Improving the Scalability of Reduct Determination in Rough Sets

Shahid Mahmood

Department of Computer Science

Submitted in partial fulfillment
of the requirements for the degree of

Master of Science

Faculty of Mathematics and Science, Brock University
St. Catharines, Ontario

Abstract

Rough Set Data Analysis (RSDA) is a non-invasive data analysis approach that solely relies on the data to find patterns and decision rules. Despite its non-invasive approach and ability to generate human readable rules, classical RSDA has not been successfully used in commercial data mining and rule generating engines. The reason is its scalability. Classical RSDA slows down a great deal with the larger data sets and takes much longer times to generate the rules.

This research is aimed to address the issue of scalability in rough sets by improving the performance of the attribute reduction step of the classical RSDA - which is the root cause of its slow performance. We propose to move the entire attribute reduction process into the database. We defined a new schema to store the initial data set. We then defined SQL queries on this new schema to find the attribute reducts correctly and faster than the traditional RSDA approach.

We tested our technique on two typical data sets and compared our results with the traditional RSDA approach for attribute reduction. In the end we also highlighted some of the issues with our proposed approach which could lead to future research.

Contents

List of Tables

List of Figures

List of Symbols and Abbreviations

Abbreviation	Description
RSDA	Rough Set Data Analysis
GA	Genetic Algorithm
DM	Data Mining
SQL	Structured Query Language

Acknowledgements

I acknowledge the discussion and supervision from my first supervisor - Professor Ivo Düntsch. It was the result of those early discussion with Prof. Düntsch that I came up with the motivation of improving the scalability of the Rough Set Algorithm.

I also acknowledge my current supervisors - Professor Brian Ross and Professor Michael Winter and Graduate Director - Professor Sheridan Houghten for their help, guidance and support. Prof. Ross's and Prof. Winter's supervision and ideas steered me into the right direction and helped me to finish my thesis in time.

I also like to acknowledge my mother and brothers praying and wishing me back home especially my wife NIDA (Non Invasive Data Analysis - Its a co-incident) for suffering through with me during this work and supporting and praying for me all the time.

In the end, I like to dedicate my endeavor and my thesis to my son Syed Shameer Mahmood in anticipation and hope that one day he will do a better job than his dad.

Chapter 1

Introduction

1.1 Introduction

This research will primarily focus on Rough Set Data Analysis (RSDA) and its issue of scalability. RSDA was developed by Z. Pawlak and his co-workers in early 1980s [19]. This technique has since been greatly researched and to date there are more than 1300 papers published related to it. At its core, RSDA is largely based on three basic operations:

- Attribute reduction
- Rule generation
- Prediction

Until recently RSDA had been considered as part of the soft computing. However now it is also regarded as a non-traditional AI technique. Unlike other soft computing techniques that require additional model assumptions like prior probabilities, degrees of belief or fuzzy functions, RSDA does not rely on any assumptions or external parameters. Due to this property, RSDA is considered as a non-invasive data analysis - a data analysis that relies on the data only. This property of not relying on the external parameters and assumptions makes the RSDA a technique of choice for rule generation in critical system like medical and financial data where a small external assumption can have a big impact in the final outcome of the analysis. The motto of the RSDA is "Let the Data Speak for Themselves".

1.2 Motivation

Despite of the non-invasive approach of RSDA, it has still not been used to a great extent in industry for rule generation and prediction. This primarily is due to issue of its scalability. Scalability is the property of a program or system that if the complexity of inputs from a specified set of inputs considered increases the runtime behavior of the program increases just moderately, e.g. polynomial with a low degree. In case of rough set algorithms, this is not the case. The classical rough set algorithm tends to slow down as the data set gets bigger. The slowness primarily occurs during the process of attribute reduction or finding reducts. This research is primarily aimed to address the issue of scalability in the rough set algorithm. We will only concentrate on the improvement of the reduct finding in the rough set to improve its performance.

1.3 Proposed Solution

The solution that we are proposing is to move the whole reduct finding process to the database instead of finding them in memory in case of the traditional rough set theory (which creates a discernibility matrix to find the reduct). We propose to create a new schema to store the original data set in the database tables and then define the SQL queries on the proposed schema to find the reducts correctly and efficiently. This way the whole process of reduct finding is externalized and be done separately from the main rough set algorithm.

1.4 Components of the Thesis

