stepping in at the last minute to bail me out.

A special thanks to Al and Lila Self for being so incredibly generous and establishing a Fellowship that has provided me with not only the monetary resources to complete this project, but has also helped me to bloom as a leader. I also owe a great debt of gratitude to the staff of the Madison and Lila Self Graduate Fellowship – Cathy Dwigans, Sharon Graham, Patty Dannenberg, Howard Mossberg, and Jim Morrison. I will spend the rest of my life trying to live up to all that they have given me.

Lastly, but most importantly, I owe a lifetime of thanks to my wife, Kim. Without her loving support, patience, and the occasional kick in the pants she has given me, I know that I would not have made it to this point. She left behind a life she loved to let me pursue this dream. I hope that now I can use my dream to make hers come true.

Abstract

While there are many approaches to data mining, it seems that there is a hole in the ability to make use of the advantages of multiple techniques. There are many methods that use rigid heuristics and guidelines in constructing rules for data, and are thus limited in their ability to describe patterns. Genetic algorithms provide a more flexible approach, and yet the genetic algorithms that have been employed don't capitalize on the fact that data models have two levels: individual rules and the overall data model. This dissertation introduces a multi-tiered genetic algorithm capable of evolving individual rules and the data model at the same time. The multi-tiered genetic algorithm also provides a means for taking advantage of the strengths of the more rigid methods by using their output as input to the genetic algorithm.

Most genetic algorithms use a single "roulette wheel" approach. As such, they are only able to select either good data models or good rules, but are incapable of selecting for both simultaneously. With the additional roulette wheel of the multi-tiered genetic algorithm, the fitness of both rules and data models can be evaluated, enabling the algorithm to select good rules from good data models. This also more closely emulates how genes are passed from parents to children in actual biology. Consequently, this technique strengthens the "genetics" of genetic algorithms. For ease of discussion, the multi-tiered genetic algorithm has been named "Arcanum."

This technique was tested on thirteen data sets obtained from The University of California Irvine Knowledge Discovery in Databases Archive. Results for these same data sets were gathered for GAssist, another genetic algorithm designed for data

mining, and J4.8, the WEKA implementation of C4.5. While both of the other techniques outperformed Arcanum overall, it was able to provide comparable or better results for 5 of the 13 data sets, indicating that the algorithm can be used for data mining, although it needs improvement.

The second stage of testing was on the ability to take results from a previous algorithm and perform refinement on the data model. Initially, Arcanum was used to refine its own data models. Of the six data models used for hypothesis refinement, Arcanum was able to improve upon 3 of them. Next, results from the LEM2 algorithm were used as input to Arcanum. Of the three data models used from LEM2, Arcanum was able to improve upon all three data models by sacrificing accuracy in order to improve coverage, resulting in a better data model overall. The last phase of hypothesis refinement was performed upon C4.5. It required several attempts, each using different parameters, but Arcanum was finally able to make a slight improvement to the C4.5 data model.

From the experimental results, Arcanum was shown to yield results comparable to GAssist and C4.5 on some of the data sets. It was also able to take data models from three different techniques and improve upon them. While there is certainly room for improvement of the multi-tiered genetic algorithm described in this dissertation, the experimental evidence supports the claims that it can perform both data mining and hypothesis refinement of data models from other data mining techniques.

Table of Contents

Chapter 1. Introduction

As more and more information is added to already immense databases, it is becoming increasingly difficult to analyze information. At the same time, it is becoming increasingly important that the information be analyzed to discover the patterns and truths that lie hidden within our vast collections of data. After all, what good is it to have all the information in the world but not know what it means, or what it might indicate?

Data are collected within some domain, meaning that all the information within that collection is related in some way. The data collected at a hospital are related in that they pertain to patients and the health and treatment of those patients. Data collected at a bank are related in that they pertain to the finances and spending habits of the bank's customers. Regardless of the domain or purpose of the data, within all collections of related data, patterns exist. These patterns can be used to discover meaning about the data and what they represent. For example, analysis of the medical records of a single hospital can reveal the types of illnesses common to the area the hospital serves. It could further reveal the types of treatment that were successful compared to the types of treatments that were not as effective.

This type of analysis, called data mining, uses computer science techniques to search for patterns within data. With the vast amounts of information that are contained in the databases of the world, data mining has the potential to reveal patterns regarding human health, the human genome, economics, consumer spending,

marketing demographics, and a host of other areas. Given this potential, and the untapped benefits that have yet to be fully realized, there is a tremendous need for more advanced data mining techniques that can find even the most obscure patterns.

1.1 Motivation

Many different data mining packages exist, taking many different approaches to finding patterns. Some use a detailed algorithm for finding patterns. Others use an approach that simulates evolution, so that solutions "evolve" from random processes. This approach is known as evolutionary programming, or a genetic algorithm.

In each case, the use of sets is critical to the analysis of the data. Patterns are found by using various combinations of sets to describe the patterns in the data. However, most set operations are omitted from these approaches, making complex statements about patterns difficult, if not impossible. Furthermore, a translation or encoding of the sets is often performed. Thus some fundamental pieces of the problem are separated from the solution. The language of sets, which is used to describe the patterns, is often extremely limited by the chosen data mining implementation. Furthermore, the means used to represent the information can limit the pattern-finding ability of the chosen technique.

While genetic algorithms have been shown to be very successful, they only partially mimic evolution. They use a "roulette wheel" combined with a fitness function to simulate natural selection, with the more fit members of the population having a higher probability of selection. Thus, the more fit members of the

population are more likely to be selected to produce offspring. In real-world reproduction, only half of the genes from each parent are passed to the child. These may or may not be the good genes that make each individual more fit for survival. Thus, the child may not necessarily be as fit as the parents. However, over many generations the inferior genes will occur in smaller percentages of the population and could potentially be eliminated completely. Few genetic algorithms address a means for determining which "genetic material" from each parent is to be passed to children.

The amount of data used in real world data mining is significant. Having a technique capable of searching through all possible patterns and selecting those that are the most descriptive can be a very time-consuming process. This is why a genetic algorithm approach was selected for this project. Genetic algorithms are capable of quickly searching large solution spaces. They are also very useful for finding solutions to problems that are difficult to find using more direct approaches. The strengths of genetic algorithms make them a good choice for data mining applications.

The motivation for this research is to develop a new tool for data mining that uses the sets as part of the solution process and takes advantage of set properties and operations. This is incorporated into a multi-tiered genetic algorithm, which uses multiple roulette wheels to more closely approximate biological reproduction in which children receive genes from each parent.

1.2 Problem Statement

This dissertation details a novel data mining tool that combines set objects with a multi-tiered genetic algorithm. By performing direct manipulation of sets, the encoding process used in genetic algorithms can be eliminated. The sets can be directly used, manipulated, mutated, combined, etc. until a solution is reached. A rule can be achieved by combining sets together into a clause, separated by set operators (i.e. AND, OR, NOT, XOR, DIFFERENCE). Thus, a rule could look like: A AND B OR C AND NOT D $\rightarrow$ Z. This collection of sets and operators can be resolved to a single set, which is a subset of Z. Because patterns within a set can also be defined as a subset of that set, this process can be used to construct rules that describe patterns within sets. Furthermore, the rules themselves can also be collected into a set. The resulting solution is thus a set of rules, each of which describes a pattern within the data. This collection of rules is hereafter referred to as a "description." Each generation of the genetic algorithm is comprised of multiple descriptions, with each description representing one member of the population.

By building code for generic set objects, the properties and advantages of sets can be used over and over regardless of the hierarchy – a set is the same as a set of sets. New sets can easily be created and manipulated by whatever algorithm is employed, and the need for multiple alternative data structures (such as arrays of bit streams to represent the population of a genetic algorithm) can be reduced. This approach more closely mimics biology in that the "organisms" used in the genetic algorithm are "multi-cellular." The smallest sets are combined together to represent a

pattern in the data (a rule). These are then gathered into an even larger set to describe the data as a whole (a description). In organisms, cells combine to form tissues, which combine to form organs, which combine to form the organism. Thus, the inspiration for the use of set theory in this research comes from biology.

The genetic algorithm described in this dissertation introduces a second "roulette wheel" to the natural selection process. As in traditional GAs, the first roulette wheel is used to select the two parents from whom children will be created. The second roulette wheel is introduced at this point to increase the probability of selecting the more favorable genes from each parent to pass to the child. The "organism" can be thought of as the collection of rules. The "genes" are the individual rules themselves. The process begins with two descriptions, or collections of rules, being selected. Which parents and which genes are considered "favorable" is determined by a fitness function. Then the rules from each description are evaluated and selected as to which will be passed to the children, with higher probability going to those rules with higher fitness values.

1.3 Research Hypothesis

It is hypothesized that the approach presented in this dissertation will produce a robust data mining technique capable of finding obscure patterns within data. The technique is robust in that it can be used on data from a wide variety of domains without having to change any of the programming or encode rules in a different way. Obscure patterns are not observable by humans because the patterns are very complex

and the data sets are often very large. The addition of the second roulette wheel should help the process to converge more rapidly to a solution, while ensuring that the stronger rules are preserved. Furthermore, because the algorithm employs fuzzy logic, it is capable of finding rules that are not true all the time, but which are true *most* of the time.

More importantly, the approach described in this dissertation will be able to perform hypothesis refinement. It can potentially take a data model produced by any data mining technique and improve the results of that data model. The ability to perform hypothesis refinement could be a significant step in the field of data mining. It would provide a tool for improving upon data models, regardless of the original method used to obtain the model, by utilizing the strong search capabilities of genetic algorithms.

1.4 Dissertation Organization

The details of the design, experiments, and results are discussed in the chapters and sections that follow. Chapter 2 explains some of the background required for the development of the multi-tiered genetic algorithm, such as set theory, data mining, and fitness metrics. It also includes some previous work that has been done in applying genetic algorithms to data mining. Lastly, it shows how the multi-tiered approach is different from previous approaches to data mining using genetic algorithms.

Chapter 3 describes the methodology employed in the design and development of the multi-tiered genetic algorithm. The structure of the Arcanum multi-tiered genetic algorithm is detailed in this chapter. It includes the design requirements and the actual design used to meet those requirements. The chapter details how input data is used, how uncertainty within the data is handled, as well as how numeric attributes are handled.

Chapter 4 discusses the evaluation of the Arcanum multi-tiered genetic algorithm. It describes how the data sets were chosen, how training and testing data were separated, and the types of metrics used for evaluating the results.

Chapter 5 includes all of the results from the experiments. The chapter also contains descriptions of the actual data sets chosen for evaluation. It shows how Arcanum compared to two other important data mining techniques. It also reports on the results of hypothesis refinement. Discussion of each of these sets of results is also included in this chapter.

Chapter 6 concludes the dissertation. It lists the theoretical contributions of the work contained in this dissertation. The chapter also includes numerous observations made during the course of this research. The limitations of the research are discussed in this chapter. The implications of some of the observations, as well as the limitations of the research, suggest some interesting future work, which is proposed in this chapter.

Chapter 2. Background and Related Work

2.1 Set Theory

In order to begin the discussion of the proposed project, several key concepts must first be defined. The most fundamental concept for this project is that of Set Theory. Set Theory can be defined as the mathematical science of the infinite (Jech, 2002). It studies the properties of sets, which are then used to formalize all the concepts of mathematics. A full discussion of Set Theory is beyond the scope of this dissertation, so for brevity only a few related key concepts are included here. The reader unfamiliar with set theory notation and terminology is referred to Kenneth Rosen's book *Discrete Mathematics and its Applications* (Rosen, 2007).

2.1.1 Crisp Sets

When the membership of sets is clearly defined, they are often called "crisp sets." This is because the size of the sets, the number of elements, and the identity of the individual elements are all very well defined. An example of a crisp set could be the number of books on a specific bookshelf. Membership in the set is easily determined – if a book is on the bookshelf, then it is in the set; books not on the shelf are not in the set. The number of elements in the set can be easily calculated by counting the books on the shelf. Questions about membership are also easily answered (*i.e.* Does the set contain "War and Peace?" or "What are the titles of all

books in the set?"). Unfortunately, there are many real-world situations which are difficult to define in terms of crisp sets.

2.1.2 Rough Sets

Because crisp sets are sometimes difficult to represent precisely, they are sometimes defined using two "rough sets." A rough set approximates the upper and lower bounds of a set that is otherwise difficult to define precisely. The upper and lower bounds are sets themselves. Suppose X is a set that is difficult to define as a crisp set. Let set A be the lower bound for set X, so that all of the elements of set A are certain to belong to set X. Let set B be the upper bound for set X, so that set B contains all of the elements that could possibly belong to set X. The definition for set A would be such that all elements of A are definitely elements of set X. In other words, $A \subseteq X$. Set B is given a much broader definition so that it contains at least some of the elements in X so that $B \cap X \neq \varnothing$ (Pawlak, 1994).

Rough sets are extremely useful when dealing with uncertainty or ambiguity. For example, if the definition of a particular set is unclear, it can be difficult to decide if some cases are elements of the set or not. By creating two rough sets, it is possible to create upper and lower bounds. As a more practical example, consider the concept of a "good student." The lower approximation would include all students for which everyone would agree they were good students. The upper approximation would include those students for which some might be considered good by some people, but not everyone would call them good students. Somewhere between these two sets lies

the actual set of good students, but because it is such a vague concept, it is easier to approximate it with rough sets than to determine a precise definition for the set.

2.1.3 Fuzzy Sets

Sometimes there are elements that could potentially belong to several sets and so the definition of membership becomes a little blurry, which is where "fuzzy sets" are useful. Fuzzy sets allow for an individual element to be assigned a probability of belonging to a set, often with a different probability for several different sets, a concept that cannot be handled with crisp sets. Each element in a fuzzy set is a given a membership value indicating the probability that the element belongs in the set (Straccia, 1998). For example, let A be the set of weekdays and B be the set of weekend days. To which set does Friday belong? In crisp sets, Friday is a weekday because the weekend only consists of Saturday and Sunday. However, to many people Friday is part of the weekend. Using fuzzy sets, Friday could be assigned to set A with a 60% probability of membership and to set B with 40% probability. Thus, fuzzy sets allow for "loose" membership and can even be used to make claims such as element *x* is more likely to be a member of the set than element *y*.

2.2 Data Mining

Data mining is the process of finding patterns that lie within large collections of data. Contrary to more traditional data analysis methods, which begin with a hypothesis and then test the hypothesis based upon the data, data mining approaches the problem from the opposite direction. Thus, data mining is discovery-driven rather

than assumption-driven (Radivojevic *et al.*, 2003). As the process searches through the data, patterns are automatically extracted.

In general, data mining objectives can be placed into two categories: *descriptive* and *predictive* (De Raedt *et al.*, 2001). The goal of descriptive data mining is to find general patterns or properties of elements in data sets. This often involves aggregate functions such as mean, variance, count, sum, etc. In other words, descriptive data mining reports patterns about the data itself. Predictive data mining, however, attempts to infer meaning from the data in order to create a model that can be used to predict future data. This is often done by grouping data elements based on similarities and then analyzing the properties those data elements have in common. The common properties should be a reasonable predictor for the given result.

Another important concept is the difference between supervised and unsupervised learning. Supervised learning takes place when the data has been pre-classified. In other words, the items in the data have already been placed into groups or been assigned some value or result. For example, in a database about house values, each item in the database will contain values such as to the number of bedrooms, square footage, etc. In supervised learning, each house will also be assigned a monetary value. The goal of the data mining process is then to find the patterns that result in a given value. Unsupervised learning, on the other hand, occurs when the data is not pre-classified. In these cases, the data mining process cannot make value judgments. It can find correlations within the data, but it is not able to make any inferences about what those patterns might mean. Thus, unsupervised

learning is descriptive while supervised learning is predictive. In order to make predictions, the data must be classified, or given value, by some outside source.

Some of the difficulties involved in data mining are problems with the data. When collecting data from different sources, the various sources often have different formats for the data, collect different types of data, and have different protocols about the data. For example, one set of records might contain a person's age, while a similar set of records from another source might not. Further compounding the problem, even within the same source, data can be erroneous or even missing. Perhaps the ages for some, but not all, of the people are recorded in the data. This can make finding patterns in the data very difficult when the data is incomplete or inconsistent. Thus, a major task in data mining is in how to handle these anomalies in the data that are not actually *part* of the data.

There are so many different approaches to data mining that it is difficult, if not impossible, to list them all. And the different techniques vary as widely as the data they are used to analyze. There are data mining techniques implementing neural networks, clustering algorithms, regression modeling, genetic algorithms, data visualization, and many more different approaches. Rather than describe the various approaches, this dissertation will focus on the techniques directly relevant to the proposed project.

2.2.1 Decision Tables

Decision tables are very similar to tables in a database. Each line in the table is called a "tuple" and represents one specific item such a house, an employee, a

business, a car, etc. Each column in the data is an "attribute" and is used to describe each tuple in the table. What distinguishes a decision table from a database table is that the decision table has some descriptive class or category associated with each tuple. The attributes can be thought of as conditions, with the decision being associated with those conditions. Table 2.1 is an example of a decision table indicating the conditions for which a patient will receive different percentages of reimbursement from his or her health insurance provider.

There are 8 tuples in Table 2.1, each representing a different condition under which a client might apply for a reimbursement from the insurance company. Each tuple can take a different value for each attribute. As seen in the table, there are three types of visits: office, hospital, and lab. The other two attributes only have two possible values: yes or no. Thus, each tuple can be described by using the values of the attributes. The "decision" in this table is the "reimbursement" attribute. The combination of an attribute with a value is called a "feature" (Kohavi and Provost, 1998). For example, (type_of_visit, office) is a feature – it specifies that the type of visit was an office visit.

Tuple #	*Deductible Met*	*Type of Visit*	*Participating Physician*	*Reimbursement*
1	yes	office	yes	90
2	yes	office	no	50
3	yes	hospital	no	80
4	yes	Lab	no	70
5	no	office	yes	0
6	no	office	no	0
7	no	hospital	no	0
8	no	Lab	no	0

Table 2.1. Decision table indicating conditions for reimbursement from insurance.

Some of the combinations of attributes in Table 2.1 are missing, such as a hospital visit that is also classified as a participating physician. To cover all possible combinations of features in the first three attributes of the table, only 12 tuples would be required. Real world data, however, can have millions or even billions of tuples.

2.2.2 Rule Induction

Rule induction is the process of taking the data and searching for meaningful patterns that can be described in terms of features. The result is a set of rules that describe the patterns in the data. Each rule consists of two parts: the *antecedent* and the *consequent* (Siler, 2005). The antecedent, or left-hand side, of the rule is the condition that must be met for the rule to be applicable. It is the "if" part of the rule. The consequent, or right-hand side, of the rule is the action or decision that follows if the antecedent is true. If the antecedent is true, then the consequent follows. A sample rule from Table 2.1 is:

(deductible_met, no) $\rightarrow$ (reimbursement, 0%)

This rule can be read as "if the deductible is not met then there is 0% reimbursement." If the antecedent, (deductible_met, no) is true then the consequent, (reimbursement, 0%) is true. This does not hold when reversed, however – the antecedent does not follow from the consequent.

The example rule describes tuples 5, 6, 7, and 8 as to how those conditions are reimbursed. It does not cover tuples 1 through 4 because they have (deductible_met,

yes). More rules need to be induced to cover the first four tuples. Thus, by creating a set of rules, all of the patterns in the decision table can be described.

2.2.3 Rules with Set Operators

A consequent can be described by a single set in the antecedent, but this occurs very rarely in real-world data. It usually requires the intersection of multiple sets to provide an adequate classification that does not include members from other consequents. These sets have to be separated by set operators. The set operators describe how the sets relate to one another. Without set operators, the meaning of the rule is ambiguous. For example:

(deductible_met, yes) (participating_physician, yes) $\rightarrow$ (reimbursement, 90%)

does not have a set operator. From Table 2.1, it is apparent that for the consequent to be true, a tuple must be a member of both sets in the antecedent, requiring that an AND operator be placed between the two sets. If an OR operator had been used then tuples 1 through 5 would be members of the antecedent. Tuples 1 through 4 are members of (deductible_met, yes) and tuple 5 is a member of (participating_physician, yes). Regardless of the antecedent, only tuple 1 would be a member of the consequent. The AND operator makes the rule correct. The OR operator would make the rule correct only one-fifth of the time. However, there are times when the OR operator is more appropriate:

(age, 15) OR (age, 17) $\rightarrow$ (age_group, teen).

Changing this OR to an AND would mean that there must be a tuple that has ages of both 15 and 17 in order to be a teen.

The four set operators used by Arcanum are:

Y AND Z – must be a member of both sets Y and Z.

Y OR Z– is a member of at least one of the sets Y and Z.

Y XOR Z – is a member of only one of the sets Y and Z.

NOT Z – is not a member of Z.

Using sets and set operators, complex statements can be constructed about the data. This is necessary because the patterns within the data rarely allow for a single set to be used as the antecedent for a rule. Statements can be constructed with sets and set operators that allow rules to make unique classifications.

2.3 Genetic Algorithms

Genetic algorithms, or GAs, are based upon evolutionary principles of natural selection, mutation, and survival of the fittest (Dulay, 2005). GAs are very different from most computer programs, which have well-defined algorithms for coming up with solutions to problems. The genetic algorithm approach is to generate a large number of potential solutions in a search space and "evolve" a solution to the problem.

2.3.1 Genetic Algorithm Techniques

One of the big keys to a successful genetic algorithm is in the development of a good "fitness function." The fitness function is how each potential solution is

evaluated by the algorithm, and is in essence how the problem to be solved by the algorithm is defined. For example, if the purpose of the genetic algorithm is to design a car, then the fitness function will provide a means for evaluating the "fitness" of any car design.

When developing a genetic algorithm, one must decide how each solution will be represented in the algorithm. For simplicity, a string of bits is most often used. The bits can be used to represent any part of the solution. Taking the car example, some of the bits might represent the color of the car, others the size of the wheels, and others the gas mileage of the car. It is the responsibility of the fitness function to understand what the bits mean and how to use them to evaluate the fitness of each potential solution.

During the first iteration of a GA, it generates an initial "population" of potential solutions, which could be random. Each member of the population is then examined and its fitness is evaluated and recorded. Once each member of the population has been evaluated, then the next generation is produced from the current generation. There are many ways of generating the next generation, but the two most popular techniques involve "crossover" and "mutation." In crossover, two members of the population are chosen at random with higher probability given to the more fit members of the population. These two members are then combined to produce two offspring. This is usually performed by selecting a position in the bit sequence and exchanging the two sequences after that position, which is called the crossover point.

For example, given the following two members of a population:

11101010

10010100

If these were crossed over at position 4, the resulting offspring would be:

11100100

10011010

There are many ways to perform the crossover, but this example is one of the simplest. The result is two new members in the next generation. In theory, these two new members should be reasonably more fit because they likely came from fit members in the previous population. Each member of the population is assigned a biased probability of selection. Because of this increased probability of selection, the most fit members of a population are more likely to be selected for crossover than less fit members. However, there is always a possibility that a less fit member will be selected instead. This technique can be thought of as a weighted roulette wheel, where each member of the population is assigned a space on the wheel, and the size of its space is determined by its fitness function. The more fit a member of the population is the larger the space it has on the roulette wheel, resulting in a higher probability of it being selected. Selection occurs by "spinning" the roulette wheel.

Usually, the crossover process continues until the size of the next generation is the same as the population size of the previous generation. After crossover has taken place, "mutation" is then applied to each member of the population. A typical mutation function is to assign a probability for flipping each bit. Thus, if a 10% value

for mutation is assigned, then each bit of each member of the population has a 10% chance of being flipped -- a 0 becomes a 1, or a 1 becomes a 0. After mutation is completed, each member of the new generation is evaluated for fitness. The previous generation dies and is replaced by the new generation. The process repeats and another generation is produced. This process of evolving new populations continues until some criteria are met. The stopping criteria could be (a) when the overall fitness of the population reaches a certain value, (b) when the overall fitness over several generations fails to change more than a specified value, or (c) after a certain number of generations have been produced.

The details in how genetic algorithms are actually implemented can vary widely. For example, how crossover is handled or how mutation is performed can vary depending on the problem they are designed to solve. Regardless of the details, the overall process remains the same: generate an initial population, evaluate the members of the population, generate a new population based upon the more fit members of the previous generation, and repeat the process until a certain stop criterion is achieved.

2.3.2 Applications of Genetic Algorithms

Genetic algorithms are very powerful search tools. By "search" it is meant that GAs are capable of pouring through a large number of potential solutions to find good solutions. Scheduling has been an area where genetic algorithms have proven very useful. The GA searches the space of potential schedules and finds those schedules which are most effective and maximize the desired criteria (such as

minimizing idle time). For example, GAs are used by some airlines to schedule their flights (Dulay, 2005). An application of a GA to a financial problem (tactical asset allocation and international equity strategies) resulted in an 82% improvement in portfolio value over a passive benchmark model, and a 48% improvement over a non-GA model used to improve the passive benchmark (Dulay, 2005). GAs have also been applied to problems such as protein motif discovery through multiple sequence alignment (Mendez, *et al.*), and obtaining neural network topologies (Taylor and Agah, 2006).

More information on genetic algorithms can be found in Goldberg (1989).

2.4 Confusion Matrices and Fitness Metrics

When discussing the results of a data mining model, two common measures are *sensitivity* and *specificity*. Sensitivity, often called the "true positive rate," measures the percentage correctly identified as positive out of the total number of positives. Specificity, often called the "true negative rate," measures the percentage correctly identified as negative out of the total number of negatives.

A helpful way to discuss sensitivity and specificity is to use a "confusion matrix" (Hamilton, 2005). A confusion matrix is an L × L matrix, where L is the number of different label values. Table 2.2 is a 2 × 2 confusion matrix. In the confusion matrix, "a" denotes those tuples which were predicted as negative and were actually negative. Quadrant "b" is composed of those tuples which were predicted as

positive but were actually negative, also known as “false positives.” Quadrant “c” are those tuples which were classified as negative but were actually positive, also known

		Predicted	
		Negative	Positive
Actual	Negative	a	B
	Positive	c	D

Table 2.2. A 2 × 2 confusion matrix.

as “false negatives.” Quadrant “d” contains the tuples that were classified as positive and were actually positive (Kohavi and Provost, 1998). Thus, using the confusion matrix, it is easier to define some fitness metrics:

$$\text{Accuracy} = (a+d)/(a+b+c+d)$$

$$\text{Sensitivity (True Positive Rate)} = d/(c+d)$$

$$\text{Specificity (True Negative Rate)} = a/(a+b)$$

$$\text{Precision} = d/(b+d)$$

$$\text{False Positive Rate} = b/(a+b) = 1 - \text{Specificity}$$

$$\text{False Negative Rate} = c/(c+d) = 1 - \text{Sensitivity}$$

- Accuracy is the percentage of tuples that are correctly classified out of all the tuples which are given a classification.
- Sensitivity, sometimes called *recall*, is the percentage of positive classification.
- Specificity is the percentage of negative classification.

- Precision indicates the number of exceptions to a rule. For example, a precision of 4/5 indicates that there is 1 exception to the rule.
- False Positive Rate is the percentage of tuples which are classified as positive but in reality are negative.
- False Negative Rate is the percentage of tuples which are classified as negative but in reality are positive.

There is one more metric which is useful when discussing the fitness of a data mining model, and that is *coverage*. The coverage of a model is the proportion of the data for which there is a rule. Thus, a model which has 90% coverage provides rules which classify 90% of the tuples. Coverage only means that a classification is made. It does not measure the accuracy of the classification.

Using these seven metrics, the results of a data mining model can be evaluated. This allows for reasonable comparisons to be made between different models.

2.5 Genetic Algorithms in Data Mining

There are currently two different approaches to rule discovery using genetic algorithms (Au *et al.*, 2003): the Michigan approach and the Pittsburg approach. The Michigan approach, first introduced by Holland (1986), represents a rule set by the entire population, with each member of the population representing a single rule. The Pittsburg approach, exemplified by LS-1 (Smith, 1983), represents an entire rule set

with a single chromosome. Thus each member of the population represents an entire set of potential rules for describing the data.

Data sets are either single-class or multi-class. This refers to the number of possible values for the decision class. If there is only one decision value, all of the rules will describe that value – thus it is a single-class set. In a multi-class set, there are multiple decision values in the data (but each rule describes a single value). Table 2.1 is a multi-class set because there are multiple values for the "reimbursement" decision. The Michigan method is very useful for multi-class problems, but it suffers in that there is no way to ensure a high coverage of the data by evaluating a single rule at a time. The Pittsburg approach has been used to learn rules for a single class. To induce rules for multiple classes, the algorithm needs to be run multiple times.

In their experiments, Au *et al.* (2003), used the Pittsburg approach to great success. They compared the results of their data mining model, DMEL, developed by a genetic algorithm, to the results developed by C4.5, a very popular and well-known data mining algorithm that uses decision trees (Quinlan, 1993). The experiment included seven different data sets that were diverse in nature. In each instance, the genetic algorithm produced more accurate results than C4.5, ranging from as little as 0.3% up to a 13% improvement in accuracy (Au *et al.*, 2003).

In their work, Flockhart and Radcliffe (2005) concluded that genetic algorithms are well suited to undirected data mining, but can also be used for directed data mining. Undirected data mining is the most common form, where the program simply looks for patterns and describes them. In directed data mining, the user

specifies the type of information in which they are interested. Using the Michigan approach, Flockhart and Radcliffe's GA-MINER was able to find interesting, non-trivial rules within the data sets used for the experiment. Flockhart and Radcliffe also pose an idea for "hypothesis refinement" in which the user can "seed" the genetic algorithm with a rule or set of rules which the GA can use as an initial population. In this way, the GA can refine the initial hypotheses to produce a better model for the data.

Chapter 3. Research Methodology

3.1 Design Overview

This project is an effort to provide a technique for data mining that essentially combines the Michigan and Pittsburg approaches (Holland, 1986 and Smith, 1983, respectively), thus performing both methods at the same time. The technique proposed here uses a combination of set theory and genetic algorithms. The desired outcome is a technique that can generate a set of rules to describe any data set, without using any of the conventional data mining techniques. For ease of discussion, the approach has been dubbed "Arcanum" (Latin for "secret" or "mystery").

Similar to the Pittsburg approach, each member of the population in Arcanum represents an entire set of rules that describes the data. Thus, a population of 50 would contain 50 different sets of rules for describing the data. Similar to the Michigan approach, Arcanum is able to find patterns for each decision class at the same time. In this manner, Arcanum combines both techniques to find a set of rules that describes every decision class in the data with only a single run. A set of rules is referred to as a "description" through the rest of the dissertation.

3.2 Creation of Sets from Input Data

Arcanum uses a decision table as input. Each unique feature in the decision table is used to create a set. Thus (deductible_met, yes) and (deductible_met, no) are two of the sets that would be created from Table 2.1. Arcanum also allows multiple

columns of the decision table to have the same attribute name. This is useful when multiple features exist for each tuple with respect to a particular attribute. For example, a single object can have several different colors. Creating a separate attribute for each color (*i.e.* Color1, Color2, Color3, etc.) limits the ability to find patterns in the data because matches only occur within the same attribute. In other words, (color1, black) would not match (color3, black) because they are from two different attributes, even though the values of the attributes are the same. However, by having three columns in the table called "color," each tuple can have multiple values for the attribute and Arcanum can still match the colors regardless of the order in which the colors appear in the table. By allowing for matching across multiple columns that have the same name, Arcanum can match an object that is red, white, and blue with another object that is blue, white, and red. In other words, having multiple columns with the same name allows for matching independent of the order in which the values occur.

Each feature set contains the tuples to which it applies. From Table 2.1, the feature set (deductible_met, yes) would contain tuples 1, 2, 3, and 4. Each unique tuple can then be described by using a logical AND of the appropriate sets. Thus, from Table 2.1 tuple 1 can be uniquely identified as:

> (deductible_met, yes) AND (type_of_visit, office) AND (participating_physician, yes).

Note that the last feature of tuple 1 is unnecessary because the first three features uniquely identify it.

Arcanum also creates a Universal Set. The Universal Set includes all the tuples. Arcanum ignores any other sets which are equal to the Universal Set because the knowledge gain from such sets is negligible and often obvious from a cursory examination of the data.

3.2.1 Numerical Attributes

When dealing with numerical attributes, the values rarely match exactly. For example, (temperature, 98.5) and (temperature, 98.6) would be separate sets even though they should probably be combined into a single set. In order to deal with this, Arcanum uses a binary discretization method to create partitions for numeric attributes. This discretization is performed locally on each attribute, meaning that the discretization process is performed on each numeric attribute independently of any other numeric attribute.

The discretization begins by creating a list of "break points" between features in the decision table. For Table 2.1, the only numeric attribute is the "reimbursement" attribute. The first break point occurs half way between the smallest value and the next largest value. The next break point occurs between the next two larger values and so on. In Table 2.1, the smallest value is 0 and the next largest is 50, thus the first break point is 25. The next break point occurs between 50 and 70, then between 70 and 80, and then between 80 and 90. Thus, the list of break points for Table 2.1 is 25, 60, 75, and 85.

Using the list of break points, intervals are then created. Two intervals are created for each break point. For Table 2.1 there will be eight intervals to partition

the reimbursement attribute because there are 4 break points. For any break point, the two intervals are from the smallest value to the break point and from the break point to the largest value. Using the break point of 25 from Table 2.1, the first interval is [0, 25] and the second interval is (25, 90]. These two intervals are then used to create sets. The set (reimbursement, [0, 25]) contains tuples 5, 6, 7, and 8. The set (reimbursement, (25, 90]) contains tuples 1, 2, 3, and 4. Each remaining break point is then used to partition the attribute and create sets.

Using this discretization process, Arcanum can then induce rules based on these intervals of values rather than trying to use each separate value.

3.2.2 Uncertainty

In real world data, there are often times when an attribute value for a tuple is not known, leading to "uncertainty" in the data. Arcanum allows two special symbols in a decision table to denote uncertainty. A "?" as an attribute value for a tuple indicates that there is no value for the attribute for the respective tuple and that the attribute value should be ignored for the tuple. This can be used in the case of multiple values for the same attribute, such as the "color" in previous examples. If there are three columns for color, but a particular tuple only has two colors, the "?" can be placed in one column to tell Arcanum to ignore that column for that tuple. The "?" can also be used to create more certainty in Arcanum. By ignoring unknown attribute values, only those values which are known are used to induce rules. Thus, the resulting rules are "certain" because they make no assumptions about the unknown values – they are simply ignored.

The “*” as an attribute value for a tuple also indicates uncertainty, but is handled differently than the “?” symbol. When the “*” is used, Arcanum adds the tuple to all sets involving that attribute. It is treated as if it contains every known value for the attribute. For example, if a ninth tuple were added to Table 2.1 and contained an “*” for the first attribute, then the tuple would be added to the sets (deductible_met, yes) and (deductible_met, no). This allows Arcanum to handle uncertainty by using possible values for the attribute when inducing rules, but means that the rules are less certain because assumptions have been made about some attribute values.

Both uncertainty symbols can be used within the same decision table, allowing the user to specify that rules involving particular attributes must be certain, but rules involving other attributes can be less certain. When there are no missing attribute values, neither symbol is necessary.

Using the concept of rough sets discussed in section 2.1.2, allowing for both methods of dealing with missing data, upper and lower bounds for the rules that describe the data can be constructed. The lower bound is created using the “?” operator, resulting only in rules that are derived from certain data. The upper bound is created using the “*” operator, resulting in rules that could be possible is the assumptions about the missing data are correct.

3.3 Modes of Operation

Arcanum can run in either the directed mode or the undirected mode (Flockhart and Radcliffe, 2005). In directed mode, the user specifies which attributes in the decision table are of interest. Only the attributes designated by the user will be used as consequents for rules. In other words, the rules induced by Arcanum will only describe those attributes. All other attributes can be used in the antecedent of rules, but not as consequents. This mode is useful when the user knows the type of information he or she wants to look for.

In the undirected mode, Arcanum will use all attributes as consequents of rules. Essentially, it will try to find patterns for every feature in the decision table. By default, Arcanum operates in the undirected mode, attempting to describe the data set as completely as possible.

3.4 Arcanum

After creating the sets from the input decision table, Arcanum employs a genetic algorithm to produce a description of the data. The outline of the process in Arcanum is as follows:

1. Seed the initial generation of descriptions
2. Evaluate rules
3. Evaluate descriptions
4. Create next generation
5. Manipulate rules

6. Repeat steps 2 through 5 until stop criterion is reached

3.4.1 Seeding the Initial Generation

The initial generation of descriptions can be seeded in one of two ways: Arcanum can randomly generate descriptions or the user can provide Arcanum with an initial description which Arcanum will then attempt to refine through the genetic algorithm.

By default, Arcanum will randomly generate an initial population of descriptions. For each description, Arcanum will generate a random rule for each feature of interest. Each of these initial rules will contain only one set in the antecedent. This will cover any possible one-to-one relationships between features, where a decision class can be described using only a single set in the antecedent of the rule.

If the user chooses to provide an initial set of rules, then Arcanum will use these rules to seed one-third of the initial population. Another third of the population will be seeded with randomly modified versions of these rules. The last third of the population will be composed of randomly generated descriptions. The decision to split the initial population into thirds was made in order to provide a diverse initial generation. The chances of finding beneficial crossovers should be increased while still allowing for some of the initial rules to survive. If the initial rules are “perfect” then they should preserved all the way to the final generation.

3.4.2 Evaluation of Rules

Usually, Sensitivity (true positives) is associated with Specificity (true negatives) when evaluating a classification method (Weiss and Kulikowski, 1991). The technique discussed here does not deal with negative classification, meaning that if a tuple does not match a rule, it cannot be concluded that the tuple is not part of the class. It can be concluded that the tuple is not part of the *subset* of the class described by the rule. A different rule might show that the tuple is part of the class, just in a different subset. In order to determine if a tuple does not belong to a class, it must be compared to all of the rules that describe that class. Because of this, Specificity has little meaning for individual rules. Precision has been chosen as a substitute because it provides information on the number of exceptions to a rule.

The Sensitivity, also known as Recall, and the Precision of each rule in each description is calculated according to the definitions shown in section 2.4. Precision identifies how inviolate the rule is, *i.e.* the frequency with which the rule is true. Recall is the proportion of the rule consequent, the decision class, which is classified by the rule. A Recall of 75% means that three-fourths of the rule's consequent are classified by the antecedent. The fitness value of the rule is calculated as follows:

$(weight_{precision} * precision) + (weight_{recall} * recall)$

The weights are converted to decimal values, so that 80% becomes 0.80. As such, the weights always add up to 1, meaning that each rule will have a fitness value ranging from 0 to 1. The weights are used so that preference can be given to models with high precision or high recall, depending on the needs of the user.

3.4.3 Evaluation of Descriptions

When evaluating a description, the purpose is to measure the collective results of the rules contained in the description. It is possible to have a description that contains very good rules that only classify a small portion of the data set and make no classification for a large portion of the data. It is also possible to have a description that classifies all of the data, but the classifications are inaccurate. To ensure that descriptions provide classifications for a large part of the data and that the classifications are correct, the Accuracy and Coverage metrics are used.

In this dissertation, the fitness of a description is defined as the weighted sum of its Accuracy and Coverage. The Accuracy of a description is the percentage of correct classifications (true positives and true negatives). An Accuracy of 25% means that only one-fourth of the classifications in the description are correct. The Coverage of a description is the percentage of the data set for which a classification is made, regardless of whether the classification is correct. A Coverage of 80% means that 20% of the data remains unclassified by the description. The fitness value of the description is calculated as follows:

$(weight_{precision} * accuracy) + (weight_{recall} * coverage)$

Thus, each description can have a fitness value ranging from 0 to 1. Note that this uses the same weights specified for Precision and Recall. This is because Accuracy reflects the Precision of the entire model, while Coverage reflects the Recall of the entire model.

3.4.4 Creating the Next Generation

Based upon the fitness values assigned to the descriptions and rules, a new generation of descriptions is created. This is done through a process of selecting two "parent" descriptions and combining rules from each of them to create two "child" descriptions. The children then become part of the new generation. Typically, this is done using a "roulette wheel" approach, like the one described in section 2.3.1. Arcanum uses *two* roulette wheels, one within the other, in order to create the children.

First, the fitness values of all descriptions are normalized, providing a selection probability for each member of the population. The higher the fitness value of a description, the more likely it is to be selected as a parent for the next generation. Two descriptions are then selected based upon these weighted values. These two descriptions will be used as parents.

Each child will receive one-half of its rules from each parent. This is done through another roulette wheel process similar to the one used to select the parents. Rules with higher fitness values are more likely to be selected to pass on to the child. A child cannot receive the same rule from the same parent more than once, however the same rule can potentially be received once from each parent, resulting in a duplicate. This is similar to the way genes are passed in biology.

Using this process, two children are created from the selected parents. After the children are created, two new parents are selected. This can result in the same parents being chosen. Because of the weighted roulette wheel process used in

selecting parents and rules, there is a chance of producing children in the next generation that have the exact same genes. The probability of generating children with the same genes is inversely proportional to the number of genes being received from each parent. In other words, the more genes each child receives from a parent, the less likely it is that their children will have the exact same genes. While the production of duplicate children could indicate convergence on a solution, it is not necessarily desirable – just because two descriptions have the same rules does not mean that those are the *best* rules. Some variance is still necessary to ensure that some possibilities are not being overlooked. This is handled in the next step, Rule Manipulation.

3.4.5 Rule Manipulation

This step simulates the role of mutation in evolution. Each rule of each description in the new generation has a chance of undergoing a random mutation. The probability for any given rule undergoing a mutation is equal to 1 – the fitness of the rule. Thus, the higher the fitness score of the rule, the less likely it is to mutate. Only one operator can be applied to a specific rule or description per generation. Thus, if one mutation is applied, then none of the others can be applied to that rule or description until the next generation. The possible operations are:

- ChangeOperator: randomly change one of the set operators to a different operator

- ChangeSet: randomly change one of the sets in the antecedent to a different set
- ComplicateRule: add AND with a random set to the end of the antecedent
    - Chance of selection is inversely proportional to Precision. The lower the Precision, the higher the chance that ComplicateRule will be applied to the rule. When Precision is low, it indicates that there are many exceptions to the rule – it is too general. Adding to the antecedent can help make the rule more specific and remove some of the exceptions, thus improving Precision.
- SimplifyRule: randomly select a position in the antecedent and delete everything in the antecedent that comes after the selected position
    - Chance of selection is inversely proportional to Recall. The lower the Recall, the higher the chance that SimplifyRule will be applied to the rule. When Recall is low, it indicates that the antecedent excludes many members of the consequent – it is too specific. Stripping off operators and sets from the end of the antecedent can make it more general and potentially include more members in the consequent that were previously being excluded.

- AddRule:
    - Unlike the other mutation operations, which affect only rules, the AddRule operator applies to a description. Each description has a

chance for having a new rule added. This chance is inversely proportional to the Coverage of the description. The lower the Coverage, the greater the chance that a new rule will be added to the description to help improve the Coverage. The new rule will consist of a single random set as the antecedent and a single random set as the consequent.

3.4.6 Linear Dropping

After a final description for the data has been reached, each of the rules in the description will undergo a process called "linear dropping." Due to the random way in which rules are constructed in this process, it is possible that they might contain extraneous information in the antecedents of the rules. For example, a rule from Table 2.1 could be:

(deductible_met, yes) AND (type_of_visit, lab) AND

(participating_physician, no) $\rightarrow$ (reimbursement, 70%)

In reality, only the first two sets are needed; the addition of "AND (participating_physician, no)" does not add any new information to the rule. Therefore, it can be dropped from the rule.

To perform linear dropping, the process will start with the last set in the antecedent of the rule. In the previous example, the process would start with (participating_physician, no) because it is the set furthest to the right in the antecedent. This set becomes the "drop set." The antecedent will then be evaluated

as if the drop set were no longer part of it. If the result from the evaluation of the new antecedent is the same as the result of the old antecedent, or if the result is a subset of the consequent, then the antecedent is changed to no longer contain the drop set (including the operator to the left of the set). The process continues by selecting a new "drop set," the set immediately to the left of the previous drop set.

Once the linear dropping process is complete, each rule should contain only the minimal amount of information required by the antecedent to describe the consequent (or a subset of it).

3.5 Implementation

When Arcanum was first conceived, much work was done to establish "proof of concept." In order to ease the coding and provide quick and easily testable code, the Visual Basic version 6 programming language was chosen. It was very easy to write the set objects and perform set operations on them. To further verify that these set objects and the relevant code would be useful and productive, some test programs were written which incorporated these objects. These tests consisted of Visual Basic implementation of a Markov Chain Monte Carlo technique that could assemble sets together into rules and then test them against a data set. A second implementation of the sets was done by developing a Visual Basic version of the LEM2 data mining algorithm. The Markov Chain Monte Carlo technique was very slow and produced very low coverage. The LEM2 implementation, however, ran quite well and produced good results in testing, which was performed by looking for motifs in various protein families (Chen and Taylor, 2008).

The experiments were conducted concurrently on several different machines. This is due in large part to the significant amount of time required for the process to complete. In order to gather the necessary experimental results, many different machines were used at the same time, each running a different experiment. All of the machines used a Microsoft Windows XP platform and had a minimum of 1GB of RAM. All of the computers had at least a 3GHz or faster processor, with the exception of one machine that only had a 1.3GHz processor. In total, four different computers were used to conduct the experiments.

Chapter 4. Evaluation

4.1 Data Sets

In order to evaluate Arcanum, several things are necessary: a group of data sets to be mined, other published results for the data sets, and some metrics that can be compared between the published results and the results from Arcanum. From a review of the literature (Bacardit and Butz, 2004; Au, *et al.*, 2003; Ratanamahatana, 2008), it appears that the most common metrics used are either *accuracy* or *error*. The accuracy metric is already calculated by Arcanum, and the error metric is easily obtained (1 – accuracy).

The specific data sets used for experimentation were selected from the University of California at Irvine Knowledge Discovery in Databases Archive (Hettich and Bay, 1999). The UCI KDD archive contains a large collection of data sets which have been used by several research groups. Using this collection provided two distinct advantages: (1) access to several diverse data sets from a single source; and (2) other researchers have published the results of their data mining on these data sets, thus providing a means of measuring the success of Arcanum by comparing it to the results of other approaches.

When selecting the data sets to be used for evaluating Arcanum, several factors were considered. It was desired to select at least six data sets: all nominal attributes, all numeric attributes, and a mix of nominal and numeric attributes; each of these with and without some missing values. Higher priority was given to those sets

with missing attribute values, as most real-world data sets have missing values. Numeric attributes were also very important because Arcanum uses a discretization method when dealing with them. Thirteen data sets were actually chosen from the UCI KDD archive. These sets were selected because published results were available and because these sets covered the desired range of nominal and numeric attributes, both with and without missing attribute values. The data sets are described on more detail in section 5.1.

Another important factor in choosing data sets will be the volume of available literature with respect to the data set. When more results can be obtained from other researchers using the same data set, more comparisons can be made, allowing more insight into the results from Arcanum. Even if the accuracy of the results are similar, it is still of interest to compare the actual results to see if the resulting rule sets are similar.

4.2 Separating Training and Testing Data Sets

It is crucial to separate data sets into two separate sets: one to be used for training and the other to test the results obtained from training. The data sets in the UCI KDD Archive are not split into these separate sets, thus some preprocessing must take place before they can be used in Arcanum.

Each data set selected from the UCI KDD archive was run using a K-fold cross-validation method, with K=10 (Souza, 2005). Thus, each tuple in a data set was randomly dropped into one of 10 buckets, resulting in 10 subsets of equal size. This

was done by randomly selecting a tuple to place in bucket 1, then randomly selecting a tuple for bucket 2, *etc*. until every tuple had been placed in a bucket. For each run of the algorithm, 9 of the subsets were used for training and the remaining set was used to test the resulting data model. In the following run, a different testing set was used. So, on the K^{th} iteration the K^{th} set was used for testing and the other 9 sets were used for training. This was done 10 times, so each subset was used once as a test set and 9 times as a training set. The results for all 10 runs were averaged together to obtain the overall results for the data set.

4.3 Metrics

The metrics measured and reported by Arcanum are:

- Accuracy: measured for the entire description, this is the percentage of correct classifications (true positives and true negatives) using the derived rules.
- Coverage: measured for the entire description, this is the percentage of the data set for which a classification is attempted.
- Precision: measured for each rule, this is the percentage of elements from the set defined by the antecedent of the rule that are also members of the consequent of the rule. The inverse, 1 – Precision, is the percentage of exceptions to the rule.
- Recall (Sensitivity): measured for each rule, this is the percentage of the decision class covered by the rule.

These metrics were selected in order to be able to measure the effectiveness of the overall description (Accuracy and Coverage), as well as the effectiveness of each individual rule (Precision and Recall). Because each rule only indicates membership and not lack of membership, measurement of negative classification can only be done for the entire description.

4.4 Comparisons

In addition to Arcanum, two other algorithms were also implemented based on the same core components of the software. The exact same code for set objects was shared between Arcanum and these programs. These programs were implemented for several reasons. First, this was done in order to test the code that drives the set objects and verify that it works properly. Second, these programs were implemented to prove that the code was reusable so that it could be used in different applications that also make use of set theory. Lastly, these programs can help evaluate the performance of Arcanum by providing their own descriptions of data sets, as well as run-time comparisons. They can also be used to generate rules to be introduced into Arcanum to test the rule-seeding portion of the program.

4.4.1 LEM2

The LEM2 algorithm (Learning by Examples Module), developed by Dr. Jerzy Grzymala-Busse at the University of Kansas, is a proven data mining technique (Chmielewski and Grzymala-Busse, 1996). Given that the pseudo-code for the algorithm was available, it was implemented to test the set theory portion of

Arcanum's code, providing a means of testing the object models within the code and ensuring that everything worked properly. This LEM2 implementation has been used on several data sets, including a non-trivial data set involving protein sequences, with great success (Chen and Taylor, 2006). Thus, the Arcanum process was built confidently upon the same object models. LEM2 was also used to generate rule sets that could be used as seed rules in the testing of the hypothesis-refinement ability of Arcanum.

4.4.2 Markov Chain Monte Carlo

Markov Chain Monte Carlo (MCMC) is a technique used to gather random samples from a probability distribution (Ridgeway and Madigan, 2002). First, a Markov model of the data is constructed. In this specific implementation, the Markov model is derived from the training data set. Each attribute is given an equal probability of selection. Thus if there are five attributes, then any attribute has a 1/5 chance of being selected. Within each attribute, the features of the attribute are assigned probabilities based on their frequency within the attribute. Therefore, features that occur more often have a higher probability of selection.

In order to use MCMC to generate rules, a data mining variant of MCMC was developed. This implementation of MCMC created rules by selecting a random set and then selecting additional random sets in order to create an antecedent for a rule. This process continued until a decision class was selected, forming the consequent, and thus completing the rule. This random rule was then tested against the data set to measure the precision and recall of the rule. If it fell below the user-provided

threshold value for either metric, then the rule was discarded. If the rule was accepted, it was added to the rules in the description and another random rule was generated. This process continued until a user-specified number of rules had been sampled.

When used for data mining, this technique was slow and not nearly as accurate as LEM2. However, the random sampling can sometimes find additional patterns within data that LEM2 ignores because the examples have already been covered by a different rule. Implementing the MCMC method allowed for further testing of code and helped ensure that set objects, rule parsing, and metrics were all working properly. The results from MCMC could potentially serve as a lower limit for measuring the success of Arcanum. However, MCMC was run several times on the "Breast.C" data set but was never able to achieve better than 50% coverage of the data. As such, while MCMC was implemented as part of this research, it was never used for comparison or to generate initial rule sets to test hypothesis-refinement in Arcanum.

4.5 Efficiency

The most obvious measure of genetic algorithm efficiency is the number of generations required to reach a solution. However, by the nature of genetic algorithm, the number of generations cannot be directly controlled. Sometimes an upper limit is placed on the number of generations, so that if the limit is exceeded the program terminates. The number of generations required by the genetic algorithm is

affected by several parameters: frequency of mutation, crossover method, encoding method, calculation of the fitness function, and size of the population.

The population size should be high in order to help ensure more diversity within the population. With more diversity comes an increasing chance of finding better solutions. However, as the population size increases, so does the amount of physical memory required to represent it in the computer. The frequency of mutation will need to be adjusted by evaluating the performance of Arcanum on some trial data sets to see which frequency of mutation will help produce the fewest number of generations before a solution is reached.

Chapter 5. Experimental Results

Testing of Arcanum was done in two phases. The first phase was to determine if Arcanum could be used to perform data mining. The second phase tested the ability of Arcanum to perform hypothesis refinement, taking a previously generated data model and attempting to improve it. This chapter contains the results and discussion of both phases of testing.

5.1 Data Sets

Data Set	Attribute Values	Missing Attribute Values
Breast.C	Mostly nominal	None
Breast.W	Entirely numeric	Very few
Cleve	Nominal and numeric	Very few
Glass	Entirely numeric	None
Hepatitis	Entirely numeric	Many
House-votes	Entirely nominal	Many
Iris	Entirely numeric	None
Ionosphere	Entirely numeric	None
Liver.Bupa	Entirely numeric	None
Lymph	Entirely numeric	None
Pima.Diabetes	Entirely numeric	None
Wdbc	Entirely numeric	None
Wpbc	Entirely numeric	Very few

Table 5.1. Description of data sets used for testing Arcanum.

To test the data mining ability of Arcanum, 13 data sets were chosen from the University of California, Irvine, Knowledge Discovery in Databases repository

(Hettich and Bay, 1999). These data sets were chosen because of the availability of published results from other data mining techniques. The sets used, and their characteristics, are highlighted in Table 5.1.

As shown in Table 5.1, most of the data sets were comprised of entirely numeric data, and thus required discretization in order to create numeric ranges for the data. Only three of the data sets contained any nominal, or text, data. Five of the sets were missing at least some attribute values. Using these data sets, Arcanum was tested on both nominal and numeric attributes, and each of those was tested both with and without any missing attribute values.

As indicated in Chapter 4, there were many factors that determined which data sets were chosen. Chief among these factors was the availability of published results on the data sets. While only three of the thirteen sets contain nominal data, the type of data being considered is secondary to the fact that results were compared from different techniques using the same data sets.

5.2 Data Mining

In order to determine which weighting scheme to use for Arcanum, some preliminary tests were run in order to gather some base-line data. Two different data sets, “Breast.C” and “Iris,” were selected for this purpose due to their relatively smaller sizes compared to the other sets (Hettich and Bay, 1999). Arcanum was then run on each set 11 times with different weights. The weights started at 100% accuracy with 0% coverage and were incremented/decremented by 10% (90/10,

80/20, etc.) each experiment until the last experiment had a weight of 0% accuracy and 100% coverage. Table 5.2 contains the results of these experiments, which were performed with a population size of 250, 25 generations, and using the entire data set as the training set (thus, no validation).

	Breast.C			Iris		
Weight	Accuracy	Coverage	**Product**	Accuracy	Coverage	**Product**
100 / 0	100.00	17.48	**17.48**	100.00	14.22	**14.22**
90 / 10	81.97	88.81	**72.80**	99.17	76.67	**76.00**
80 / 20	81.25	89.51	**72.73**	97.52	100.00	**97.52**
70 / 30	77.83	100.00	**77.83**	94.60	99.33	**94.00**
60 / 40	77.39	100.00	**77.39**	97.24	97.33	**94.64**
50 / 50	76.89	100.00	**76.89**	94.83	100.00	**94.83**
40 / 60	77.05	100.00	**77.05**	96.64	100.00	**96.64**
30 / 70	77.16	99.65	**76.89**	93.66	100.00	**93.66**
20 / 80	75.31	100.00	**75.31**	94.61	100.00	**94.61**
10 / 90	77.41	100.00	**77.41**	83.89	100.00	**83.89**
0 / 100	43.75	100.00	**43.75**	34.84	100.00	**34.84**

Table 5.2. Base-line experiments to determine Arcanum weighting scheme.

From Table 5.2, it can be seen that there is a significant change in results around the 80/20 weighting scheme. From 90/10 to 80/20 in the "Iris" data set, there is significant improvement. However this leap occurs between 80/20 and 70/30 in the "Breast.C" data set. The best result in "Breast.C" occurred with the 70/30 weighting scheme. The best result in "Iris" occurred with the 80/20 weighting scheme. The most significant difference between the two schemes was in the difference in Coverage in the "Breast.C" set. The 70/30 scheme provided much better Coverage,

and so it was chosen as the weighting scheme for the Arcanum data mining experiments.

After selecting a weighting scheme, results were gathered from all of the data sets. The results for Arcanum are contained in Table 5.3 and are compared to the results reported for GAssist and C4.5 on the same data sets. All of the results from Arcanum used a population size of 500, a maximum of 50 generations, a precision

Data Set	Accuracy	Coverage	Arcanum	GAssist	J4.8 (C4.5)	
Breast-c	77.2	83.9	**64.8**	**70.5**	**75.5**	
Breast-w	96.7	98.6	**95.3**	**N/A**	**95.0**	+
Cleve	68.8	83.7	**57.6**	**80.4**	**76.8**	
Glass	89.4	48.1	**43.0**	**66.8**	**66.8**	
Hepatitis	91.9	53.0	**48.6**	**89.8**	**83.9**	
House-votes	97.5	98.4	**95.9**	**97.1**	**96.3**	+
Iris	98.6	96.0	**94.6**	**95.3**	**96.0**	+
Ionosphere	76.9	87.3	**67.1**	**92.0**	**91.5**	
Liver.Bupa	84.5	59.0	**49.8**	**62.4**	**N/A**	
Lymph	72.6	88.5	**64.3**	**80.8**	**77.0**	
Pima.Diabetes	77.7	96.0	**74.6**	**74.7**	**73.8**	+
Wdbc	95.6	97.0	**92.7**	**94.3**	**N/A**	+
Wpbc	78.2	64.3	**50.3**	**75.3**	**N/A**	

Table 5.3. Comparison of Results from Arcanum to GAssist and C4.5.

weight of 70%, and a recall weight of 30%. Thus, accuracy was more highly valued in the models than coverage. Note that this population size and number of generations is twice that of the previous experiments run to obtain the weighting scheme. Table 5.3 shows the Accuracy and Coverage metrics from Arcanum for each data set, and Figure 5.1 displays these results in a bar graph for a visual comparison. The overall result reported for Arcanum is the product of these two values (Accuracy * Coverage). Accuracy and Coverage were not reported separately for GAssist or C4.5.

GAssist is a genetic algorithm for data mining, developed using the Pittsburg approach (Bacardit and Garrell, 2003). J4.8 is a Java implementation of the C4.5 algorithm and is included as part of the WEKA software (Witten and Frank, 2005). These two techniques were ideal for comparison because C4.5 is still considered one of the very best data mining algorithms, and GAssist is a genetic algorithm used for data mining.

In 5 of the 13 data sets examined, Arcanum was able to develop a data model that was comparable to GAssist and C4.5 (marked with "+" symbols). In these five cases, Arcanum was within 2% of the accuracy of the other methods, and in two of the cases, "Breast.w" and "Pima.Diabetes," it obtained a better data model than C4.5. While there is certainly some improvement that needs to be done before Arcanum can compete on all levels with these algorithms, these results show that the potential exists. When compared to GAssist, Arcanum gains further appeal because it has

significantly fewer parameters than GAssist, demonstrating that the results are more from the algorithm than from possible "fine-tuning" by the user. Fewer parameters

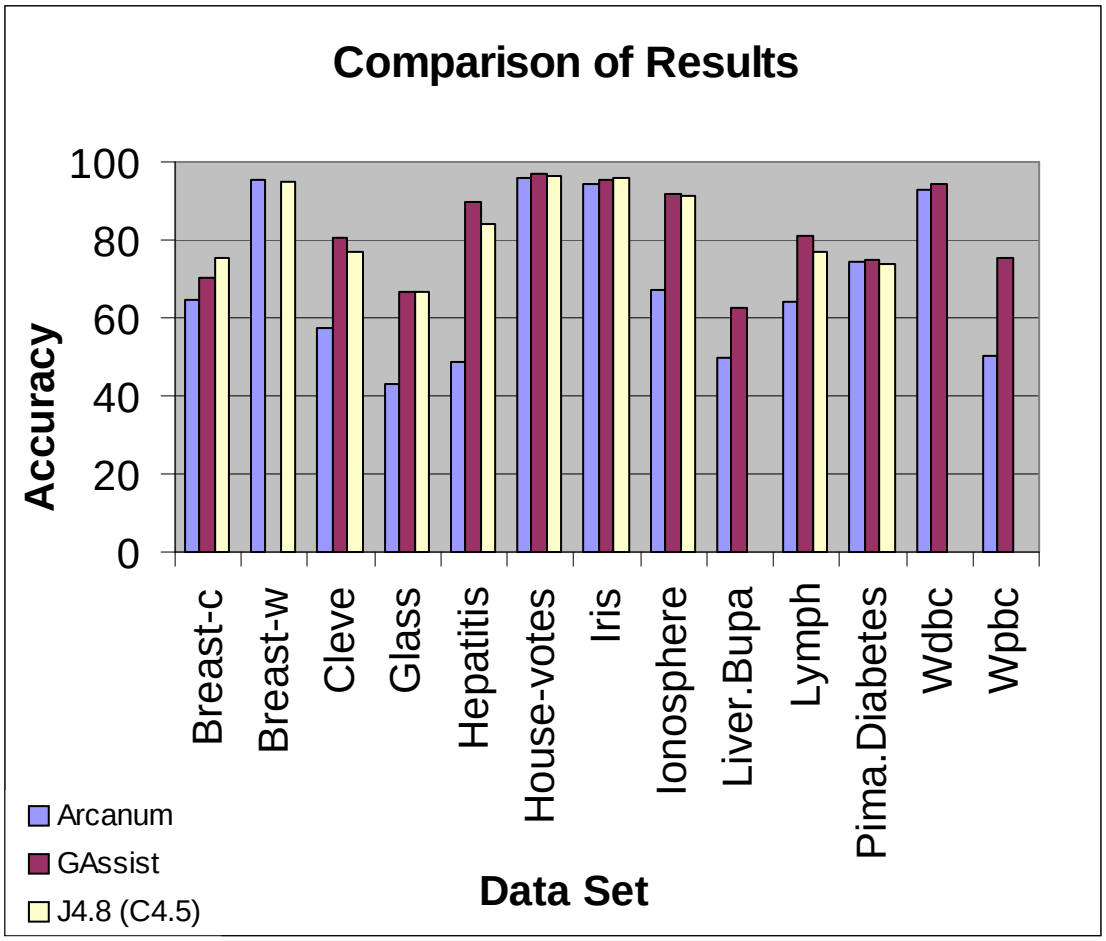

Figure 5.1. Graph of results from Arcanum, GAssist, and C4.5.

also mean that fewer experiments are required to determine the optimal setting for the parameters. In a paper comparing GAssist to XCS, Bacardit mentions 19 parameters for GAssist (Bacardit, 2007). Arcanum only has, at most, 8 parameters if one counts the type of model validation scheme used. For the purposes of this dissertation, only 4 of the Arcanum parameters were ever varied:

1. the size of the population

2. the maximum number of generations to be run
3. the weight placed on precision
4. the weight placed on recall

The 19 parameters for GAssist mentioned by Bacardit include:

1. crossover probability
2. selection method
3. tournament size (the tournament selection method was used)
4. population size
5. probability of mutating an individual
6. initial number of rules per individual
7. probability of “1” in initialization
8. rule deletion operator: iteration of activation
9. minimum number of rules
10. fitness function
11. iteration of activation of fitness function
12. initial theory length ratio
13. weight relax factor
14. knowledge representation method
15. split and merge probability
16. probability of re-initialization at initial iteration
17. probability of re-initialization at final iteration
18. merge restriction probability

19. maximum number of intervals

As previously mentioned, the Arcanum parameters focused heavily on precision over recall, and thus favored models with high accuracy even though the coverage of the model might be low. In the cases where Arcanum did not perform as well as GAssist and C4.5, the predictive accuracy of Arcanum's models was very high but had extremely limited coverage. In other words, when the model made a prediction, it was highly accurate, but the model failed to make any prediction at all for a large percentage of data. This indicates that one of the areas of improvement for Arcanum might be in improving the fitness function to help ensure greater coverage in the resulting data models. Future works needs to be done to determine if this could be achieved by perhaps changing the weight parameters in Arcanum to favor Recall more highly than Precision. This is suggested because the resulting data models had a high Precision and a low Recall, which reflected the way the weights were assigned; Accuracy was favored over Coverage.

It is also worth noting that there is no readily apparent way to stereotype the data sets upon which Arcanum would be successful. The data sets upon which it produced very successful results included sets with nominal attributes, sets with numeric attributes, sets that had large numbers of missing attribute values, sets with only a few missing attribute values, and sets with no missing attribute values at all. Thus, the factor or factors which determine when Arcanum will be successful are not obvious, requiring future experimentation to determine those factors.

From the results discussed in this section, it appears, in at least some cases, that Arcanum has the potential to perform data mining on a level comparable with other successful techniques, such as GAssist and C4.5. Some refinements are required to reach that level consistently, but the results suggest that such performance is possible using the multi-tiered genetic algorithm technique described in this dissertation.

5.3 Hypothesis Refinement

After the data mining proof of concept stage, Arcanum was then tested on whether it could be used to improve a previously generated data model. During this stage, data models from different techniques were used as a starting point for Arcanum. These data models included some generated by Arcanum in the data mining stage, some models generated by the LEM2 algorithm, and a data model generated by C4.5, using the J4.8 implementation included in WEKA.

The six data models chosen from Arcanum represent the two worst data models from Arcanum: Glass and Hepatitis; three models with very good results: Breast.W, House-votes, and Wdbc; and one model in the middle: Breast.C. The refinements were performed while keeping the same population size and number of generations used while data mining, 500 and 50, respectively. Unlike the previous experiments used to compare the different techniques, the refinement experiments were run with equal weighting of Precision and Recall. The equal weighting scheme was chosen to allow Arcanum to take advantage of improvements that could be made in precision or recall, thus giving Arcanum flexibility in how it attempted to improve

the previous models. Table 5.4 shows the results after this refinement. The results for the Glass data set were actually significantly lower using a 50/50 weighting

	Original Data Set			After Refinement		
Data Set	**Accuracy**	**Coverage**	**Product**	**Accuracy**	**Coverage**	**Product**
Breast-c	77.2	83.9	**64.79%**	80.0	89.4	**71.47%**
Breast-w	96.7	98.6	**95.28%**	85.4	87.7	**74.93%**
Glass*	89.4	48.1	**43.0%**	89.6	57.5	**51.48%**
Hepatitis	91.9	53.0	**48.63%**	85.0	61.1	**51.94%**
House-votes	97.5	98.4	**95.88%**	97.2	98.4	**95.67%**
Wdbc	95.6	97.0	**92.69%**	94.1	96.8	**91.12%**

Table 5.4. Results of hypothesis refinement on Arcanum data models

scheme (the result was 34.4%), so it was run again using a 70/30 weighting scheme to see if there would be improvement. The result shown for Glass in Table 5.4 is the result after the 70/30 weighting scheme.

While only half of these data models were improved after refinement, the results show there is potential improvement to be gained by using Arcanum for hypothesis refinement. After refinement, the Arcanum model for the Breast-c set yielded better results than the GAssist model, as the Arcanum model was refined to 71.47% vs. the 70.5% of GAssist. The three models that failed to improve were already very good, comparable to both C4.5 and GAssist, with the Breast-w model already better than the C4.5 result. Figure 5.2 compares the results of each set before and after hypothesis refinement.

Of the six models originally generated by Arcanum, the program was able to improve the quality of its own data models through hypothesis refinement. One model was improved to the point it was comparable with both GAssist and C4.5. Of the models that failed to improve, each of those were already comparable with GAssist and C4.5, and one of them was even better than C4.5. These results suggest

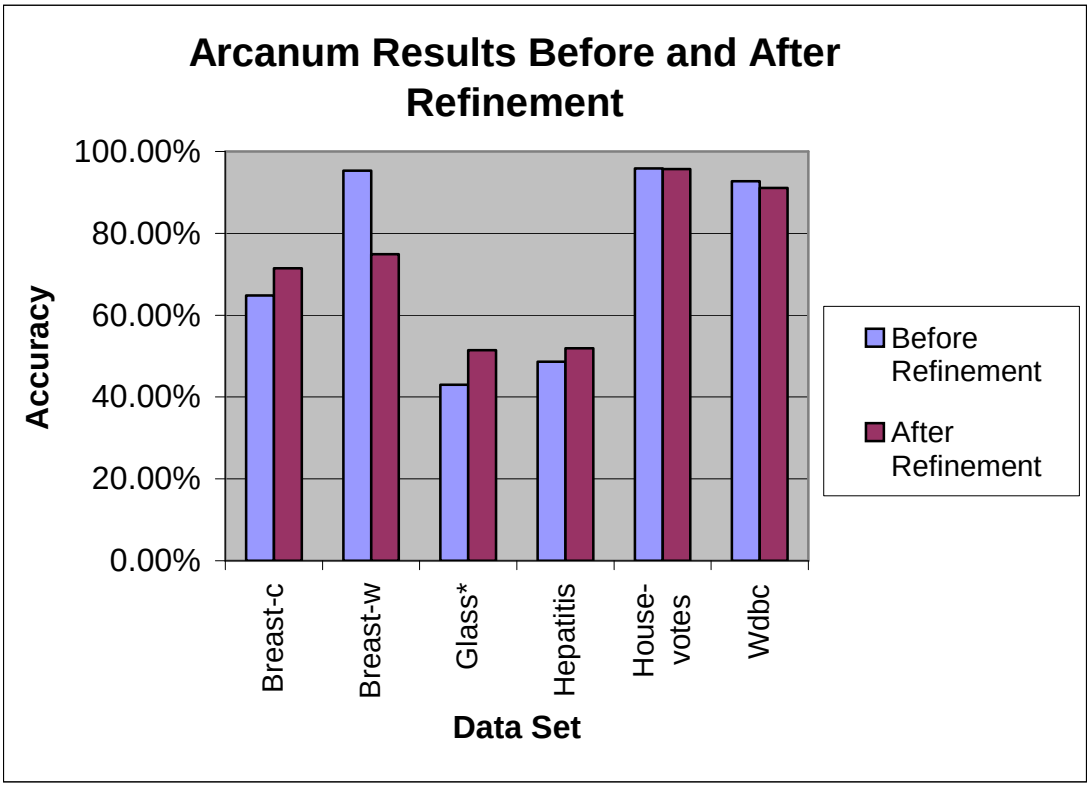

Figure 5.2. Comparison of Arcanum results before and after hypothesis refinement.

that hypothesis refinement is a potentially useful tool and that, at least in some cases, it can take a data model and make it better.

However, each of these data models were generated by Arcanum. To further test the abilities of Arcanum's hypothesis refinement, three data models generated by the LEM2 algorithm were used for hypothesis refinement. These results, shown in

Table 5.5, used a population size of 500, ran for 50 generations, used a 50/50 weighting scheme, and K-fold cross-validation with K=10. Figure 5.3 is a graph of these comparisons.

Arcanum was able to improve the results of each of these three data models from LEM2 using hypothesis refinement. It is also worth noting that in each case, the resulting model is better than the one obtained when Arcanum performed data mining

	LEM2			After Refinement		
Data Set	**Accuracy**	**Coverage**	**Product (accuracy x coverage)**	**Accuracy**	**Coverage**	**Product (accuracy x coverage)**
Iris	99.2%	86.0%	**85.33%**	95.4%	99.3%	**94.79%**
Hepatitis	96.2%	61.9%	**59.56%**	73.9%	97.1%	**71.78%**
House-votes	99.7%	71.0%	**70.8%**	98.0%	98.6%	**96.68%**

Table 5.5. Results of hypothesis refinement on LEM2 data models.

on the same data set. After refinement, the data model for "Hepatitis" is much better than the one obtained using just Arcanum. The "Iris" and "House-votes" models are both comparable to GAssist and C4.5, with the "House-votes" model slightly better than C4.5. While there was significant improvement in the LEM2 models after hypothesis refinement, the results reflect the success of the original data models obtained from Arcanum. The Arcanum data models for "Iris" and "House-votes" were both comparable to GAssist and C4.5, while the data model for "Hepatitis" was inferior. This remained true even after hypothesis refinement on the LEM2 models – "Iris" and "House-votes" were comparable while "Hepatitis," even though remarkably improved, was still inferior to the GAssist and C4.5 models for the same data set.

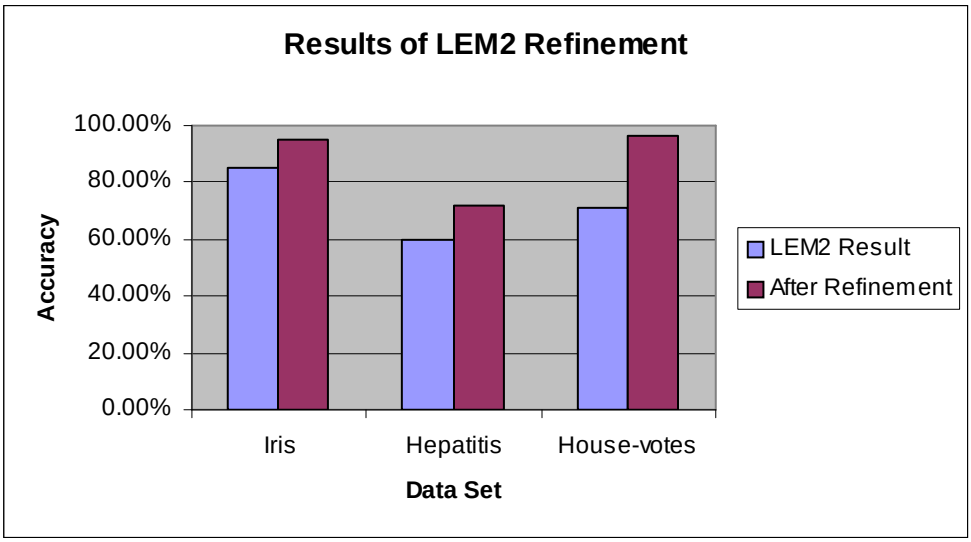

Figure 5.3. LEM2 results before and after hypothesis refinement

One last series of experiments were run to test Arcanum's hypothesis refinement ability. A data model was obtained using J4.8, the Java implementation of C4.5 included in WEKA. The resulting decision tree was then converted into a set of

Weighting Scheme	Accuracy	Coverage	Product (accuracy x coverage)
30/70	74.6%	89.3%	**66.6%**
50/50	75.0%	84.7%	**63.5%**
70/30	75.1%	94.0%	**70.6%**
80/20	77.7%	80.8%	**62.8%**
50/50 (population 1,000)	75.1%	94.6%	**71.1%**
70/30 (population 1,000)	77.2%	98.6%	**76.1%**

Table 5.6. Results of hypothesis refinement on C4.5 Breast-c data model.

rules that could be used as input to Arcanum. For these experiments, the "Breast.C" data set was chosen. This was done because it was a set upon which Arcanum had

not performed comparably to GAssist or C4.5, because there was plenty of room to potentially improve the C4.5 model above the 75.5% accuracy it had achieved, and because the C4.5 model demonstrated superior performance to the GAssist model.

Table 5.6 shows the different weighting schemes and population sizes used in an effort to improve the C4.5 data model. As shown in Table 5.3, the accuracy of the original C4.5 model for "Breast.C" was 75.5%. After several different weighting schemes failed to improve the C4.5 model, the decision was made to try again but with double the population size, resulting in a population size of 1,000 rather than 500. After doubling the population size used by the algorithm, Arcanum was finally able to slightly improve upon the C4.5 data model. Figure 5.4 graphically demonstrates the result of each experiment. The resulting model made a slight sacrifice in coverage in order to improve overall accuracy, producing a slightly better model than the original. Thus, Arcanum was able to use hypothesis refinement to improve a C4.5 data model, and was even able to do so on a data set upon which Arcanum did not perform as well as C4.5.

These three sets of experiments, using data models produced from three different techniques, each show that Arcanum is able to perform hypothesis refinement to improve upon previously generated data models. This suggests that the hypothesis refinement capability of Arcanum could be a potentially useful tool for data miners. Using Arcanum, previously constructed data models can, in some cases, be improved. Even in the cases where Arcanum fails to improve upon a data model, the worst case situation is that the original model is used, meaning that researchers

are no worse off for having tried hypothesis refinement. In the worst case, they keep

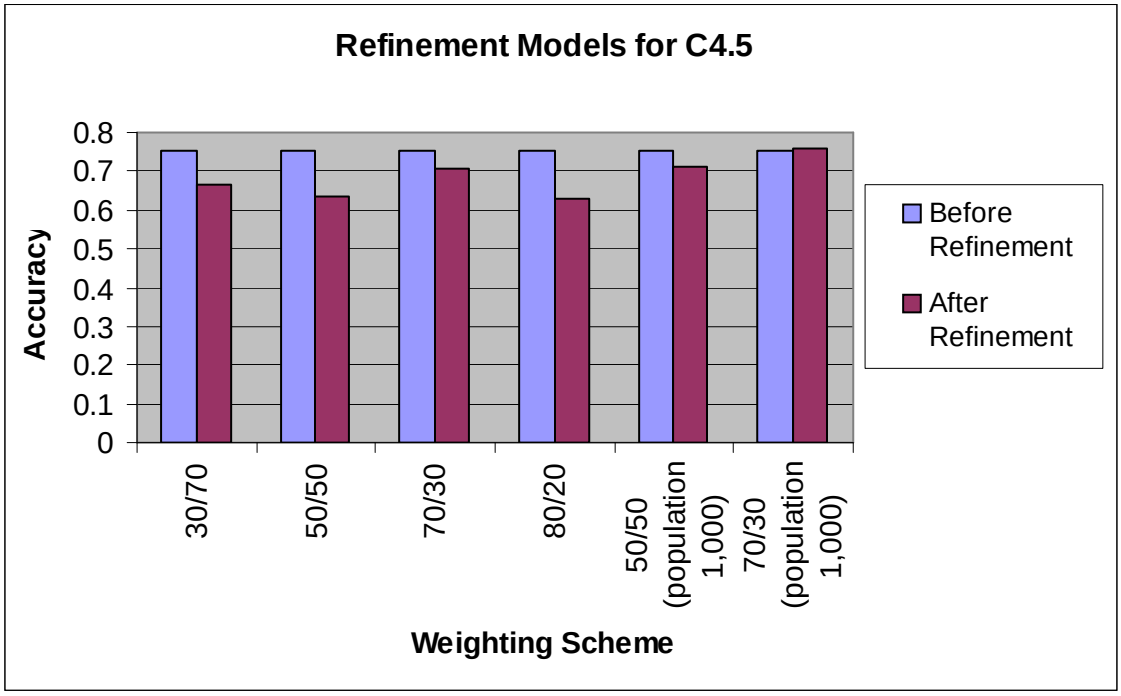

Figure 5.4. Results of hypothesis refinement on C4.5 model using various weighting schemes.

their original model, but the potential for improving the model using hypothesis refinement makes attempting the process worthwhile.

5.4 Rules

5.4.1 Arcanum rule evolution

This section lists rules generated by Arcanum. The rules listed include random rules from the start of the process, rules from the middle of the process, and the final set of rules. The rules listed are from the "Breast.C" data set and were obtained using a population size of 500 over 50 generations with a 70/30 weighting scheme.

It is important to note that it is currently impossible to determine for certain which rules evolved from which. While it is possible to find rules that are very similar, any conclusion that one rule led to the other is conjecture.

One last caveat is to make note that some of the rules, while the same, have different Precision and Recall (same as Accuracy and Coverage, but for individual rules) values. This could be accounted for by the K-fold cross-validation scheme used. It cannot be assumed that the same subset of the data was used in each of the generations listed below. They rules could potentially have come from different K's in the K-fold cross-validation.

Initial random rules:
Accuracy = 74.32%
Coverage = 100.00%

Precision: 80.10%; Recall: 80.10%
(deg_malig,1..2) -> (class,no-recurrence-events)

Precision: 78.40%; Recall: 83.08%
(inv_nodes,0-2) -> (class,no-recurrence-events)

Precision: 63.16%; Recall: 5.97%
(tumor_size,35-39) -> (class,no-recurrence-events)

Precision: 52.94%; Recall: 52.94%
(deg_malig,3) -> (class,recurrence-events)

Evolved Rules (from Generation 20):
Accuracy = 78.27%
Coverage = 99.22%

Precision: 79.12%; Recall: 80.45%
(deg_malig,1..2) -> (class,no-recurrence-events)

Precision: 78.65%; Recall: 84.36%
(inv_nodes,0-2) -> (class,no-recurrence-events)

Precision: 80.72%; Recall: 74.86%
(irradiat,no) AND (node_caps,no) -> (class,no-recurrence-events)

Precision: 77.27%; Recall: 85.47%
(node_caps,no) -> (class,no-recurrence-events)

Precision: 96.00%; Recall: 13.41%
(tumor_size,10-14) -> (class,no-recurrence-events)

Precision: 100.00%; Recall: 2.23%
(tumor_size,5-9) -> (class,no-recurrence-events)

Precision: 59.62%; Recall: 39.24%
(node_caps,yes) -> (class,recurrence-events)

Final Rules:
Accuracy = 77.55%
Coverage = 100.00%

Precision: 72.41%; Recall: 35.59%
(age,50-59) -> (class,no-recurrence-events)

Precision: 100.00%; Recall: 3.95%
(breast_quad,right_low) AND (deg_malig,1) -> (class,no-recurrence-events)

Precision: 79.78%; Recall: 80.23%
(deg_malig,1..2) -> (class,no-recurrence-events)

Precision: 77.78%; Recall: 51.41%
(deg_malig,2) -> (class,no-recurrence-events)

Precision: 81.25%; Recall: 29.38%
(inv_nodes,0-2) AND (age,50-59) -> (class,no-recurrence-events)

Precision: 100.00%; Recall: 0.56%
(menopause,premeno) OR (tumor_size,0-4) AND (age,20-29) -> (class,no-recurrence-events)

Precision: 76.02%; Recall: 84.18%
(node_caps,no) OR (tumor_size,20-24) -> (class,no-recurrence-events)

Precision: 76.56%; Recall: 27.68%
(node_caps,no) AND (age,40-49) -> (class,no-recurrence-events)

Precision: 76.67%; Recall: 28.75%

(deg_malig,3) AND (node_caps,yes) -> (class,recurrence-events)

Note that one of the original random rules persevered throughout the entire process to remain in the final specimen:

> Precision: 80.10%; Recall: 80.10%
> (deg_malig,1..2) -> (class,no-recurrence-events)

This demonstrates that the multi-tiered genetic algorithm can find and persist rules from the original generation.

It is also interesting that a majority of the final rules reflect the weighting scheme that was used. Most of the rules have a precision between 70% and 80%, while the recall hovers near 30%. There are enough rules that differ to show that rules are not limited to the weighting scheme, but can exceed it. However, it begs the question of whether the current fitness function and the use of the weighting scheme might not inadvertently lead to generating rules that tightly fit the weighting scheme, rather than using it as a guide to evolving the rules. At the same time, this also reflects the results from Table 5.2, where it was empirically observed that a 70/30 weighting scheme gave the best result for the “Breast.C” data set.

To explore the answer to whether the algorithm is tightly fitting rules to the weighting scheme, consider the rules in the final data model, in which 6 of the 9 rules

have a Precision between 60% and 80%. However, of the 3 exceptions, 2 of them have a Precision of 100%. With respect to Recall, 4 of the 9 rules have a Recall between 20% and 40%. Of the 6 rules with Recall metrics outside that range, 2 of them are over 80% and 2 are below 5%. Furthermore, consider the 6 rules that have precision between 60% and 80%. Only 3 of them have a Recall between 20% and 40%. Thus, only 3 of the 9 rules have both a Precision and Recall that is within even 10% of both metrics. Two of the rules fall outside that range on both metrics. So it does not appear to be the case that the weighting scheme causes an inadvertent "overfitting" of rules to the weighting scheme.

5.4.2 Rules after hypothesis-refinement of LEM2

This section contains the rules generated by the LEM2 algorithm for the "House-votes" data set. It also contains rules generated by Arcanum during hypothesis-refinement of the LEM2 rules and the final set of rules that resulted from the refinement. The hypothesis refinement used a population size of 500, 50 generation limit, and a 50/50 weighting scheme.

Rules from LEM2:
Accuracy = 99.68%
Coverage = 71.03%

Precision: 100.00%; Recall: 82.02%
(physician_fee_freeze,n) and (adoption_of_the_budget_resolution,y) ->
(class,democrat)

Precision: 100.00%; Recall: 52.98%
(physician_fee_freeze,y) and (crime,y) and (el_salvador_aid,y) and
(religious_groups_in_schools,y) and (mx_missile,n) and

(adoption_of_the_budget_resolution,n) and (duty_free_exports,n) and (synfules_corporation_cutback,n) -> (class,republican)

Evolved Rules (from Generation 25):
Accuracy = 96.90%
Coverage = 99.31%

Precision: 98.24%; Recall: 62.55%
(crime,n) -> (class,democrat)

Precision: 91.42%; Recall: 79.78%
(education_spending,n) -> (class,democrat)

Precision: 96.15%; Recall: 74.91%
(el_salvador_aid,n) -> (class,democrat)

Precision: 99.19%; Recall: 91.76%
(physician_fee_freeze,n) -> (class,democrat)

Precision: 100.00%; Recall: 82.02%
(physician_fee_freeze,n) AND (adoption_of_the_budget_resolution,y) -> (class,democrat)

Precision: 92.09%; Recall: 97.02%
(physician_fee_freeze,y) -> (class,republican)

Precision: 97.83%; Recall: 80.36%
(synfules_corporation_cutback,n) AND (physician_fee_freeze,y) -> (class,republican)

Final Rules:
Accuracy = 97.86%
Coverage = 98.62%

Precision: 100.00%; Recall: 82.02%

(physician_fee_freeze,n) AND (adoption_of_the_budget_resolution,y) -> (class,democrat)

Precision: 99.40%; Recall: 62.17%
(aid_to_nicaraguan_contras,y) OR (physician_fee_freeze,n) AND (crime,n) -> (class,democrat)

Precision: 97.87%; Recall: 68.91%
(el_salvador_aid,n) AND (adoption_of_the_budget_resolution,y) -> (class,democrat)

Precision: 99.44%; Recall: 67.04%
(mx_missile,y) AND (physician_fee_freeze,n) -> (class,democrat)

Precision: 99.19%; Recall: 91.76%
(physician_fee_freeze,n) -> (class,democrat)

Precision: 100.00%; Recall: 38.58%
(physician_fee_freeze,n) AND (education_spending,n) AND (immigration,n) -> (class,democrat)

Precision: 96.54%; Recall: 94.01%
(physician_fee_freeze,n) OR (el_salvador_aid,n) -> (class,democrat)

Precision: 93.89%; Recall: 46.07%
(religious_groups_in_schools,n) AND (education_spending,n) -> (class,democrat)

Precision: 92.09%; Recall: 97.02%
(physician_fee_freeze,y) -> (class,republican)

In this collection of rules, it can be observed that Arcanum improved the overall results from LEM2 by slightly sacrificing the accuracy of the model in order to improve the coverage. As hypothesized in section 3.4.1., a good rule was found in the first generation, was preserved throughout the generations, and used in the final rule collection from Arcanum:

> Precision: 100.00%; Recall: 82.02%
> (physician_fee_freeze,n) AND
> (adoption_of_the_budget_resolution,y) -> (class,democrat)

Several things can be concluded from this, not the least of which is that LEM2 generated a rule that Arcanum could not improve upon. As was also observed in section 5.4.1, this shows that Arcanum can preserve a rule all the way from the first generation to the final one. From these results, and the results of the C4.5 experiments, an argument can be made for the use of a multi-tiered genetic algorithm for hypothesis refinement. In the worst case, hypothesis refinement cannot improve upon the original model. Thus, the worst that can happen is that the result is the original data model. However, as shown in the experimental data, there is a potential for improving upon the original data model.

Chapter 6. Conclusion

This dissertation has shown how the Pittsburg and Michigan approaches to using genetic algorithms for data mining can be combined using a multi-tiered approach. This technique was implemented in a project called "Arcanum." Testing performed with Arcanum shows that it has the potential to be a successful data mining tool, but some refinements are required first. More importantly, testing of the hypothesis refinement capability of this approach showed that it can take a data model generated by some other technique and improve upon that data model.

6.1 Contributions

The contributions of this research are two-fold: the development of a multi-tiered genetic algorithm technique and its ability to perform not only data mining but also hypothesis refinement. The multi-tiered genetic algorithm is not only a closer approximation to genetics in the natural world, but is also a way of combining the two competing schools of thought for genetic algorithms in data mining.

Perhaps the most notable contribution of this research is the examination of hypothesis refinement. The research in this dissertation shows that hypothesis refinement can be a useful tool for data mining. Once a data model has been achieved, performing hypothesis refinement upon it using the Arcanum technique can improve the overall performance of the model. This shows it to be a valuable step in the data mining process. In the worst case, Arcanum is unable to improve the

previous model. Thus, there are no drawbacks to attempting hypothesis refinement: the outcome is either the original model, or one that is better than before.

6.2 Observations

Numerous observations were made during the course of this research which are worthy of discussion. For example, from empirical observations it appears that the optimum weights for Precision and Recall are 70/30 or 30/70. In nearly every case when multiple models were constructed using different weights, the combination of 70% Precision and 30% Recall obtained the best result. The models obtained using 30% Precision and 70% Recall were often the second best.

Similarly, some experiments were performed to observe what happened when the weights were set to extremes. When the weights were set to 100% Precision and 0% Recall, the resulting model contained a single rule that was 100% accurate (and described only a small portion of the data). When the weights were flipped, the resulting model contained a handful of rules that were so generic they each described a large portion of the data, but did so erroneously. It appears that the algorithm is "constrained" between 70/30 and 30/70. Outside of those bounds for the weights, the results get very skewed toward either Precision or Recall.

During one experiment, something interesting was observed in the performance graph. The graph can be see in Figure 6.1. The green lines show the average fitness of the population over time. There is a different line for each of the

different K-fold runs. The blue lines indicate the fitness of the best specimen encountered. Again, the multiple lines correspond to each of the K-fold runs.

It is readily apparent the something happened during one of the K-fold runs. Given that the graph spans 50 generations from end-to-end, it appears that some local optima were overcome around generation 25. Note the corresponding effect this had on the best specimen. Whatever happened was notable and resulted in a jump in improvement in average fitness, as well as a radical improvement in the best specimen, which continued to improve at a rapid rate, much faster than it had achieved at any point prior to this "discovery."

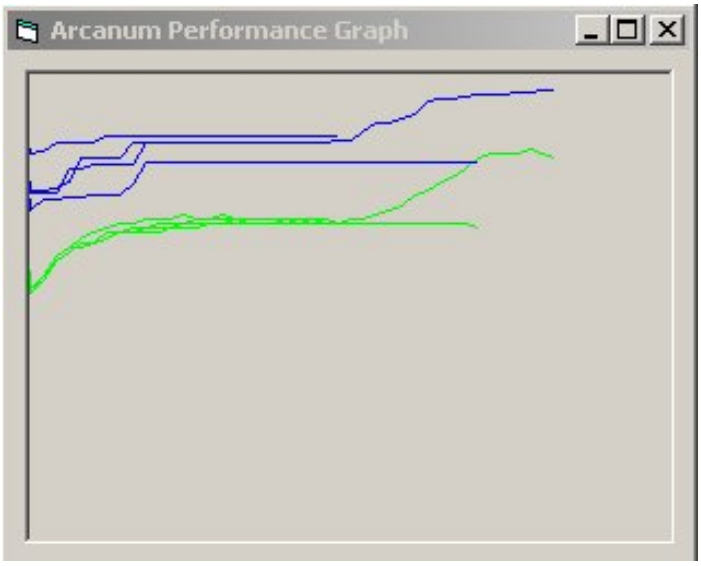

Figure 6.1. An atypical Arcanum performance graph.
Accuracy of best specimen vs. average fitness over K runs.

There are several possible explanations for this phenomenon. It could be that during this particular K-fold run that the training data were optimal. It is also possible, that the population received a large number of "clones" of the best specimen, which would explain the sudden upswing in average fitness, but does not

explain why fitness continued to improve at such a tremendous pace. It is also possible that the mutation operators in the genetic algorithm happened to overcome some local optima at that point which none of the other runs were able to achieve. The last two possibilities pose some interesting questions which will be discussed further in section 6.4.

6.3 Limitations

One of the biggest limitation for this research has been the amount of time required for Arcanum to complete an experiment. The process requires considerable time. This is likely due to the way the process was coded. Minimal efforts were made to ensure that the code was optimized. Efforts were instead focused on ensuring that the process was bug-free and that the proof-of-concept could be tested sufficiently.

Another limitation of this work is that the mutation operations are very simple. It is likely that the process could be improved by investigating more robust or intelligent mutation operations.

It is also difficult to track how a particular rule changes over time, especially as part of refinement. While it is possible to show how a rule potentially evolved, this is a manual effort and requires non-trivial amounts of time to tie two rules together. Even at that, the linkage between an initial rule and a final rule is merely conjecture; it cannot be proven that the final rule truly evolved from the earlier one.

Another limitation of Arcanum is that it does not check for a minimal covering. It is possible for two rules to describe the same subset of the data, or for a

rule to be completely covered by a different rule. In other words, some rules might actually be redundant and unnecessary. Arcanum does not perform any checks to see if this situation exists. This decision was made because one of the visions of the research is for it to lead to a process that can mine a collection of data and find all of the rules within the data. Redundancy is allowed within Arcanum because having multiple rules that describe the same data could be useful in cases where a piece of information needed for a specific rule might be missing, and so while that rule might not be able to make a classification, there might be sufficient data for a different rule to make a classification.

6.4 Future Work

As discussed in section 6.2, some interesting results were observed during experimentation. Two of the possible explanations suggest some future work that should be pursued.

In the case of the genetic operators perhaps overcoming some local optima, this suggests that some additional work should be done to explore the possibility of increasing the frequency of mutation based upon the consistency of the average fitness level. In other words, if the average fitness of the population has been consistent over time, then the chances for mutations should increase to bring about increased genetic diversity within the population. This would help ensure that the population does not become genetically stagnant with a shallow gene pool from which to create new generations, and thus failing to improve upon the best specimen.

This could be achieved using a chance of mutation based upon simulated annealing techniques, where the chance of mutation increases as the average fitness of the population remains relatively consistent.

The second possibility suggested is that the population received a number of "clones" of the best specimen, resulting in a sudden jump in average fitness because multiple members of the population had the same genes as the best specimen. However, this by itself fails to explain why both the average fitness and the fitness of the best specimen continued to improve at an increased rate. The other piece of the puzzle might be found in Arcanum's hypothesis refinement. If the population were seeded with clones of the best specimen, or even slightly modified versions, this would be similar to performing hypothesis refinement, which was shown to be very successful. It is possible that a better way to perform data mining in Arcanum would be through utilizing hypothesis refinement. Whenever a new best specimen is found, treat it as if it were a "hypothesis" and seed the population with it as described in section 3.4.1. This could make use of the proven strength of the Arcanum technique and potentially improve the deficiencies that were encountered when performing data mining with Arcanum.

Future work should also include an examination of these same data sets using different weighting schemes other than the ones used for this dissertation. All of the results favored Accuracy over Recall, which is not a surprise considering that a weighting scheme of 70% accuracy and 30% recall was used. While this weighting scheme was obtained from empirical results, as discussed in section 5.2, it is possible

that better results could be obtained by placing more emphasis on recall. Similarly, additional work also needs to be done to determine which factors strongly correlate to the success of this technique. It's possible that using larger population sizes or an increased number of generations could significantly improve the results obtained via this multi-tiered technique. Perhaps there is a way to determine which weighting scheme, population size, and number of generations are optimal for a given data set.

6.5 Lessons Learned

Similar to the subject matter of this dissertation, the researcher also evolved while conducting this research. Given a chance to start over, there are several things that would have been done differently. This section is included with future doctoral candidates in mind in order to pass on lessons learned from the research process, but which are not part of the work itself.

First, there was a great deal more data that could have been gathered. For example, "Arcanum Performance Graphs" such as Figure 6.1 should have been saved for each experiment performed. Similarly, collecting snapshots of the best specimen in each generation was added much later in the research. Consequently, a large amount of potentially interesting information was lost because it wasn't captured. When performing research, it is important to capture and store all available data – one never knows what the research might lead to, and without adequate data capture it is difficult to fully explore the results and the implications of the results.

Secondly, it is important that every decision made as part of the research can be justified with experimental data. For example, when choosing the weighting

scheme for Arcanum, both the 80/20 and the 70/30 weighting schemes were observed to be optimal for a data set. Further experimentation should have been performed before the choice was made, such as trying a 75/25 scheme, or testing the schemes on more data sets. Without the experimental results to verify a decision, the decision can be seen as arbitrary.

Lastly, many more experiments should have been run. For example, when choosing the weighting scheme, rather than running the experiments once on each of the sets, the experiments should have been performed multiple times on each of the sets, so that an average of the results could be obtained. In each case where results were obtained from Arcanum, multiple experiments should have been run. While a K-Fold Cross-Validation was used in the experiments, meaning that each was run multiple times, the experiments should have been repeated so that multiple K-Fold experiments could be averaged. In addition to running on each data set multiple times, all of the data sets should have been included in the hypothesis refinement section in order for the work to be complete.

However, there are real-world time constraints on how much experimentation can be done while still completing the research in a timely fashion. There is a constant conflict between time and quality of work, and sometimes trade-offs must be made. If the quality of the research will suffer because of time constraints, then the scope of the research should be reduced so that maximum quality can be achieved in the time allowed.

6.6 Final Remarks

The multi-tiered genetic algorithm technique described in this dissertation and the experiments performed using it, indicate that it has some potential in the realm of data mining and in the pursuit of finding patterns within complex data sets. The multi-tiered approach also has the potential to use genetic algorithms to explore data in more than two dimensions, perhaps opening more opportunities for the application of genetic algorithms to complex problems.

References

1. M. S. Arumugam and M.V.C. Rao. (2005). Novel Hybrid Approaches for Real Coded Genetic Algorithm to Compute the Optimal Control of a Single Stage Hybrid Manufacturing Systems. *International Journal of Computational Intelligence*, Vol. 1, No. 3, 189-206.

2. W.-H. Au, K.C.C. Chan, and X. Yao. (2003). A Novel Evolutionary Data Mining Algorithm with Applications to Churn Prediction. *IEEE Transactions of Evolutionary Computation*, Vol.7, No. 6, 532-545.

3. J. Bacardit and M.V. Butz. (2004). Data Mining in Learning Classifier Systems: Comparing XCS with GAssist. *Illinois Genetic Algorithms Laboratory*, IlliGAL Report No. 20040XX

4. J. Bacardit and J.M. Garrell. (2007). Bloat control and generalization pressure using the minimum description length principle for a pittsburgh approach learning classifier system. *Revised Selected Papers of the International Workshop on Learning Classifier Systems 2003-2005*, Lecture Notes in Computer Science, Springer, 59-79.

5. X.-W. Chen and C.M. Taylor. (2008) "Predicting Protein Function Using Sequence Information." Manuscript in preparation.

6. M.R. Chmielewski and J.W. Grzymala-Busse (1996). Global Discretization of Continuous Attributes as Preprocessing for Machine Learning. *International Journal of Approximate Reasoning*, Vol. 15, No. 4, 319-331.

7. S. Choenni. (2000). Design and Implementation of a Genetic-Based Algorithm for Data Mining. In *Proceedings of the 26th International Conference on Very Large Data Bases*, 33-42.

8. CiteSeer. (2008).What is Data Mining? http://citeseer.ist.psu.edu/69212.html.

9. L. De Raedt, H. Blockeel, L. Dehaspe, and W. Van Laer. (2001). Three companions for data mining in first order logic. In S. Dzeroski and N. Lavrac (Eds.) *Relational Data Mining*. Springer-Verlag, 105-139.

10. N. Dulay. (2008). Genetic Algorithms. http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/tcw2/report.html.

11. I.W. Flockhart and N.J. Radcliffe. (1996). A Genetic Algorithm-Based Approach to Data Mining. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, Portland, Oregon.

12. D. Goldberg. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley.

13. H.J. Hamilton. (2008). Confusion Matrix. http://www2.cs.uregina.ca/~hamilton/courses/831/notes/confusion_matrix/confusion_matrix.html.

14. S. Hettich and S.D. Bay. (1999). The *UCI KDD Archive*. Department of Information and Computer Science, University of California at Irvine, http://kdd.ics.uci.edu.

15. J. Holland. (1986). Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In *Machine Learning: An Artificial Intelligence Approach*, R. Michalski, J. Carbonell, and T. Mitchell, Eds. San Mateo, CA: Morgan Kaufmann.

16. T. Jech. (2002). Set Theory. In E.N. Zalta (ed.) *The Stanford Encyclopedia of Philosophy*, Fall 2002 Edition. http://plato.stanford.edu/archives/fall2002/entries/set-theory/.

17. R. Kohavi. (1995). The Power of Decision Tables. In Lavrae, N., & Wrobel, S. (Eds.), *Machine Learning* : ECML-95 : 8th European Conference on Machine Learning, Heraclion, Crete, Greece. Springer Verlag.

18. R. Kohavi and F. Provost. (1998). Glossary of Terms, Machine Learning, Special Issue on Applications of Machine Learning and the Knowledge Discovery Process, Vol. 30, No. 2-3.

19. J. Mendez, A. Falcon, and J. Lorenzo. (2003). A Procedure for Biological Sensitive Pattern Matching in Protein Sequences. *In Proceedings of the First Iberian Conference on Pattern Recognition and Image Analysis*, Mallorca, Spain, 547-555.

20. Z. Pawlak. (1994). Rough Sets Present State and Further Prospects. In *Proceedings of the Third International Workshop on Rough Set and Soft Computing*, San Jose, California, 72-76.

21. R. Quinlan. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Diego, California.

22. Z. Radivojevic, M. Cvetanovic, V. Milutinovic. (2003). Data Mining: A Brief Overview and Recent IPSI Research, *Annals of Mathematics, Computing, and Teleinformatics*, Vol. 1, No. 1, 84-91.

23. C. Ratanamahatana. (2008). CloNI: Clustering of √N-Interval Discretization. http://www.cs.ucr.edu/~ratana/CloNI.pdf.

24. G. Ridgeway and D. Madigan. (2002). A Sequential Monte Carlo Method for Bayesian Analysis of Massive Datasets. *Journal of Knowledge Discovery and Data Mining*, Vol. 7, 301-319.

25. J.C. Roden, B.W. King, D. Trout, A. Mortazavi, B.J. Wold, and C.E. Hart. (2006). Mining Gene Expression Data by Interpreting Principal Components. *BMC Bioinformatics*, Vol. 7, 194.

26. B. Rogers. (2008). Decision Tables Examples: Medical Insurance. http://faculty.sxu.edu/~rogers/sys/decision_tables.html.

27. K. Rosen. (2007). Discrete Mathematics and its Applications (6^{th} ed.). McGraw-Hill, New York, NY.

28. S.C. Shah and A. Kusiak. (2004). Data Mining and Genetic Algorithm Based Gene/SNP Selection. *Artificial Intelligence in Medicine*, Vol. 31, 183-196.

29. W. Siler. (2008). Rule-Based Reasoning: Antecedent and Consequent. http://members.aol.com/wsiler/chap03.htm.

30. S. Smith. (1983). Flexible learning of problem solving heuristics through adaptive search. In *Proc. 8th Int. Joint Conf. Artificial Intelligence*, Karlsruhe, Germany, pp. 422–425.

31. J. Souza, S. Matwin, and N. Japkowicz. (2002). Evaluating Data Mining Models: A Pattern Language. In *Proceedings of the Ninth Conference on Pattern Language of Programs*, Urbana, Illinois.

32. U. Straccia. (1998). A fuzzy description logic. In *Proceedings of the 15th National Conference on Artificial Intelligence*, Madison, Wisconsin, 594-599.

33. C.M. Taylor and A. Agah. (2006). Evolving Neural Network Topologies for Object Recognition. In Proceedings of the Sixth International Symposium on Soft Computing for Industry, World Automation Congress, Budapest, Hungary.

34. C.M. Taylor and A. Agah. (2008). Data Mining and Genetic Algorithms: Finding Hidden Meaning in Biological and Biomedical Data. *Computational Intelligence in Biology: Current Trends and Open Problems.* Smolinski, Tomasz G., Milanova, Mariofanna M., and Hassanien, Aboul-Ella (Eds.).

35. C.M. Taylor and A. Agah. (in review). Data mining using an enhanced genetic algorithm with direct manipulation of sets. Applied Intelligence.

36. Two Crows Corporation. (2008). Data Mining Glossary. http://www.twocrows.com/glossary.htm.

37. J. Valdes and G. Mateescu. (2002). Time series model mining with similarity-based neuro-fuzzy networks and genetic algorithms: a parallel implementation. *Proceedings of the RSCTC'02, Special Session on Distributed and Collaborative Data Mining*. Pennsylvania.

38. J. Vesanto. (1999). SOM-based data visualization methods. *Intelligent Data Analysis,* vol. 3.

39. S.M. Weiss and C.A. Kulikowski. (1991) *Computer Systems That Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning and Expert Systems*. Morgan Kaufmann, 1991.

40. What is Data Mining? (2005). Retrieved November 2005. http://citeseer.ist.psu.edu/69212.html.

41. Wikipedia. (2008). Rough set. Retrieved January 2009.

42. Wikipedia. (2008). Fuzzy set. Retrieved January 2009.

43. Wikipedia. (2008). Decision table. Retrieved January 2009.

44. G. Williams. (2008). Data Mining Desktop Survival Guide. http://www.togaware.com/datamining/survivor/Cross_Validation.html.

45. I.H. Witten and E. Frank. (2005). *Data Mining: Practical machine learning tools and techniques,* 2nd Edition, Morgan Kaufmann, San Francisco.

46. L.K. Woolery and J. Grzymala-Busse. (1994). Machine Learning for an Expert System to Predict Preterm Birth Risk. *Journal of the American Medical Informatics Association,* Vol. 1, No. 6, 439-446.

47. L. Yin, C.-H. Huang, and S. Rajasekaran. (2008). Parallel Data Mining of Bayesian Networks from Gene Expression Data. In Proceedings of the Eight International Conference on Research in Computational Molecular Biology.

Appendix 1. Arcanum Source Code

This appendix contains all of the source code for the Arcanum Project. Each subsection is the name of a module or class object, and contains the full source code listing for that module or class object.

A1.1 GlobalVariables

```
Global ComplimentsOkay As Boolean
Global Universe As SetClass
Global root As SuperSet
'Global rules As SuperSet
Global rules As Description
Global Decision() As Long
Global Decisions As SuperSet
Global atts() As Long
Global usedecision As Boolean
Global rweight As Single
Global pweight As Single
Global PopulationSize As Long
Global Population As Collection
Global otemp As SetClass
Global partitions As Collection
Global clocktime As Date
Global K As Long
Global TEMPPATH As String
Global BestPop As Long
Global MissingValues As Boolean
Global GenCount As Long
Global PrevPopFitness As Single
Global PrevBestFitness As Single
Global iFlag As Long
Global ruleindex As Long
Global rfile As Long
Global tFile As Long
Global txtRuleFile As String
Global bUseLnRoulette As Boolean
Global GenAvg As Long
Global TotCoverage As Single
Global TotAccuracy As Single
```

A1.2 LEMModule

```
Option Explicit
```

```
Sub SetControlVariables()
   msg "Setting up control variables..."
   If Main.chkCompliments.Value = 1 Then
      ComplimentsOkay = True
   Else
      ComplimentsOkay = False
   End If
   rweight = Val(Main.txtRecall)
   pweight = Val(Main.txtPrecision)
   PopulationSize = Val(Main.txtPopSize)
   If Main.optNoValidate.Value = True Then
      K = 0
      iFlag = 0
   Else
      If Main.optKFold.Value = True Then iFlag = 2
      If Main.optLOO.Value = True Then iFlag = 1
      K = Val(Main.txtK)
   End If
   GenCount = Val(Main.txtGenerations)
   txtRuleFile = Main.txtRuleFile.Text
   If Main.chkLog.Value = 1 Then
      bUseLnRoulette = True
   Else
      bUseLnRoulette = False
   End If
   GenAvg = 5 'Int(GenCount / 5)
   TEMPPATH = App.Path + "\Temp"
End Sub

Sub InduceRules()
   Dim d As SetClass
   Dim i As Long

   msg "Inducing rules..."
   Set d = New SetClass
   Set rules = New Description
'   While Decision(1) < root.Cardinality
   While Decisions.cardinality > 0
      GetNextDecision d
      msg "    " + d.attname + ", " + d.valname, True
      'Debug.Print d.attname + ", " + d.valname
      MLEM d
   Wend
```

```
    'msg "Gathering statistics...", True
    'GatherStatistics

    LinearDropping

    msg "Coverage: " + Format(rules.Coverage, "0.00"), True

End Sub

Sub PrintRules(Optional txtfile As String, Optional count As Boolean, Optional
Coverage As Single, Optional Accuracy As Single)
    Dim i As Long
    Dim ifile As Long
    Dim buf As String
    If txtfile = "" Then
        For i = 1 To rules.cardinality
            buf = "Precision: " + Format(rules.Element(i).Precision, "percent")
            buf = buf + ";  Recall: " + Format(rules.Element(i).Recall, "percent")
            If count Then
                buf = buf + ";  Count: " + Str(rules.Element(i).count)
            End If
            'Debug.Print buf
            buf = rules.Element(i).clause + " -> "
            buf = buf + "(" + rules.Element(i).Decision.attname + "," +
rules.Element(i).Decision.valname + ")"
            'Debug.Print buf
            'Debug.Print
        Next
    Else
        ifile = FreeFile
        Open txtfile For Output As ifile
        If Accuracy <> 0 Or Coverage <> 0 Then
            Print #ifile, "Accuracy = " + Format(Accuracy, "percent")
            Print #ifile, "Coverage = " + Format(Coverage, "percent")
            Print #ifile, vbNewLine
        End If
        For i = 1 To rules.cardinality
            buf = "Precision: " + Format(rules.Element(i).Precision, "percent")
            buf = buf + ";  Recall: " + Format(rules.Element(i).Recall, "percent")
            If count Then
                buf = buf + ";  Count: " + Str(rules.Element(i).count)
            End If
            Print #ifile, buf
            buf = rules.Element(i).clause + " -> "
```

```
            buf = buf + "(" + rules.Element(i).Decision.attname + "," +
rules.Element(i).Decision.valname + ")"
            Print #ifile, buf
            Print #ifile, ""
        Next
        Close ifile
    End If
End Sub

Sub BuildSetsFromFile(ByVal txtfile As String, blnRemoveUniversals)
    On Error GoTo BuildSetsFromFile_Error
    Dim ifile As Long
    Dim buf As String * 50000
    Dim attnames As Collection
    Dim attvals As Collection
    Dim att As SetClass
    Dim example
    Dim i As Long
    Dim x As Long
    Dim attcount As Long

    msg "Building sets..."
    MissingValues = False
    usedecision = False
    example = 0
    Set Decisions = New SuperSet
    Set root = New SuperSet
    Set Universe = New SetClass
        Universe.attname = "Universe"
        Universe.valname = "All Examples"
    Set attnames = New Collection
    Set attvals = New Collection
    ifile = FreeFile
    Open txtfile For Input As ifile
    Line Input #ifile, buf
    ParseLine buf, attnames
    If attnames.count = 0 Then
        Line Input #ifile, buf
        ParseLine buf, attnames
    End If
    While Not EOF(ifile)
        example = example + 1
        Line Input #ifile, buf
        'Debug.Print Trim(buf)
```

```
    ParseLine buf, attvals
    attcount = attvals.count
    For i = 1 To attcount
       If attvals(i) = "*" Or attvals(i) = "?" Then
          MissingValues = True
       End If
       root.AddToSet attnames(i), attvals(i), example
       Universe.AddToSet example
       x = root.Contains(attnames(i), attvals(i))
       If usedecision Then
          If atts(i) = 1 Then
             root.Element(x).Decision = True
          Else
             root.Element(x).Decision = False
          End If
       Else
          root.Element(x).Decision = True
       End If
       If root.Element(x).Decision = True Then
          Decisions.InsertSet root.Element(x)
       End If
    Next
Wend
ReDim Decision(root.cardinality) As Long
'Line Input #iFile, buf
'ParseLine buf, attnames

Close ifile

CreateAdditionalSets

If MissingValues Then
    HandleMissingValues
End If

If blnRemoveUniversals = True Then
    If Universe.cardinality > 1 Then
       RemoveUniversalAttributes
    End If
End If

root.FindRelevantSets

'DisplaySets
```

```
BuildSetsFromFile_Exit:
   Exit Sub

BuildSetsFromFile_Error:
   ErrorHandler "LEMmodule::BuildSetsFromFile", Err
   GoTo BuildSetsFromFile_Exit

End Sub

Sub ParseLine(ByVal abuf As String, attnames As Collection)
   Dim i As Long
   Dim buf As String
   Dim word As String
   Dim wordon As Boolean

   buf = Trim(abuf)
   i = 1
   word = ""
   Set attnames = New Collection
   wordon = False
   While i <= Len(buf)
      Select Case Mid(buf, i, 1)
         Case " ", "," 'end of word
            wordon = False
         Case "<" 'starts a line indicating decisions
            FindDecisions buf
            Exit Sub
         Case "[" 'start of attribute line...new word
            wordon = True
            i = i + 1
            While Mid(buf, i, 1) = " "
               i = i + 1
            Wend
            i = i - 1
         Case "]" 'end of attribute line...end word
            wordon = False
         Case Else
            wordon = True
            word = word + Mid(buf, i, 1)
      End Select
      If Not wordon And Len(word) > 0 Then
         attnames.Add word
         word = ""
      ElseIf i = Len(buf) Then
```

```
            attnames.Add word
            word = ""
        End If
        i = i + 1
    Wend
End Sub

Sub GetNextDecision(d As SetClass)
    Dim i As Long
    Dim ThisOne As Boolean

    Set d = New SetClass

'    Debug.Print Decisions.Cardinality
    Set d = Decisions.Element(1)
    Decisions.RemoveMember Decisions.Element(1)
'    Debug.Print Decisions.Cardinality
'    If Not usedecision Then
'        Decision(1) = Decision(1) + 1
'        If Decision(1) <= root.Cardinality Then
'            d.Union root.Element(Decision(1))
'            d.attname = root.Element(Decision(1)).attname
'            d.valname = root.Element(Decision(1)).valname
'        End If
'    Else
'        ThisOne = False
'        'Decision(1) = Decision(1) + 1
'        While Decision(1) < root.Cardinality And Not ThisOne
'            Decision(1) = Decision(1) + 1
'            ThisOne = root.Element(Decision(1)).Decision
'        Wend
'        If Decision(1) <= root.Cardinality Then
'            d.Union root.Element(Decision(1))
'            d.attname = root.Element(Decision(1)).attname
'            d.valname = root.Element(Decision(1)).valname
'        End If
'    End If
End Sub

Sub ConvertNumber(x As Long)
    Decision(x) = Decision(x) + 1
    If Decision(x) > root.cardinality - (UBound(Decision) - x) Then
        Decision(x) = 1
        ConvertNumber x - 1
```

```
    End If
End Sub

Sub MLEM(ByVal d As SetClass)
    On Error GoTo MLEM_Error
    Dim rule As RuleClass
    Dim rel As SetClass
    Dim setlist As SuperSet
    Dim Decision As SetClass

    Set Decision = New SetClass
    Decision.Union d
    Decision.attname = d.attname
    Decision.valname = d.valname
    Set setlist = New SuperSet
    setlist.Union root
    setlist.RemoveMember d
    Set rel = New SetClass
    While d.cardinality > 0
        'Find the set that has the most in common, but isn't the same set
        Set rule = New RuleClass
        FindMostRelevant d, rel, setlist
        rule.Decision = Decision
        rule.AddToRule rel, "and"
        'rule.attname = "->(" + d.attname + "," + d.valname + ")"
        'rule.valname = rule.valname + "(" + rel.attname + "," + rel.valname + ")"
        'rule.Union rel
        rule.ReEvaluate
        setlist.RemoveMember rel
        While Not d.Rulesubset(rule) And rel.attname <> ""
            FindMostRelevant d, rel, setlist
            If rel.attname <> "" Then
                rule.AddToRule rel, "and"
                rule.ReEvaluate
                'rule.Intersect rel
                'rule.valname = rule.valname + "&(" + rel.attname + "," + rel.valname + ")"
                setlist.RemoveMember rel
            End If
        Wend
        If rel.attname <> "" Then
            rules.AddRule rule
            d.Difference rule.SetFromRule
        Else
            d.Difference d
```

```
      End If
   Wend
MLEM_Exit:
   Exit Sub

MLEM_Error:
   ErrorHandler "LEMmodule::MLEM", Err
   GoTo MLEM_Exit
End Sub

Sub FindMostRelevant(d As SetClass, rel As SetClass, setlist As SuperSet)
   Dim S As SetClass
   Dim i As Long

   Set rel = New SetClass
   For i = 1 To setlist.cardinality
      Set S = New SetClass
      S.Union setlist.Element(i)
      S.attname = setlist.Element(i).attname
      S.valname = setlist.Element(i).valname
      S.Intersect d
      If S.cardinality > rel.cardinality Then
         Set rel = S
      End If
   Next

      i = root.Contains(rel.attname, rel.valname)
      If i <> 0 Then
         Set rel = root.Element(i)
         setlist.RemoveMember rel
      Else
      End If

End Sub

Sub CreateAdditionalSets()
   On Error GoTo CreateAdditionalSets_Error
   Dim i As Long
   Dim j As Long
   Dim slist As SuperSet
   Dim sets As Collection
   Dim otemp As SetClass

   Set sets = New Collection
```

```
    Set slist = New SuperSet
    slist.InsertSet root.Element(1)
    For i = 2 To root.cardinality
        If root.Element(i).attname = slist.Element(1).attname Then
            slist.InsertSet root.Element(i)
        Else
            sets.Add slist
            Set slist = New SuperSet
            slist.InsertSet root.Element(i)
        End If
    Next
    sets.Add slist

    For i = 1 To sets.count
        If IsNumeric(sets(i).Element(1).valname) Then
            PartitionSet sets(i)
        Else
            If ComplimentsOkay Then
                ComplimentSet sets(i)
            End If
        End If
    Next

    'DisplaySets
CreateAdditionalSets_Exit:
    Exit Sub

CreateAdditionalSets_Error:
    ErrorHandler "LEMmodule::CreateAdditionalSets", Err
    GoTo CreateAdditionalSets_Exit
End Sub

Sub PartitionSet(S As SuperSet)
    On Error GoTo PartitionSet_Error
    Dim i As Long
    Dim x As Long
    Dim part As SetClass

    S.Sort True
    For x = 2 To S.cardinality - 1
        Set part = New SetClass
        part.attname = S.Element(1).attname
        part.valname = S.Element(1).valname
        For i = 1 To x
```

```
            part.Union S.Element(i)
        Next
        part.valname = part.valname + ".." + S.Element(i - 1).valname
        root.InsertSet part
        Set part = New SetClass
        part.attname = S.Element(1).attname
        part.valname = S.Element(x + 1).valname
        For i = x + 1 To S.cardinality
            part.Union S.Element(i)
        Next
        If part.valname <> S.Element(i - 1).valname Then
            part.valname = part.valname + ".." + S.Element(i - 1).valname
            root.InsertSet part
        End If
    Next
    Set part = New SetClass
    part.attname = S.Element(1).attname
    If S.cardinality < 2 Then
        part.valname = S.Element(1).valname
    Else
        part.valname = S.Element(2).valname
    End If
    For i = 2 To S.cardinality
        part.Union S.Element(i)
    Next
    part.valname = part.valname + ".." + S.Element(i - 1).valname
    root.InsertSet part
PartitionSet_Exit:
    Exit Sub

PartitionSet_Error:
    ErrorHandler "LEMmodule::PartitionSet", Err
    GoTo PartitionSet_Exit
End Sub

Sub ComplimentSet(S As SuperSet)
    Dim i As Long

    If S.cardinality > 2 Then
        For i = 1 To S.cardinality
          If Not usedecision Or S.Element(i).Decision = False Or Not
S.Element(i).valname = "?" Then
            Set otemp = New SetClass
            otemp.Union S.Element(i)
```

```
            otemp.attname = S.Element(i).attname
            otemp.valname = S.Element(i).valname
            otemp.Compliment
            root.InsertSet otemp
            'Set otemp = Nothing
          End If
        Next
    End If
End Sub

Sub DisplaySets(Optional iShowElements As Boolean)
    Dim i As Long
    Dim x As Long
    Dim buf As String

    For i = 1 To root.cardinality
        If root.Element(i).Decision Then
            Debug.Print root.Element(i).attname + ", " + root.Element(i).valname + ";
decision=TRUE"
        Else
            Debug.Print root.Element(i).attname + ", " + root.Element(i).valname + ";
decision=FALSE"
        End If
        If iShowElements Then
            buf = ""
            For x = 1 To root.Element(i).cardinality
                buf = buf + Str(root.Element(i).Element(x)) + ", "
            Next
            buf = Left(buf, Len(buf) - 2)
            Debug.Print vbTab + buf
        End If
    Next
End Sub

Sub GatherStatistics()
    Dim i As Long
    Dim pos As Long
    Dim Coverage As Single
    Dim general As Long
    Dim specific As Long
    Dim attname As String
    Dim valname As String

    For i = 1 To rules.cardinality
```

```
      GetNames rules.Element(i).attname, attname, valname
      pos = root.Contains(attname, valname)
      general = root.Element(pos).cardinality
      specific = rules.Element(i).cardinality
      Coverage = specific / general
      rules.Element(i).stats = Str(specific) + "/" + Str(general) + " = " +
Format(specific / general, "percent")
   Next
End Sub

Sub GetNames(buf As String, attname As String, valname As String)
   Dim i As Long

   i = 1
   buf = Right(buf, Len(buf) - 3)
   buf = Left(buf, Len(buf) - 1)
   While Mid(buf, i, 1) <> ","
      i = i + 1
   Wend
   attname = Left(buf, i - 1)
   valname = Right(buf, Len(buf) - i)
End Sub

Sub FindDecisions(buf As String)
   Dim i As Long
   Dim x As Long

   i = 0
   x = 0
   While i < Len(buf)
      i = i + 1
      If Mid(buf, i, 1) = "a" Then
         x = x + 1
         ReDim Preserve atts(x) As Long
         atts(x) = 0
      ElseIf Mid(buf, i, 1) = "d" Then
         x = x + 1
         ReDim Preserve atts(x) As Long
         atts(x) = 1
      End If
   Wend

   usedecision = True
End Sub
```

```
Sub RemoveUniversalAttributes()
   Dim i As Long
   Dim obj As SetClass

   i = 0
   While i < root.cardinality
      i = i + 1
      If root.Element(i).Equal(Universe) Then
         Set obj = root.Element(i)
         root.RemoveMember obj
         i = i - 1
      End If
   Wend
End Sub

Sub msg(S As String, Optional append As Boolean)
   If append = True Then
      Main.txtOut = Main.txtOut + vbNewLine + S
   Else
      Main.txtOut = S
   End If
   DoEvents
End Sub

Sub ReduceRule(rule As RuleClass)
   On Error GoTo ReduceRule_Error
   Dim i As Long
   Dim j As Long
   Dim target As RuleClass
   Dim mark As RuleClass
   Dim mark_temp As SetClass
   Dim target_temp As SetClass

   Set target = rule.CopyRule
   Set mark = target.CopyRule
   j = mark.RuleSize
   'Debug.Print target.Decision.valname + " -> " + target.clause
   While j > 0
      mark.RemoveFromRule j
      mark.ReEvaluate
      Set mark_temp = mark.SetFromRule
      Set target_temp = target.SetFromRule
```

```
        If ((target.cardinality = mark.cardinality) Or
(rule.Decision.SubSet(mark_temp))) And target.cardinality > 0 Then
            Set target = mark.CopyRule
        End If
        j = j - 2
        If target.cardinality > 0 Then
            Set mark = target.CopyRule
        End If
        'Debug.Print mark.Decision.valname + " -> " + mark.clause
    Wend
    Set rule = target.CopyRule
ReduceRule_Exit:
    Exit Sub

ReduceRule_Error:
    ErrorHandler "LEMmodule::ReduceRule", Err
    GoTo ReduceRule_Exit
End Sub

Sub LinearDropping()
    Dim rule As RuleClass
    Dim r As RuleClass
    Dim i As Long
    Dim j As Long

    Set r = New RuleClass
    msg "Performing linear dropping...", True
    For i = 1 To rules.cardinality
        Set rule = rules.Element(i)
        Set r = rule.CopyRule
        j = r.RuleSize
        Debug.Print rule.Decision.valname + " -> " + rule.clause
        While r.cardinality = rule.cardinality And j > 0
            r.RemoveFromRule j
            Debug.Print "    " + r.clause
            If rule.Decision.SubSet(r.SetFromRule) Then
                rule.RemoveFromRule j
                Set r = rule.CopyRule
                j = r.RuleSize
            Else
                Set r = rule.CopyRule
                j = j - 2
            End If
           'Set r = rule.CopyRule
```

```
        'j = j - 2
      Wend
      Debug.Print rule.Decision.valname + " -> " + rule.clause
      ReduceRule rules.Element(i)
   Next
End Sub

Sub HandleMissingValues()
   On Error GoTo HandleMissingValues_Error
   Dim i As Long
   Dim j As Long

   Debug.Print "Missing Values"
   i = 0
   While i < root.cardinality
      i = i + 1
      If root.Element(i).valname = "?" Then
         root.RemoveMember root.Element(i)
         i = i - 1
      End If
      If i < 1 Then i = 1
      If root.Element(i).valname = "*" Then
         For j = 1 To root.cardinality
            If root.Element(j).attname = root.Element(i).attname And
root.Element(j).attname <> "*" Then
               root.Element(j).Union root.Element(i)
            End If
         Next
         root.RemoveMember root.Element(i)
         i = i - 1
      End If
   Wend
HandleMissingValues_Exit:
   Exit Sub

HandleMissingValues_Error:
   ErrorHandler "LEMmodule::HandleMissingValues", Err
   GoTo HandleMissingValues_Exit
End Sub
```

A1.3 ArcanumGA

```
Sub Arcanum()
   On Error GoTo Arcanum_Error
```

```
    Dim BestSpecimen As Description
    Dim Parent1 As Description
    Dim Parent2 As Description
    Dim Child As Description
    Dim NewPopulation As Collection
    Dim i As Long
    Dim x As Long
    Dim generation As Long
    Dim Fitness As Collection
    Dim avgfitness As Single
    Dim sum As Single

    avgfitness = 0
    Set Fitness = New Collection
    InitializeGraph
    CreateInitialPopulation txtRuleFile
    'DisplayAllDescriptions
    'set best member to BestSpecimen
    Set BestSpecimen = Population(1)
    BestPop = 0
    GetBestSpecimen Population, BestSpecimen, 1
    Fitness.Add Int(PopFitness * 100) / 100
    PlotFitness generation, Fitness(1), BestSpecimen.Fitness
    'while stop criteria not met
    For generation = 1 To GenCount
        msg "Population " + Str(generation) + " :: Fitness " + Str(Fitness(Fitness.count))
+ " :: BestSpecimen " + Str(BestSpecimen.Fitness), True
        Set NewPopulation = New Collection

'*********************************************************************
    '3/20 :: moved mutation to be before crossover
    '       to fix, cut and paste before Set Population = NewPopulation
    '       also need to change from Population to NewPopulation
    'Perform genetic operations on each member of the population
        'PerformGeneticOperations NewPopulation
        PerformGeneticOperations Population

'*********************************************************************
*
    'Add the BestSpecimen to the new population, thus preserving it
        Set Child = New Description
        For i = 1 To BestSpecimen.Cardinality
            Child.AddRule BestSpecimen.Element(i).CopyRule
        Next
```

```
      NewPopulation.Add Child
   'Create a new generation by mating parents together and forming 2 children
      While NewPopulation.count < PopulationSize - 1
         SelectParents Population, Parent1, Parent2
         For i = 1 To 2  'Create 2 children from the same parents
            Crossover Parent1, Parent2, Child
            NewPopulation.Add Child
         Next
      Wend
      SelectParents Population, Parent1, Parent2
      Crossover Parent1, Parent2, Child
      NewPopulation.Add Child
      Set Population = NewPopulation
   'if best member better than BestSpecimen, then replace BestSpecimen with the new
best
      'Debug.Print "   selecting best specimen..."
      GetBestSpecimen Population, BestSpecimen, generation
   'stop if average fitness of the new generation is less than the average of previous N
generations
      If Fitness.count >= GenAvg Then
         sum = 0
         For i = 1 To Fitness.count
            sum = sum + Fitness(i)
         Next
         avgfitness = Int((sum / Fitness.count) * 100) / 100
         Fitness.Remove 1
      End If
      Fitness.Add Int(PopFitness * 100) / 100
      PlotFitness generation, Fitness(Fitness.count), BestSpecimen.Fitness
      'Test to see if at least GenAvg generations have passed
      If Fitness.count >= GenAvg Then
         'If the fitness of the new population is equal to the average fitness
         'of the previous five generations, then stop.
         'Debug.Print Str(Fitness(Fitness.count)) + " ; AVG = " + Str(avgfitness)
         If Fitness(Fitness.count) < avgfitness Then
            generation = GenCount + 1
         End If
      End If
   'wend
   Next
   'Debug.Print "Population " + Str(i) + " :: Fitness " + Str(PopFitness)
   'DisplayDescription BestSpecimen, "Best Fitness " + Str(BestSpecimen.Fitness) +
" ; occurred in Generation " + Str(BestPop)
   PostProcess BestSpecimen
```

```
    'Debug.Print "...after post-processing..."
    'DisplayDescription BestSpecimen, "Best Fitness " + Str(BestSpecimen.Fitness) +
" ; occurred in Generation " + Str(BestPop)
    Set rules = BestSpecimen

    'MsgBox ("Done!")
Arcanum_Exit:
    Exit Sub

Arcanum_Error:
    ErrorHandler "ArcanumGA::Arcanum", err
    GoTo Arcanum_Exit
End Sub
Function GetBestSpecimen(Population As Collection, BestSpecimen As Description,
generation As Long) As Long
    Dim i As Long
    Dim j As Long
    Dim best As Long
    For i = 1 To Population.count
        If Population(i).Fitness > BestSpecimen.Fitness Then
            Set BestSpecimen = New Description
            For j = 1 To Population(i).Cardinality
                BestSpecimen.AddRule Population(i).Element(j).CopyRule
            Next
            BestPop = generation
            'Debug.Print "     New Best Specimen in Generation " + Str(generation)
        End If
    Next
    BestSpecimen.ReEvaluate
End Function

Sub Crossover(Parent1 As Description, Parent2 As Description, Child As
Description)
    Set Child = New Description
    GetRulesFromParent Parent1, Child
    GetRulesFromParent Parent2, Child
    Child.ReEvaluate
End Sub

Sub GetRulesFromParent(ByVal P As Description, Child As Description)
    Dim StopPoint As Long
    Dim i As Long
    Dim x As Long
    Dim sum As Single
```

```
    Dim selectfit As Single
    Dim Parent As Description

    Set Parent = New Description
    For x = 1 To P.Cardinality
       Parent.AddRule P.Element(x).CopyRule
    Next

'   DisplayDescription Parent, "Parent"
    StopPoint = Int((Parent.Cardinality + 1) / 2)
    For i = 1 To StopPoint
       sum = 0
       For x = 1 To Parent.Cardinality
        If Parent.Element(x).Fitness <> 0 Then
          If bUseLnRoulette Then
             sum = sum + Log(Parent.Element(x).Fitness)
          Else
             sum = sum + Parent.Element(x).Fitness
          End If
        End If
       Next
       Randomize
       sum = Rnd(1) * sum
       x = 1
       While sum > 0
        If Parent.Element(x).Fitness > 0 Then
          If bUseLnRoulette Then
             sum = sum - Log(Parent.Element(x).Fitness)
          Else
             sum = sum - Parent.Element(x).Fitness
          End If
        End If
          If sum > 0 Then
             x = x + 1
          End If
       Wend
       Child.AddRule Parent.Element(x).CopyRule
       Parent.DeleteRule x
    Next
'   DisplayDescription Child, "Child"
End Sub

Sub SelectParents(ByVal Pop As Collection, Parent1 As Description, Parent2 As
Description)
```

```
    Dim i As Long
    Dim x As Long
    Dim sum As Single
    Dim Population As Collection

    Set Population = New Collection
    For i = 1 To Pop.count
       Population.Add Pop(i)
    Next

    For i = 1 To 2
       sum = 0
       For x = 1 To Population.count
         If Population(x).Fitness > 0 Then
           If bUseLnRoulette Then
              sum = sum + Log(Population(x).Fitness)
           Else
              sum = sum + Population(x).Fitness
           End If
         End If
       Next
       Randomize
       sum = Rnd(1) * sum
       x = 1
       While sum > 0
         If Population(x).Fitness > 0 Then
           If bUseLnRoulette Then
              sum = sum - Log(Population(x).Fitness)
           Else
              sum = sum - (Population(x).Fitness)
           End If
         End If
           If sum > 0 Then
              x = x + 1
           End If
       Wend
       If i = 1 Then
          Set Parent1 = Population(x)
       Else
          Set Parent2 = Population(x)
       End If
       Population.Remove x
    Next
End Sub
```

```
Function PopFitness() As Single
   Dim sum As Single
   Dim i As Long

   sum = 0
   For i = 1 To Population.count
      sum = sum + (Population(i).Accuracy * pweight) + (Population(i).Coverage *
rweight)
   Next
   PopFitness = sum / i
End Function
Function Fitness(S As Collection) As Single
   Dim i As Long
   Dim Recall As Single
   Dim Precision As Single

   If rweight + pweight <> 1 Then
      MsgBox ("Invalid Recall and Precision Weights")
      Exit Function
   End If
   Recall = 0
   Precision = 0

   For i = 1 To S.count
      Recall = Recall + S(i).Recall
      Precision = Precision + S(i).Precision
   Next
   Recall = Recall / S.count
   Precision = Precision / S.count
   Fitness = Recall * rweight + Precision * pweight
End Function

Sub CreateInitialPopulation(Optional txtRuleFile As String)
   Dim i As Long
   Dim r As Description
   Dim allrules As Collection
   Dim rules As Collection
   Dim nr As RuleClass
   Dim idx As Long
   Dim rule_desc As Description

   Set rule_desc = New Description
   Set Population = New Collection
```

```
    If txtRuleFile = "" Then
'       MsgBox ("No Rule File")
        For i = 1 To PopulationSize
            Set r = New Description
            CreateRuleSet r
            Population.Add r
        Next
    Else
        'Make sure the rules file is valid.
        err = 0
        rfile = FreeFile
        Open txtRuleFile For Input As rfile
        If err <> 0 Then
            MsgBox txtRuleFile + " is not a valid file.  Process aborted."
            err = 0
            End
        End If
        'Read the rules from the file
        Set allrules = New Collection
        Set rules = New Collection
        While Not EOF(rfile)
            GetNextRule rules
            'allrules.Add rules
          If rules.count > 0 Then
            Set nr = New RuleClass
            idx = root.Contains(rules(rules.count - 1), rules(rules.count))
            If idx > 0 Then
                nr.Decision = root.Element(idx)
                idx = root.Contains(rules(1), rules(2))
                If idx > 0 Then
                    nr.AddToRule root.Element(idx), "and"
                End If
                For i = 4 To rules.count - 3 Step 3
                    idx = root.Contains(rules(i), rules(i + 1))
                    If idx > 0 Then
                        nr.AddToRule root.Element(idx), rules(i - 1)
                    Else
                        Set nr = New RuleClass
                        i = rules.count
                    End If
                Next
                nr.ReEvaluate
            Else
                i = rules.count
```

```
            End If
            If i <= rules.count And nr.Clause <> "" Then
               rule_desc.AddRule nr
            End If
          End If
        Wend
        'The description is built.  Copy to 2/3 of the population.
        For i = 1 To Int(PopulationSize / 3)
           Population.Add rule_desc
        Next
        'Build new descriptions for the remaining 1/3 of the population.
        For i = Int(PopulationSize / 3) + 1 To PopulationSize
           Set r = New Description
           CreateRuleSet r
           Population.Add r
        Next
    End If
    For i = 1 To PopulationSize
       Population(i).ReEvaluate
    Next
    'DisplayAllDescriptions
End Sub

Sub CreateRuleSet(r As Description)
    Dim i As Long
    Dim rc As RuleClass
    Dim x As Long
    Dim roottemp As SuperSet
    Dim Z As Long

    For i = 1 To Decisions.Cardinality
       Set rc = New RuleClass
       rc.Decision = Decisions.Element(i)
       Z = root.Contains(rc.Decision.attname, rc.Decision.valname)
       Set roottemp = root.RelevantSets(Z)
       x = Int(Random(1, roottemp.Cardinality))
       While (roottemp.Element(x).attname = rc.Decision.attname) And
(roottemp.Element(x).valname = rc.Decision.valname)
          x = Int(Random(1, roottemp.Cardinality))
       Wend
       rc.AddToRule roottemp.Element(x)
       rc.ReEvaluate
       r.AddRule rc
    Next
```

```
End Sub

Function Random(min As Long, max As Long) As Single
   Randomize
   If max = 1 Then
      Random = Rnd()
   Else
      Random = Rnd() * (max - min + 1) + min
   End If
End Function

Sub DisplayAllDescriptions()
   Dim i As Long
   Dim j As Long
   Dim obj As RuleClass
   Dim buf As String

   For i = 1 To Population.count
      Debug.Print "Description " + Str(i) + " :: Fitness " + Str(Population(i).Fitness)
      DisplayDescription Population(i)
   Next
   Debug.Print ":: Population Fitness " + Str(PopFitness)
End Sub

Sub DisplayDescription(d As Description, Optional tag As String)
   Dim j As Long
   Dim obj As RuleClass
   Dim buf As String

   Debug.Print tag
   For j = 1 To d.Cardinality
      Set obj = d.Element(j)
      buf = "     " + obj.Clause
      buf = buf + " -> " + obj.Decision.attname + "," + obj.Decision.valname
      buf = buf + "  ::  P=" + Str(obj.Precision) + " ; R=" + Str(obj.Recall)
      Debug.Print buf
   Next
End Sub

Sub PerformGeneticOperations(Population As Collection)
   Dim i As Long
   Dim j As Long
   Dim x As Long
   Dim chance As Single
```

```
    Dim obj As RuleClass
    Dim newset As SetClass
    Dim objtemp As SetClass
    Dim sum As Single
    Dim okay As Boolean
    Dim sNoChange As Single
    Dim sChangeoperator As Single
    Dim sChangeSet As Single
    Dim sSimplifyRule As Single
    Dim sComplicateRule As Single
    Dim sAddRule As Single
    Dim rulelength As Long

    For i = 1 To Population.count
       For j = 1 To Population(i).Cardinality
          sum = 0
          Set obj = Population(i).Element(j)
       'Set the percentage chance for each mutation to be selected
          sNoChange = (obj.Precision * pweight + obj.Recall * rweight)
          'sChangeOperator = 1 - obj.Precision
          sChangeoperator = 1 - (obj.Precision * pweight + obj.Recall * rweight)
          'sChangeSet = 1 - obj.Precision
          sChangeSet = 1 - (obj.Precision * pweight + obj.Recall * rweight)
          'sSimplifyRule = (1 - obj.Recall) * rweight
          sSimplifyRule = 1 - (obj.Precision * pweight + obj.Recall * rweight)
          sComplicateRule = (1 - obj.Precision) '* pweight
          sAddRule = (1 - Population(i).Coverage) '* rweight

          'sum = sum + (3 * (1 - obj.Precision)) + (1 - obj.Recall)
          sum = sum + sNoChange + sChangeoperator + sChangeSet + sSimplifyRule +
sComplicateRule

          Randomize
          chance = Rnd(1) * sum
       '*** ChangeOperator
**********************************************************************
********************
          If chance <= sChangeoperator Then
             'Debug.Print "      ChangeOperator" + "  " + obj.clause
             chance = Int(Rnd(1) * (obj.olist.count / 2 + 1))
             If chance > 0 And (chance * 2) <= obj.olist.count Then
                rulelength = obj.olist.count
                sum = Int(Rnd(1) * 3) + 1
                Select Case sum
```

```
            Case 1
               obj.olist.Add "AND", , chance * 2
               obj.olist.Remove chance * 2 + 1
            Case 2
               obj.olist.Add "OR", , chance * 2
               obj.olist.Remove chance * 2 + 1
            Case 3
               obj.olist.Add "XOR", , chance * 2
               obj.olist.Remove chance * 2 + 1
            Case Else
               Debug.Print "Something wrong with ChangeOperator!!"
            End Select
            If rulelength <> obj.olist.count Then
               MsgBox ("Oops!  ChangeOperator malfunction!")
            End If
         End If

         chance = 1000
         obj.ReEvaluate
      Else
         'chance = chance - ((1 - obj.Precision))
         chance = chance - sChangeoperator
      End If
   '*** ChangeSet
************************************************************************
************************
      If chance <= sChangeSet Then
         'Debug.Print "     ChangeSet" + "  " + obj.clause
         Dim roottemp As SuperSet
         Set roottemp = New SuperSet

         rulelength = obj.olist.count
         root.Contains obj.Decision.attname, obj.Decision.valname, x
         'Debug.Print root.Element(x).attname + ", " + root.Element(x).valname
         'RootTemp.Union root.RelevantSets(x)
         Set roottemp = root.RelevantSets(x)
         'For x = 1 To obj.olist.count Step 2
         '   RootTemp.RemoveMember obj.olist(x)
         'Next
         Randomize
         Set newset = obj.Decision
         While newset.attname = obj.Decision.attname And newset.valname =
obj.Decision.valname And roottemp.Cardinality > 1
            x = Int(Rnd(1) * roottemp.Cardinality) + 1
```

```
            Set newset = roottemp.Element(x)
          Wend
          x = Int(Rnd(1) * (obj.olist.count / 2)) + 1
          obj.olist.Add newset, , (x - 1) * 2 + 1
          obj.olist.Remove (x - 1) * 2 + 2
          'Set obj.olist(x) = newset
          If rulelength <> obj.olist.count Then
            MsgBox ("Oops!  ChangeSet malfunction!")
          End If
          chance = 1000
          obj.ReEvaluate
        Else
          'chance = chance - ((1 - obj.Precision))
          chance = chance - sChangeSet
        End If
      '*** SimplifyRule
*************************************************************************
**********************
        If chance <= sSimplifyRule Then
          'Debug.Print "      SimplifyRule" + "  " + obj.clause
          'x = obj.olist.count - 1
          'okay = True
          'While x > 1 And okay
          '    If obj.olist(x) <> "AND" Then
          '        x = x - 2
          '    Else
          '        okay = False
          '    End If
          'Wend
          'If x > 1 Then
          '    obj.olist.Remove x
          '    obj.olist.Remove x
          'End If
          If obj.olist.count > 1 Then
            x = Int(Rnd(1) * (obj.olist.count / 2)) + 2
            While obj.olist.count > (x * 2 - 3)
              obj.olist.Remove obj.olist.count
            Wend
          End If

          chance = 1000
          obj.ReEvaluate
        Else
          'chance = chance - (1 - obj.Recall)
```

```
            chance = chance - sSimplifyRule
         End If
      '*** ComplicateRule
**********************************************************************
********************
         If chance <= sComplicateRule Then
            rulelength = obj.olist.count
            'Debug.Print "      ComplicateRule" + "  " + obj.clause
            Set newset = SelectSet(obj)
            If newset.attname <> obj.Decision.attname And newset.valname <>
obj.Decision.valname Then
               obj.AddToRule newset, "AND"
               obj.ReEvaluate
            End If
            If rulelength < obj.olist.count - 2 Then
               MsgBox ("Oops!  ComplicateRule malfunction!")
            End If
            chance = 1000
            obj.ReEvaluate
         Else
            'chance = chance - (1 - obj.Precision)
            chance = chance - sComplicateRule
         End If
         'If chance <= sNoChange Then
         '    MsgBox ("No change to this rule.")
         'End If
      Next
      '*** AddRule
**********************************************************************
**************************
      chance = Rnd(1)
      If chance <= sAddRule Then
         'Debug.Print "      AddRule to Description " + Str(i)
         Population(i).AddRule NewRule
      End If
      Population(i).ReEvaluate
   Next
End Sub

Function SelectSet(r As RuleClass) As SetClass
   Dim i As Long
   Dim x As Long
   Dim notOkay As Boolean
   Dim roottemp As SuperSet
```

```
    Set roottemp = New SuperSet
    root.Contains r.Decision.attname, r.Decision.valname, x
    'Debug.Print root.Element(x).attname + ", " + root.Element(x).valname
    'RootTemp.Union root.RelevantSets(x)
    Set roottemp = root.RelevantSets(x)

    Randomize
    x = Int(Random(1, roottemp.Cardinality))
    notOkay = True
  While notOkay
    notOkay = False
    While ((roottemp.Element(x).attname = r.Decision.attname) And
(roottemp.Element(x).valname = r.Decision.valname))
        x = Int(Random(1, roottemp.Cardinality))
        notOkay = True
    Wend
    For i = 1 To r.olist.count Step 2
        If ((roottemp.Element(x).attname = r.olist(i).attname) And
(roottemp.Element(x).valname = r.olist(i).valname)) Then
            x = Int(Random(1, roottemp.Cardinality))
            notOkay = True
        End If
    Next
    If r.olist.count > (roottemp.Cardinality - 2) * 2 Then
        notOkay = False
        Set SelectSet = r.Decision
        Exit Function
    End If
  Wend
    Set SelectSet = root.Element(x)
End Function
Function NewRule() As RuleClass
    Dim i As Long
    Dim rc As RuleClass
    Dim x As Long

    Set rc = New RuleClass
    Randomize
    i = Int(Rnd(1) * Decisions.Cardinality) + 1
    rc.Decision = Decisions.Element(i)
    x = Int(Random(1, root.Cardinality))
    While (root.Element(x).attname = rc.Decision.attname) And
(root.Element(x).valname = rc.Decision.valname)
```

```
      x = Int(Random(1, root.Cardinality))
   Wend
   rc.AddToRule root.Element(x)
   rc.ReEvaluate
   Set NewRule = rc
End Function

Sub PostProcess(BS As Description)
   Dim i As Long
   Dim j As Long

   i = 0
   j = 1
   While i < BS.Cardinality
      i = i + 1
      'If BS.Element(i).Precision = 0 Or BS.Element(i).Recall = 0 Then
      '    BS.DeleteRule i
      '    i = i - 1
      'End If
      j = i
      If i = 0 Then i = 1
      While j < BS.Cardinality
         j = j + 1
         If BS.Element(i).Clause = BS.Element(j).Clause Then
            BS.DeleteRule j
            j = j - 1
         End If
      Wend
   Wend
End Sub

Sub InitializeGraph()
   Load Graph
   Graph.Visible = True
   Graph.Enabled = True
   Graph.pbGraph.ScaleHeight = 1
   Graph.pbGraph.ScaleWidth = GenCount
   PrevPopFitness = 1
   PrevBestFitness = 1
   DoEvents
End Sub

Sub PlotFitness(generation As Long, PopFitness As Single, BestFitness As Single)
   Graph.pbGraph.ForeColor = vbGreen
```

```
   Graph.pbGraph.Line (generation - 1, 1 - PrevPopFitness)-(generation, 1 -
PopFitness)
   Graph.pbGraph.ForeColor = vbBlue
   Graph.pbGraph.Line (generation - 1, 1 - PrevBestFitness)-(generation, 1 -
BestFitness)
   PrevPopFitness = PopFitness
   PrevBestFitness = BestFitness
   DoEvents
End Sub

Sub ErrorHandler(routine As String, err As ErrObject)
   Debug.Print routine + " error..."
   Debug.Print "..." + err.Source
   Debug.Print "..." + err.Description
End Sub
```

A1.4 FileHandler

```
Sub CreateFiles(datafile As String, filelist As Collection, iFlag As Long)
   TEMPPATH = App.Path + "\Temp"

   TotCoverage = 0
   TotAccuracy = 0
   Select Case iFlag
   Case 0  'no validation...just use the datafile

   Case 1  'Leave-One-Out validation
      LeaveOneOut datafile, K, filelist
   Case 2  'K-fold Cross-validation
      KFold datafile, K, filelist
   End Select
   filelist.Add datafile   'Add the original datafile to the list
End Sub

Sub ProcessFiles(filelist As Collection, iFlag As Long, Optional iResume As Long)
   On Error GoTo ProcessFiles_Error
   Dim x As Long
   Dim Coverage As Single
   Dim Accuracy As Single

   'TotCoverage = 0
   'TotAccuracy = 0
   If iResume < 1 Then iResume = 1
   For x = iResume To K
```

```
      BuildSetsFromFile filelist(1), True
      Select Case iFlag
      Case 0  'MLEM
         InduceRules
         strExt = ".MLEM"
      Case 1  'Arcanum
         Arcanum
         strExt = ".ARCANUM"
      End Select
      PrintRules Main.txtOutFile.Text + Trim(Str(x)) + strExt, , rules.Coverage,
rules.Accuracy
      VerifyRules Main.txtOutFile.Text + Trim(Str(x)) + strExt, filelist(2), 0,
Coverage, Accuracy
      TotCoverage = TotCoverage + Coverage
      TotAccuracy = TotAccuracy + Accuracy
      Kill filelist(1)
      Kill filelist(2)
      Kill Main.txtOutFile.Text + Trim(Str(x)) + strExt
      Kill Main.txtOutFile.Text + Trim(Str(x)) + strExt + ".verified"
      filelist.Remove 1
      filelist.Remove 1
      PrintTempFile TEMPPATH + "\temp." + Trim(Str(x)) + ".txt", TotCoverage,
TotAccuracy, iFlag
      If x > 1 Then
         Kill TEMPPATH + "\temp." + Trim(Str(x - 1)) + ".txt"
      End If
      msg "*** Completed iteration " + Str(x)
   Next
   If K > 1 Then
      Kill TEMPPATH + "\temp." + Trim(Str(x - 1)) + ".txt"
      'Debug.Print "Coverage = " + Format(TotCoverage / K, "percent")
      'Debug.Print "Accuracy = " + Format(TotAccuracy / K, "percent")
   End If
   BuildSetsFromFile Main.txtDataFile, True
   Select Case iFlag
   Case 0  'MLEM
      InduceRules
      strExt = ".MLEM"
   Case 1  'Arcanum
      Arcanum
      strExt = ".ARCANUM"
   End Select
   If K > 0 Then
      PrintRules Main.txtOutFile.Text + strExt, , TotCoverage / K, TotAccuracy / K
```

```
    Else
        PrintRules Main.txtOutFile.Text + strExt, , rules.Coverage, rules.Accuracy
    End If
    msg "Process Complete.", True
    Main.clock.Enabled = False
    Beep
    MsgBox "DONE!!"
ProcessFiles_Exit:
    Exit Sub

ProcessFiles_Error:
    ErrorHandler "FileHandler::ProcessFiles", err
    GoTo ProcessFiles_Exit
End Sub

Sub KFold(datafile As String, K As Long, filelist As Collection)
    On Error GoTo KFold_Error
    Dim ifile As Long
    Dim buf As String '* 50000
    Dim alldata As Collection
    Dim headers As Collection
    Dim KSets() As String
    Dim i As Long
    Dim j As Long
    Dim Z As Long
    Dim count As Long
    Dim idx As Long
    Dim txtTestFile As String
    Dim txtTrainFile As String
    Dim iTest As Long
    Dim iTrain As Long

    Set alldata = New Collection

    'Gather all data from the input file
    ifile = FreeFile
    Open datafile For Input As ifile
    While Not EOF(ifile)
        Line Input #ifile, buf
        alldata.Add buf
    Wend
    Close ifile

    'Remove file headers
```

```
Set headers = New Collection
While Left(alldata(1), 1) = "[" Or Left(alldata(1), 1) = "<"
   headers.Add alldata(1)
   alldata.Remove 1
Wend

'Split Data randomly into K sets
ReDim KSets(K) As String
ReDim KSets(K, Int(alldata.count / K) + 1) As String
'Debug.Print UBound(KSets, 1)
'Debug.Print UBound(KSets, 2)
count = 0
While alldata.count > K
   count = count + 1
   For i = 1 To K
      Randomize
      idx = Int((Rnd(1) * alldata.count)) + 1
      KSets(i, count) = Trim(alldata(idx))
      alldata.Remove idx
   Next
Wend
count = count + 1
For i = 1 To alldata.count
   Randomize
   idx = Int(Rnd(1) * alldata.count) + 1
   KSets(i, count) = Trim(alldata(idx))
   alldata.Remove idx
Next
Set alldata = Nothing

'DisplayKFoldSets KSets, K, count

'Make training and testing files from the K sets
iTrain = FreeFile
iTest = iTrain + 1
For i = 1 To K
   txtTestFile = TEMPPATH + "\test." + Trim(Str(i)) + ".txt"
   txtTrainFile = TEMPPATH + "\train." + Trim(Str(i)) + ".txt"
   Open txtTrainFile For Output As iTrain
   Open txtTestFile For Output As iTest
   'Print the data from I into the test file
   For j = 1 To headers.count
      Print #iTest, headers(j)
   Next
```

```
      For j = 1 To count
         Print #iTest, KSets(i, j)
      Next
      'Print data from all other sets into the training file
      For j = 1 To headers.count
         Print #iTrain, headers(j)
      Next
      For j = 1 To K
         If j <> i Then 'use all of the sets except the Ith one
            For Z = 1 To count
               If Trim(KSets(j, Z)) <> "" Then
                  Print #iTrain, KSets(j, Z)
               End If
            Next
         End If
      Next
      filelist.Add txtTrainFile
      filelist.Add txtTestFile
      Close iTrain
      Close iTest
   Next
KFold_Exit:
   Exit Sub

KFold_Error:
   ErrorHandler "FileHandler::KFold", err
   GoTo KFold_Exit

End Sub

Sub DisplayKFoldSets(KSets() As String, K As Long, count As Long)
   Dim i As Long
   Dim j As Long

   For i = 1 To K
      Debug.Print "SET " + Str(i) + " contains..."
      For j = 1 To count
         Debug.Print vbTab + Trim(KSets(i, j))
      Next
   Next
End Sub

Sub LeaveOneOut(txtDataFile As String, iIterations, filelist As Collection)
   Dim iData As Long
```

```
Dim iTrain As Long
Dim iTest As Long
Dim records As Collection
Dim buf As String
Dim i As Long
Dim x As Long
Dim iT As Long
Dim Sample As Long
Dim txtTrainFile As String
Dim txtTestFile As String

iData = FreeFile
iTrain = iData + 1
iTest = iTrain + 1

Set records = New Collection

Open txtDataFile For Input As iData
While Not EOF(iData)
   Line Input #iData, buf
   records.Add buf
Wend
Close iData
x = 1
While Left(records(x), 1) = "[" Or Left(records(x), 1) = "<"
   x = x + 1
Wend
x = x - 1
For iT = 1 To iIterations
   txtTestFile = TEMPPATH + "\test." + Trim(Str(iT)) + ".txt"
   txtTrainFile = TEMPPATH + "\train." + Trim(Str(iT)) + ".txt"
   Open txtTrainFile For Output As iTrain
   Open txtTestFile For Output As iTest
   Randomize
   Sample = Int(Rnd(1) * (records.count - x)) + x + 1
   For i = 1 To records.count
      If i <> Sample Then
         Print #iTrain, records(i)
      End If
   Next
   For i = 1 To x
      Print #iTest, records(i)
   Next
   Print #iTest, records(Sample)
```

```
        filelist.Add txtTrainFile
        filelist.Add txtTestFile
        Close iTrain
        Close iTest
    Next
End Sub

Sub PrintTempFile(tempfile As String, TotCov As Single, TotAcc As Single, iMod
As Long)
    Dim iTemp As Long

    iTemp = FreeFile
    Open tempfile For Output As iTemp

    Print #iTemp, Main.txtDataFile.Text
    Print #iTemp, Main.txtOutFile.Text
    Print #iTemp, TotCov
    Print #iTemp, TotAcc
    Print #iTemp, pweight
    Print #iTemp, rweight
    Print #iTemp, GenCount
    Print #iTemp, PopulationSize
    Print #iTemp, ComplimentsOkay
    Print #iTemp, iFlag
    Print #iTemp, K
    Print #iTemp, iMod

    Close iTemp
End Sub

Sub RestoreProcess()
    Dim iTemp As Long
    Dim tempfile As String
    Dim buf As String
    Dim iFlag As Long
    Dim filelist As Collection
    Dim K As Long
    Dim i As Long

    Debug.Print "...resuming..."

    tempfile = Dir$(App.Path + "\temp\temp*")
    If tempfile = "" Then Exit Sub
```

```
iTemp = FreeFile
Open App.Path + "\temp\" + tempfile For Input As iTemp

Load Main
Main.Visible = True
Input #iTemp, buf
Main.txtDataFile.Text = buf
Input #iTemp, buf
Main.txtOutFile.Text = buf
Input #iTemp, buf
TotCoverage = val(buf)
Input #iTemp, buf
TotAccuracy = val(buf)
Input #iTemp, buf
Main.txtPrecision = buf
Input #iTemp, buf
Main.txtRecall = buf
Input #iTemp, buf
Main.txtGenerations = buf
Input #iTemp, buf
Main.txtPopSize = buf
Input #iTemp, buf
If buf = "True" Then
   Main.chkCompliments.Value = 1
Else
   Main.chkCompliments.Value = 0
End If
Input #iTemp, buf
If buf = "0" Then
   Main.optNoValidate.Value = True
ElseIf buf = "1" Then
   Main.optLOO.Value = True
Else
   Main.optKFold.Value = True
End If
Input #iTemp, buf
Main.txtK = buf
Input #iTemp, buf
iFlag = val(buf)
Close iTemp

'Create filelist
Set filelist = New Collection
tempfile = Right(tempfile, Len(tempfile) - 5)
```

```
    K = val(tempfile)
    For i = K + 1 To val(Main.txtK)
        filelist.Add App.Path + "\temp\train." + Trim(Str(i)) + ".txt"
        filelist.Add App.Path + "\temp\test." + Trim(Str(i)) + ".txt"
    Next

    'Initialize variables
    SetControlVariables

    'ProcessFiles
    ProcessFiles filelist, iFlag, K + 1

End Sub
```

A1.5 MarkovModel

```
Sub MCMC()
    Dim d As SetClass
    Dim i As Long
    Dim r As RuleClass

    Set rules = New Description
    msg "Performing MCMC..."

    Categorize root
    For i = 1 To PopulationSize
        Set r = RandomRule
        While r.Precision < pweight Or r.Recall < rweight
            Set r = RandomRule
        Wend
        ReduceRule r
        rules.AddRule r
        msg "..." + Str(i) + "; (" + Str(rules.Cardinality) + " unique)"
        'Debug.Print r.Decision.valname + "   " + r.Clause
        'Debug.Print "   P:" + Str(r.Precision) + " ; R:" + Str(r.Recall)
    Next
    'LinearDropping
    rules.Sort

    msg "Coverage: " + Format(rules.Coverage, "0.00"), True
    msg "Writing rules to file...", True
    PrintRules Main.txtOutFile.Text, True

    msg "Process complete.", True
```

```
    Main.clock.Enabled = False
End Sub

Sub Categorize(root As SuperSet)
    Dim i As Long
    Dim currentatt As String
    Dim partition As SuperSet

    Set partitions = New Collection
    i = 1
    While i < root.Cardinality
        currentatt = root.Element(i).attname
        Set partition = New SuperSet
        partition.Name = currentatt
        While i <= root.Cardinality And root.Element(i).attname = currentatt
            partition.InsertSet root.Element(i)
            If i = root.Cardinality Then
                currentatt = ""
            Else
                i = i + 1
            End If
        Wend
        partitions.Add partition
    Wend
End Sub

Function RandomRule() As RuleClass
    Dim r As RuleClass
    Dim i As Long
    Dim j As Long
    Dim S As SetClass

    'Get a decision for the rule
    Set r = New RuleClass
    r.Decision = GetRandomDecision

    'Find the first state
    i = Int(Random(1, partitions.count))
    'Add attributes until a decision state is reached
    While partitions(i).Contains(r.Decision.attname, r.Decision.valname) = 0
        Set S = Sample(partitions(i))
        r.AddToRule S, "and"
        i = Int(Random(1, partitions.count))
```

```
    Wend
    r.ReEvaluate
    'Select an attribute in that state

    Set RandomRule = r
End Function

Function GetRandomDecision() As SetClass
    Dim i As Long
    Dim S As SetClass

    Set S = New SetClass
    i = Int(Random(1, root.Cardinality))
    Set S = root.Element(i)
    If usedecision Then
        While Not S.Decision
            i = Int(Random(1, root.Cardinality))
            Set S = root.Element(i)
        Wend
    End If
    Set GetRandomDecision = S
End Function

Function Sample(S As SuperSet) As SetClass
    Dim i As Long

    i = Int(Random(1, S.Cardinality))
    Set Sample = S.Element(i)
    'Debug.Print S.Element(i).attname + ","; S.Element(i).valname
End Function
```

A1.6 RuleCheck

```
Dim pos() As String

Function VerifyRules(txtRules As String, txtfile As String, Optional FullReportFlag
As Long, Optional Coverage As Single, Optional Accuracy As Single)
    On Error GoTo 0
    Dim rules As Collection
    Dim allrules As Collection
    Dim rule_desc As Description
    Dim iFlag As Long
    Dim att As SetClass
    Dim i As Long
```

```
Dim r As RuleClass
Dim outf As Long
Dim txtVerify As String
Dim Family As SetClass
Dim x As Long
Dim idx As Long
Dim bRulesAdded As Boolean

bRulesAdded = False
iDecision = 0
ReDim header(0) As String
ReDim Names(0) As String
HeadPtr = 1
iCounter = 1

'Parse each line of the input file
BuildSetsFromFile txtfile, False

rfile = FreeFile
ruleindex = 0
Set rules = New Collection
Set allrules = New Collection
Set rule_desc = New Description
Open txtRules For Input As rfile
txtVerify = txtRules + ".verified"
outf = FreeFile
Open txtVerify For Output As outf
While Not EOF(rfile)
   GetNextRule rules
   allrules.Add rules
   BuildRanges rules
   'Set att = New SetClass
   Set r = New RuleClass
  If rules.count <> 0 Then
   idx = root.Contains(rules(rules.count - 1), rules(rules.count))
   If idx > 0 Then
      r.Decision = root.Element(idx)
      If root.Contains(rules(1), rules(2)) > 0 Then
         r.AddToRule root.Element(root.Contains(rules(1), rules(2))), "and"
      End If
      'att.Union root.Element(root.Contains(rules(1), rules(2)))
      For i = 3 To rules.count - 3 Step 3
         'att.Intersect root.Element(root.Contains(rules(i), rules(i + 1)))
         x = root.Contains(rules(i + 1), rules(i + 2))
```

```
            If x > 0 Then
               r.AddToRule root.Element(root.Contains(rules(i + 1), rules(i + 2))),
rules(i)
            Else
               Set r = New RuleClass
               i = rules.count
            End If
         Next
         r.ReEvaluate
      Else
         i = rules.count
      End If
      If i < rules.count And r.Clause <> "" Then
         bRulesAdded = True
         rule_desc.AddRule r
         rule_desc.ReEvaluate
         Print #outf, "Rule:  " + rules(rules.count - 1) + ", " + rules(rules.count)
         Print #outf, "    " + r.Clause
         Print #outf, "   Precision: " + Format(r.Precision, "percent") + ";  Recall: " +
Format(r.Recall, "percent")
         If FullReportFlag = 1 Then
            For i = 1 To r.Cardinality
               If r.Decision.Contains(r.Element(i)) <> 0 Then
                  Print #outf, vbTab + Str(r.Element(i)) + " : CORRECT"
               Else
                  Set Family = FindClass(r.Decision.attname, r.Element(i))
                  Print #outf, vbTab + Str(r.Element(i)) + " : INCORRECT -> (" +
Family.attname + " , " + Family.valname + ")"
               End If
            Next
            If r.Cardinality = 0 Then
               Print #outf, vbTab + "  :::NO INSTANCES OF THIS RULE"
            End If
         End If
      Else
         Print #outf, "Rule:  " + rules(rules.count - 1) + ", " + rules(rules.count)
         buf = ""
         For i = 1 To (rules.count) / 2 Step 2
            buf = buf + "(" + rules(i) + "," + rules(i + 1) + ")"
            If i <> rules.count - 3 Then
               buf = buf + " & "
            End If
         Next
         buf = buf + " -> (" + rules(i) + "," + rules(i + 1) + ")"
```

```
            Print #outf, "    " + buf
            Print #outf, vbTab + "  :::NO INSTANCES OF THIS RULE"
        End If
        'Debug.Print "Rule:  " + rules(rules.count - 1) + ", " + rules(rules.count)

        'Debug.Print "    Precision: "
        'For i = 1 To att.cardinality
            'Debug.Print "      " + Str(att.Element(i))
        'Next
        'Debug.Print "    " + r.clause

        'Debug.Print "    Precision: " + Format(r.Precision, "0.00") + ";  Recall: " +
Format(r.Recall, "0.00")

        'Debug.Print root.Element(root.Contains(rules(rules.count - 1),
rules(rules.count))).SubSet(att)
      End If
    Wend

    If bRulesAdded Then
        Coverage = rule_desc.Coverage
        Accuracy = rule_desc.Accuracy
        'Debug.Print "Coverage: " + Format(Coverage, "percent")
        'Debug.Print "Accuracy: " + Format(Accuracy, "percent")
    Else
        Coverage = 0
        Accuracy = 1
        'Debug.Print "Coverage: " + Format(Coverage, "percent")
        'Debug.Print "Accuracy: " + Format(Accuracy, "percent")
    End If

    Close rfile
    Close outf
End Function

Sub GetNextRule(rules As Collection)
    Set rules = New Collection
    Dim buf As String
    Dim i As Long
    Dim word As String
    Dim wordon As Boolean

    Line Input #rfile, buf
    While Left(buf, 1) <> "(" And Not EOF(rfile)
```

```
      Line Input #rfile, buf
   Wend
   If EOF(rfile) And buf = "" Then
      Exit Sub
   End If
   'While Len(buf) = 0 And Not EOF(rFile)
   '   Line Input #rFile, buf
   'Wend
   i = 1
   wordon = False
   While i < Len(buf)
      Select Case mid(buf, i, 1)
         Case "("       'starts a new word
            word = ""
            wordon = True
         Case ")"       'ends a word
            rules.Add word
            word = ""
            wordon = False
         Case ",", " "      'ends a word, starts a new word
            If word <> "" And word <> "and" Then
               rules.Add word
            End If
            word = ""
         Case "&"       'burn the character

         Case "-"       'if next character is > then burn, else part of word
            If mid(buf, i + 1, 1) = ">" Then
               i = i + 1
            Else
               word = word + mid(buf, i, 1)
            End If
         Case Else
            word = word + mid(buf, i, 1)
      End Select
      i = i + 1
   Wend
   rules.Add word
   ruleindex = ruleindex + 1
   'Line Input #rfile, buf
End Sub

Sub HighlightString(buf As String, st As String, linedx As Long, pos() As String)
   Dim x As Long
```

```
    st = Format(st, ">")
    x = InStr(buf, st)
    While x > 0
        pos(linedx, x) = st
        x = Len(st) + x + 25
        x = InStr(x, buf, st)
    Wend
End Sub

Sub CheckFileForString(txtfile As String, allrules As Collection)
    Dim ifile As Long
    Dim ofile As Long
    Dim buf As String
    Dim lines As Collection
    Dim i As Long
    Dim j As Long
    Dim x As Long
    Dim idx As Long
    Dim rules As Collection
    Dim st As String
    Dim dxn As String

    Set rules = New Collection
    Set lines = New Collection
    ifile = FreeFile
    Open txtfile For Input As ifile
    ofile = FreeFile
    Open txtfile + ".txt" For Output As ofile

    While Not EOF(ifile)
        Line Input #ifile, buf
        lines.Add Format(buf, ">")
    Wend

    ReDim pos(lines.count, 5000) As String
    For i = 1 To allrules.count
        For j = 2 To allrules(i).count Step 2
            For x = 1 To lines.count
                dxn = Format(allrules(i)(allrules(i).count), ">")
                If dxn = Right(lines(x), Len(dxn)) Then
                    buf = lines(x)
                    st = allrules(i)(j)
                        'lines.Add HighlightString(buf, st)
```

```
                HighlightString buf, st, x, pos
              pos(x, 5000) = dxn
            End If
          Next
        Next
    Next

    For i = 1 To UBound(pos, 1)
      buf = pos(i, 5000)
      For j = 1 To UBound(pos, 2) - 1
        If pos(i, j) <> "" Then
          buf = buf + " -> " + pos(i, j)
        End If
      Next
      Print #ofile, buf
    Next

    Close ofile
    Close ifile
End Sub

Function FindClass(sAttName As String, vItem As Long) As SetClass
    Dim i As Long
    Dim found As Boolean

    found = False
    i = 1
    While Not found And i < root.Cardinality
      If root.Element(i).attname = sAttName Then 'correct attribute
        If root.Element(i).Contains(vItem) <> 0 Then 'found it
          Set FindClass = root.Element(i)
          found = True
        Else 'didn't find it
          i = i + 1
        End If
      Else 'wrong attribute
        i = i + 1
      End If
    Wend
End Function

Sub BuildRanges(rules As Collection)
    Dim i As Long
    Dim idx As Long
```

```
    Dim minpoint As Single
    Dim maxpoint As Single
    Dim ipoint As Long
    Dim S As SetClass
    Dim bCreatable As Boolean

    i = 1
    While i < rules.count
       If rules(i) = "AND" Or rules(i) = "XOR" Or rules(i) = "OR" Then

       Else
          'if it's a range, then see if it already exists
          If IsRange(rules(i + 1), minpoint, maxpoint) Then
             idx = root.Contains(rules(i), rules(i + 1), ipoint)
             If idx = 0 Then  'set doesn't exist...see if it's creatable
                Set S = New SetClass
                S.attname = rules(i)
                S.valname = rules(i + 1)
                bCreatable = True
                While root.Element(ipoint).attname = rules(i) And bCreatable
                  If Not (IsRange(root.Element(ipoint).valname)) Then
                    If val(root.Element(ipoint).valname) >= minpoint _
                    And val(root.Element(ipoint).valname) <= maxpoint Then
                       S.Union root.Element(ipoint)
                    End If
                  End If
                  ipoint = ipoint + 1
                  If ipoint > root.Cardinality Then
                     ipoint = 1
                     bCreatable = False
                  End If
                Wend
                If S.Cardinality <> 0 Then
                   root.InsertSet S
                End If
             End If
          End If
          i = i + 1
       End If
       i = i + 1
    Wend
End Sub
```

```
Function IsRange(range As String, Optional minpoint As Single, Optional maxpoint
As Single) As Boolean
   Dim i As Long

   i = InStr(range, "..")
   If i > 1 Then
      IsRange = True
      minpoint = val(Left(range, i))
      maxpoint = val(Right(range, Len(range) - i - 1))
   End If
End Function
```

A1.7 Description object

```
Private rules As Collection
Private bChanged As Boolean
Private sCoverage As Single
Private sAccuracy As Single

Private Sub Class_Initialize()
   Set rules = New Collection
End Sub

Public Sub AddRule(r As RuleClass)
   On Error GoTo AddRule_Error
   Dim idx As Long
   Dim att As SetClass
   Dim pos As Long

   bChanged = True
   pos = Contains(r.Decision.attname, r.Decision.valname, r.Clause, idx)
   If pos = 0 Then  'add new set
      'Set att = New SetClass
      'att.attname = r.Decision.attname
      'att.valname = r.Decision.valname
      If idx > rules.count Or idx < 1 Then
         rules.Add r
      Else
         rules.Add r, , idx
      End If
      r.count = 1
      'Sort
   Else    'set is already there, just add the example
      rules(pos).count = rules(pos).count + 1
```

```
    End If

AddRule_Exit:
    Exit Sub

AddRule_Error:
    ErrorHandler "Description::Addrule", err
    GoTo AddRule_Exit
End Sub

Public Property Get Cardinality() As Long
    Cardinality = rules.count
End Property

Public Sub DeleteRule(x As Long)
    bChanged = True

    rules.Remove x

    a = Coverage
    a = Accuracy
    'Debug.Print Coverage
    'Debug.Print Accuracy
    bChanged = False
End Sub

Public Property Get Element(x As Long) As RuleClass
    Set Element = rules(x)
End Property

Public Function Contains(attname As String, valname As String, Clause As String,
Optional insertpoint As Long) As Long
    Dim found As Boolean
    Dim low As Long
    Dim mid As Long
    Dim high As Long

    attname = Format(attname, "<")
    valname = Format(valname, "<")
    Clause = Format(Clause, "<")
    found = False
    high = rules.count
    low = 1
    mid = Int((high - low) / 2) + 1
```

```
    Contains = 0

    While Not found And mid >= low And mid <= high
        If rules(mid).Decision.attname > attname Then
            high = mid - 1
        ElseIf rules(mid).Decision.attname < attname Then
            low = mid + 1
        ElseIf rules(mid).Decision.attname = attname Then
            If rules(mid).Decision.valname > valname Then
                high = mid - 1
            ElseIf rules(mid).Decision.valname < valname Then
                low = mid + 1
            ElseIf rules(mid).Decision.valname = valname Then
                If rules(mid).Clause > Clause Then
                    high = mid - 1
                ElseIf rules(mid).Clause < Clause Then
                    low = mid + 1
                ElseIf rules(mid).Clause = Clause Then
                    found = True
                    Contains = mid
                End If
            End If
        End If
        If high = low Then
            mid = low
        Else
            mid = Int((high - low) / 2) + low + 1
        End If
    Wend
    insertpoint = mid
End Function

Public Sub Sort()
    Dim i As Long
    Dim j As Long
    Dim max As Single
    For i = 0 To rules.count - 1
        max = 1
        For j = 1 To rules.count - i
            If (rules(j).count) > rules(max).count Then
                max = j
            End If
        Next
        rules.Add rules(max)
```

```
      rules.Remove max
    Next
End Sub

Public Property Get Coverage() As Single
    Dim U As SetClass
    Dim d As Collection
    Dim i As Long
    Dim numofclasses As Long
    Dim found As Boolean

  If bChanged = True Then
    Set d = New Collection
    Set U = New SetClass
    U.Union Universe
    For i = 1 To rules.count
       U.Difference rules(i).SetFromRule
       If d.count = 0 Then
          d.Add rules(i).Decision
       Else
          j = 0
          found = False
          While j < d.count And Not found
             j = j + 1
             If d(j).attname = rules(i).Decision.attname And _
                d(j).valname = rules(i).Decision.valname Then
                 found = True
             End If
          Wend
          If Not found Then
             d.Add rules(i).Decision
          End If
       End If
    Next

    numofclasses = d.count
    If numofclasses = 0 Or Decisions.Cardinality = 0 Then
       Coverage = ((Universe.Cardinality - U.Cardinality) / Universe.Cardinality)
    Else
       Coverage = ((Universe.Cardinality - U.Cardinality) / Universe.Cardinality) _
            * (numofclasses / Decisions.Cardinality)
    End If
    sCoverage = Coverage
  Else
```

```
    Coverage = sCoverage
  End If
End Property

Public Property Get Accuracy() As Single
    Dim i As Long
    Dim sum As Long
    Dim total As Long

  If bChanged = True Then
    sum = 0
    total = 0
    For i = 1 To rules.count
       sum = sum + (rules(i).Precision * rules(i).Cardinality)
       total = total + rules(i).Cardinality
    Next
    If total = 0 Then
       Accuracy = 0
    Else
       Accuracy = sum / total
    End If
    sAccuracy = Accuracy
  Else
    Accuracy = sAccuracy
  End If
End Property

Public Property Get Fitness() As Single
    Fitness = (Accuracy * pweight) + (Coverage * rweight)
End Property

Public Sub ReEvaluate()
    x = Coverage
    x = Accuracy
    'Debug.Print Coverage
    'Debug.Print Accuracy
    bChanged = False
End Sub
```

A1.8 RuleClass object

```
Private oDecision As SetClass
Public olist As Collection
Private oExamples As Collection
```

```
Private ostats As String
Private icount As Long
Private bChanged As Boolean
Private sClause As String
Private sPrecision As Single
Private sRecall As Single

Public Property Let count(i As Long)
   icount = i
End Property
Public Property Get count() As Long
   count = icount
End Property
Public Property Let stats(buf As String)
   ostats = buf
End Property
Public Property Get stats() As String
   stats = ostats
End Property
Private Sub Class_Initialize()
   Set olist = New Collection
   Set oxamples = New Collection
End Sub
Public Property Let Decision(val As SetClass)
   Set oDecision = val
End Property
Public Property Get Decision() As SetClass
   Set Decision = oDecision
End Property
Public Sub RemoveFromRule(i As Long)
   If i < 1 Then Exit Sub
   If i = 1 Then
      olist.Remove 1
      If olist.count > 0 Then olist.Remove 1
   Else
      'i = (i - 1) * 2
      olist.Remove i
      olist.Remove i - 1
   End If
   'ReEvaluate
End Sub
Public Property Get Clause() As String
   Dim buf As String
   Dim i As Long
```

```
    If olist.count = 0 Then
        Clause = ""
        Exit Property
    End If
    If bChanged = True Then
        buf = "(" + olist(1).attname + "," + olist(1).valname + ") "
        For i = 2 To olist.count - 1 Step 2
            buf = buf + olist(i) + " "
            buf = buf + "(" + olist(i + 1).attname + "," + olist(i + 1).valname + ") "
        Next
        sClause = buf
        Clause = buf
    Else
        Clause = sClause
    End If
End Property
Public Property Get RuleSize() As Long
    RuleSize = olist.count ' * 2 - 1
End Property
Public Property Get Cardinality() As Long
    Cardinality = oExamples.count
End Property
Public Sub AddToRule(S As SetClass, Optional operator As String)
    If olist.count = 0 Then
        olist.Add S
    Else
        olist.Add operator
        olist.Add S
    End If
    'ReEvaluate
End Sub
Public Property Get Element(i As Long) As Variant
    If i > oExamples.count Or i = 0 Then
        Element = 0
    Else
        Element = oExamples(i)
    End If
End Property
Public Sub ReEvaluate()
    Dim i As Long
    Dim oset As SetClass

    If olist.count = 0 Then
```

```
        Set oExamples = New Collection
        Exit Sub
    End If
    Set oset = New SetClass
    oset.Union olist(1)
    For i = 2 To olist.count - 1 Step 2
        Select Case olist(i)
        Case "and", "AND"
            oset.Intersect olist(i + 1)
        Case "or", "OR"
            oset.Union olist(i + 1)
        Case "xor", "XOR"
            oset.XORS olist(i + 1)
        Case "diff", "DIFF"
            oset.Difference olist(i + 1)
        Case Else
            Debug.Print "Failed to evaluate operator " + olist(i)
        End Select
    Next
    oset.GetList oExamples
    bChanged = True
    x = Clause
    x = Precision
    x = Recall
    'Debug.Print Clause
    'Debug.Print Precision
    'Debug.Print Recall
    bChanged = False
End Sub
Public Property Get Precision() As Single
    Dim i As Long
    Dim count As Long

    If olist.count = 0 Then
        Precision = 0
        Exit Property
    End If
    If bChanged = True Then
        i = 0
        For i = 1 To oExamples.count
            If oDecision.Contains(oExamples(i)) <> 0 Then
                count = count + 1
            End If
        Next
```

```
        If count = 0 Then
            Precision = count
        Else
            Precision = count / oExamples.count
        End If
        sPrecision = Precision
    Else
        Precision = sPrecision
    End If
End Property
Public Property Get Recall() As Single
    Dim i As Long
    Dim count As Long

    If olist.count = 0 Then
        Recall = 0
        Exit Property
    End If
    If bChanged = True Then
        i = 0
        For i = 1 To oExamples.count
            If oDecision.Contains(oExamples(i)) <> 0 Then
                count = count + 1
            End If
        Next
        If count = 0 Then
            Recall = 0
        Else
            Recall = count / oDecision.Cardinality
        End If
        sRecall = Recall
    Else
        Recall = sRecall
    End If
End Property

Public Property Get Fitness() As Single
    Fitness = Precision + Recall
End Property

Public Property Get SetFromRule() As SetClass
    Dim S As SetClass

    Set S = New SetClass
```

```
    S.AddList oExamples
    Set SetFromRule = S
End Property
Public Property Get CopyRule() As RuleClass
    Dim r As RuleClass
    Dim i As Long

    Set r = New RuleClass
    r.Decision = oDecision
    r.AddToRule olist(1)
    For i = 3 To olist.count Step 2
        r.AddToRule olist(i), olist(i - 1)
        'If i + 1 < olist.count Then
        '    r.AddToRule olist(i), olist(i - 1)
        'Else
        '    r.AddToRule olist(i)
        'End If
    Next
    r.ReEvaluate
    Set CopyRule = r
End Property
```

A1.9 SetClass object

```
Private oAttName As String
Private oValName As String
Private olist As Collection
Private oDecision As Boolean


Public Property Let Decision(val As Boolean)
    oDecision = val
End Property
Public Property Get Decision() As Boolean
    Decision = oDecision
End Property
Public Property Let stats(buf As String)
    ostats = buf
End Property
Public Property Get stats() As String
    stats = ostats
End Property
Public Property Let valname(buf As String)
    oValName = Format(buf, "<")
```

```
End Property
Public Property Get valname() As String
   valname = oValName
End Property
Public Property Let attname(buf As String)
   oAttName = Format(buf, "<")
End Property
Public Property Get attname() As String
   attname = oAttName
End Property
Public Property Get Cardinality() As Long
   Cardinality = olist.count
End Property
Public Property Get Element(i As Long) As Variant
   If i > olist.count Or i = 0 Then
      Element = 0
   Else
      Element = olist(i)
   End If
End Property
Public Sub AddToSet(i As Variant)
   Dim idx As Long

   If Contains(i, idx) = 0 Then
      If idx > olist.count Or idx < 1 Then
         olist.Add i
      Else
         olist.Add i, , idx
      End If
      'Sort
   End If
End Sub
Public Sub RemoveMember(i As Variant)
   Dim x As Long
   x = Contains(i)
   If x <> 0 Then
      olist.Remove x
   End If
   'Sort
End Sub
Public Sub AddList(c As Collection)
   Dim i As Long
   For i = 1 To c.count
      AddToSet c(i)
```

```
    Next
End Sub
Public Sub GetList(c As Collection)
    Set c = olist
End Sub
Private Sub Sort()
    Dim i As Long
    Dim j As Long
    Dim min As Single
    For i = 0 To olist.count - 1
        min = 1
        For j = 1 To olist.count - i
            If val(olist(j)) < val(olist(min)) Then
                min = j
            End If
        Next
        olist.Add olist(min)
        olist.Remove min
    Next
End Sub
Public Sub Union(x As SetClass)
    Dim i As Long
    For i = 1 To x.Cardinality
        AddToSet x.Element(i)
    Next
    'Sort
End Sub
Public Sub Intersect(x As SetClass)
    Dim i As Long
    Dim idx As Long
    i = 1
    While i <= olist.count
        If x.Contains(olist(i), idx) = 0 Then
            olist.Remove i
            i = i - 1
        End If
        i = i + 1
    Wend
    'Sort
End Sub
Private Sub Class_Initialize()
    Set olist = New Collection
End Sub
```

```
Public Function Contains(i As Variant, Optional insertpoint As Long) As Long
   Dim found As Boolean
   Dim low As Long
   Dim mid As Long
   Dim high As Long

   found = False
   high = olist.count
   low = 1
   mid = Int((high - low) / 2) + 1
   Contains = 0

   While Not found And mid >= low And mid <= high
      If olist(mid) > i Then
         high = mid - 1



      ElseIf olist(mid) < i Then
         low = mid + 1
      ElseIf olist(mid) = i Then
         found = True
         Contains = mid
      End If
      If high = low Then
         mid = low
      Else
         mid = Int((high - low) / 2) + low + 1
      End If
   Wend
   insertpoint = mid
End Function
Public Sub Difference(Y As SetClass)
   Dim i As Long
   For i = 1 To Y.Cardinality
      RemoveMember Y.Element(i)
   Next
End Sub
Public Sub XORS(Y As SetClass)
   Dim x As SetClass
   Dim Z As SetClass
   Set Z = New SetClass
   Set x = New SetClass
```

```
    Z.Union Y
    x.AddList olist
    Z.Intersect x
    x.Union Y
    x.Difference Z
    x.GetList olist
End Sub
Public Function SubSet(Y As SetClass) As Boolean
    'returns TRUE if Y is a subset of X, otherwise returns FALSE
    Dim i As Long
    Dim idx As Long

    i = 1
    SubSet = True
    If Y.Cardinality = 0 Then SubSet = False
    While SubSet And i < Y.Cardinality
        If Contains(Y.Element(i), idx) <> 0 Then
            i = i + 1
        Else
            SubSet = False
        End If
    Wend
End Function
Public Function Rulesubset(r As RuleClass) As Boolean
    Dim i As Long
    Dim idx As Long

    i = 1
    Rulesubset = True
    If r.Cardinality = 0 Then Rulesubset = False
    While Rulesubset And i < r.Cardinality
        If Contains(r.Element(i), idx) <> 0 Then
            i = i + 1
        Else
            Rulesubset = False
        End If
    Wend
End Function
Public Sub Compliment()
    Dim otemp As New SetClass
    Dim i As Long

    Set otemp = New SetClass
    otemp.Union Universe
```

```
    For i = 1 To olist.count
        otemp.RemoveMember olist(i)
    Next
    Set olist = Nothing
    Set olist = New Collection
    For i = 1 To otemp.Cardinality
        olist.Add otemp.Element(i)
    Next
    oValName = "not " + oValName
    Set otemp = Nothing
End Sub
Public Sub ShowMembers()
    Dim i As Long

    Debug.Print oAttName + " , " + oValName
    For i = 1 To olist.count
        Debug.Print "    " + Str(olist(i))
    Next
End Sub
Public Function Equal(S As SetClass) As Boolean
    Dim eq As Boolean
    Dim i As Long

    eq = True
    If S.Cardinality <> olist.count Then
        eq = False
    End If
    i = 1
    While eq = True And i < olist.count
        If olist(i) <> S.Element(i) Then
            eq = False
        End If
        i = i + 1
    Wend
    Equal = eq
End Function
Public Function Intersects(S As SetClass) As Boolean
    Dim i As Long
    Dim idx As Long
    i = 1
    While i <= olist.count
        If S.Contains(olist(i), idx) <> 0 Then
            Intersects = True
            Exit Function
```

```
      Else
         i = i + 1
      End If
   Wend
End Function
```

A1.10 SuperSet object

```
Private oName As String
Private olist As Collection
Private oRelevantSets As Collection

Public Property Let Name(buf As String)
   oName = buf
End Property
Public Property Get Name() As String
   Name = oName
End Property
Public Property Get Cardinality() As Long
   Cardinality = olist.count
End Property
Public Property Get Element(i As Long) As SetClass
   If i > olist.count Or i = 0 Then
      Set Element = Nothing
   Else
      Set Element = olist(i)
   End If
End Property
Public Sub InsertSet(i As SetClass)
   Dim idx As Long
   Dim pos As Long

   pos = Contains(i.attname, i.valname, idx)
   If pos = 0 Then
      If idx > olist.count Or idx < 1 Then
         olist.Add i
      Else
         olist.Add i, , idx
      End If
   End If
End Sub
Public Sub AddToSet(attname As String, valname As String, example As Variant)
   Dim idx As Long
   Dim att As SetClass
```

```
    Dim pos As Long

    pos = Contains(attname, valname, idx)
    If pos = 0 Then  'add new set
        Set att = New SetClass
        att.attname = attname
        att.valname = valname
        att.AddToSet example
        If idx > olist.count Or idx < 1 Then
            olist.Add att
        Else
            olist.Add att, , idx
        End If
        'Sort
    Else    'set is already there, just add the example
        olist(pos).AddToSet example
    End If
End Sub
Public Sub RemoveMember(i As SetClass)
    Dim x As Long
    'If IsEmpty(i) Then
        x = Contains(i.attname, i.valname)
        If x <> 0 Then
            olist.Remove x
        End If
    'End If
    'Sort
End Sub
Public Sub AddList(c As Collection)
    Dim i As Long
    For i = 1 To c.count
        AddToSet c(i)
    Next
End Sub
Public Sub GetList(c As Collection)
    Set c = olist
End Sub
Public Sub Sort(Optional bNumeric As Boolean)
    Dim i As Long
    Dim j As Long
    Dim min As Single
    For i = 0 To olist.count - 1
        min = 1
        If bNumeric = False Then
```

```
            For j = 1 To olist.count - i
               If olist(j).attname <= olist(min).attname _
               And olist(j).valname < olist(min).valname Then
                  min = j
               End If
            Next
         Else
            For j = 1 To olist.count - i
               If olist(j).attname <= olist(min).attname _
               And val(olist(j).valname) < val(olist(min).valname) Then
                  min = j
               End If
            Next
         End If
         olist.Add olist(min)
         olist.Remove min
      Next
   End Sub
   Public Sub Union(x As SuperSet)
      Dim i As Long
      Dim pos As Long
      Dim idx As Long
      For i = 1 To x.Cardinality
         pos = Contains(x.Element(i).attname, x.Element(i).valname, idx)
         If pos = 0 Then
            If idx > olist.count Or idx < 1 Then
               olist.Add x.Element(i)
            Else
               olist.Add x.Element(i), , idx
            End If
         End If
      Next
      'Sort
   End Sub
   Public Sub Intersect(x As SuperSet)
      Dim i As Long
      i = 1
      While i <= olist.count
         If x.Contains(olist(i)) = 0 Then
            olist.Remove i
            i = i - 1
         End If
         i = i + 1
      Wend
```

```
    Sort
End Sub
Private Sub Class_Initialize()
    Set olist = New Collection
End Sub

Public Function Contains(attname As String, valname As String, Optional insertpoint
As Long) As Long
    Dim found As Boolean
    Dim low As Long
    Dim mid As Long
    Dim high As Long

    attname = Format(attname, "<")
    valname = Format(valname, "<")
    found = False
    high = olist.count
    low = 1
    mid = Int((high - low) / 2) + 1
    Contains = 0

    While Not found And mid >= low And mid <= high
        If olist(mid).attname > attname Then
            high = mid - 1
        ElseIf olist(mid).attname < attname Then
            low = mid + 1
        ElseIf olist(mid).attname = attname Then
            If olist(mid).valname > valname Then
                high = mid - 1
            ElseIf olist(mid).valname < valname Then
                low = mid + 1
            ElseIf olist(mid).valname = valname Then
                found = True
                Contains = mid
            End If
        End If
        If high = low Then
            mid = low
        Else
            mid = Int((high - low) / 2) + low + 1
        End If
    Wend
    insertpoint = mid
End Function
```

```
Public Sub Difference(Y As SuperSet)
   Dim i As Long
   For i = 1 To Y.Cardinality
      RemoveMember Y.Element(i)
   Next
End Sub
Public Sub XORS(Y As SuperSet)
   Dim x As SuperSet
   Dim Z As SuperSet
   Set Z = New SuperSet
   Set x = New SuperSet

   Z.Union Y
   x.AddList olist
   Z.Intersect x
   x.Union Y
   x.Difference Z
   x.GetList olist
End Sub
Public Function SubSet(Y As SuperSet) As Boolean
   Dim i As Long

   i = 1
   SubSet = True
   If Y.Cardinality = 0 Then SubSet = False
   While SubSet And i < Y.Cardinality
      If Contains(Y.Element(i)) <> 0 Then
         i = i + 1
      Else
         SubSet = False
      End If
   Wend
End Function
Public Sub FindRelevantSets()
   Dim otemp As SuperSet
   Set oRelevantSets = New Collection

   For i = 1 To olist.count
      Set otemp = New SuperSet
      If olist(i).Decision = True Then
         For j = 1 To olist.count
            If i <> j And olist(i).Intersects(olist(j)) Then
               otemp.InsertSet olist(j)
            End If
```

```
            Next
        End If
        oRelevantSets.Add otemp
    Next
End Sub
Public Function RelevantSets(i As Long) As SuperSet
    Set RelevantSets = oRelevantSets(i)
End Function
```

A1.11 FileDialog form

```
Option Explicit
Private objTextBox As TextBox

Private Sub Dir1_Change()
    File1.Path = Dir1.Path
End Sub

Private Sub Drive1_Change()
    Dir1.Path = Drive1.Drive
End Sub

Public Property Let FileBox(obj As TextBox)
    Set objTextBox = obj
End Property

Private Sub File1_DblClick()
    OKButton_Click
End Sub

Private Sub Form_Load()
    Drive1.Drive = "c:\"
    Dir1.Path = App.Path + "\DataFiles"
    File1.Path = App.Path + "\DataFiles"
End Sub

Private Sub OKButton_Click()
    objTextBox.Text = File1.Path + "\" + File1.FileName
    Unload FileDialog
End Sub
```

A1.12 Main form

```
Private Sub btnArcanum_Click()
   On Error GoTo 0
   Dim i As Long
   Dim x As Long
   Dim filelist As Collection
   Dim TotCoverage As Single
   Dim TotAccuracy As Single
   Dim Coverage As Single
   Dim Accuracy As Single

   SetControlVariables
   Set filelist = New Collection

   err = 0
   i = FreeFile
   Open Main.txtDataFile.Text For Input As i
   If err <> 0 Then
      err = 0
      MsgBox "Please indicate a valid data set file."
      Exit Sub
   End If

   CreateFiles Main.txtDataFile.Text, filelist, iFlag
   ProcessFiles filelist, 1
End Sub

Private Sub btnBrowse1_Click()
   Load FileDialog
   FileDialog.FileBox = txtDataFile
   FileDialog.Visible = True
   btnMLEM.Enabled = True
   btnArcanum.Enabled = True
   btnVerify.Enabled = True
End Sub

Private Sub btnBrowse2_Click()
   Load FileDialog
   FileDialog.FileBox = txtOutFile
   FileDialog.Visible = True
End Sub

Private Sub btnBrowse3_Click()
```

```
    Load FileDialog
    FileDialog.FileBox = txtRuleFile
    FileDialog.Visible = True
End Sub

Private Sub btnMCMC_Click()
    On Error GoTo 0
    Dim i As Long

    SetControlVariables
    clock.Enabled = True

    err = 0
    i = FreeFile
    Open Main.txtDataFile.Text For Input As i
    If err <> 0 Then
        err = 0
        MsgBox "Please indicate a valid data set file."
        Exit Sub
    End If
    BuildSetsFromFile Main.txtDataFile.Text
    MCMC
End Sub

Private Sub btnMLEM_Click()
    On Error GoTo 0
    Dim i As Long
    Dim filelist As Collection

    SetControlVariables
    clock.Enabled = True

    Set filelist = New Collection
    err = 0
    i = FreeFile
    Open Main.txtDataFile.Text For Input As i
    If err <> 0 Then
        err = 0
        MsgBox "Please indicate a valid data set file."
        Exit Sub
    End If

    CreateFiles Main.txtDataFile.Text, filelist, iFlag
    ProcessFiles filelist, 0
```

```
    'BuildSetsFromFile Main.txtDataFile.Text
    'InduceRules
    'msg "Writing rules to file...", True
    'PrintRules Main.txtOutFile.Text

    'msg "Process complete.", True

    'clock.Enabled = False
End Sub

Private Sub btnVerify_Click()
    On Error GoTo 0
    Dim i As Long
    Dim FullReportFlag As Long

    SetControlVariables
    clock.Enabled = True

    err = 0
    i = FreeFile
    Open Main.txtDataFile.Text For Input As i
    If err <> 0 Then
        err = 0
        MsgBox "Please indicate a valid data set file."
        Exit Sub
    End If

    err = 0
    i = FreeFile
    Open Main.txtRuleFile.Text For Input As i
    If err <> 0 Then
        err = 0
        MsgBox "Please indicate a valid Rules file."
        Exit Sub
    End If

    FullReportFlag = MsgBox("Do you want FULL results (examples for each rule)?",
vbYesNo)

    If FullReportFlag = 6 Then 'Display FULL results
        VerifyRules Main.txtRuleFile.Text, Main.txtDataFile.Text, 1
    Else 'Display only the rules and statistics
        VerifyRules Main.txtRuleFile.Text, Main.txtDataFile.Text, 0
```

```
    End If

    clock.Enabled = False
End Sub

Private Sub clock_Timer()
    If lblRunTime.Caption = "" Then
        clocktime = Now()
    End If
    'lblRunTime.Caption = Format(val(lblRunTime.Caption) + 1, "hh:mm:ss")
    'lblRunTime.Caption = (val(lblRunTime.Caption) + 1)
    lblRunTime.Caption = Format(Now() - clocktime, "h:mm:ss")
    DoEvents
End Sub

Private Sub Form_Load()
    Dim buf As String
    Dim ans As Long

    btnMLEM.Enabled = False
    btnArcanum.Enabled = False

    buf = Dir(App.Path + "\temp\temp*")
    If buf <> "" Then
        ans = MsgBox("ARCANUM was previously interrupted.  Would you like to
resume where it left off?", vbYesNo)
        If ans = 6 Then
            RestoreProcess
        End If
    End If
End Sub

Private Sub txtDataFile_Change()
    txtOutFile.Text = txtDataFile.Text + ".rules" + "." +
Trim(Str((val(txtPrecision.Text) * 100))) + "_" + Trim(Str((val(txtRecall.Text) *
100)))
End Sub



Private Sub txtPrecision_LostFocus()
    Dim x As Single
    x = val(txtPrecision)
    If x > 1 Then
```

```
      x = 1
   End If
   If x < 0 Then
      x = 0
   End If
   txtRecall = Format(1 - x, "0.00")
   txtPrecision = Format(x, "0.00")
   txtOutFile.Text = txtDataFile.Text + ".rules" + "." +
Trim(Str((val(txtPrecision.Text) * 100))) + "_" + Trim(Str((val(txtRecall.Text) *
100)))
End Sub

Private Sub txtRecall_LostFocus()
   Dim x As Single
   x = val(txtRecall)
   If x > 1 Then
      x = 1
   End If
   If x < 0 Then
      x = 0
   End If
   txtRecall = Format(x, "0.00")
   txtPrecision = Format(1 - x, "0.00")
   txtOutFile.Text = txtDataFile.Text + ".rules" + "." +
Trim(Str((val(txtPrecision.Text) * 100))) + "_" + Trim(Str((val(txtRecall.Text) *
100)))
End Sub
```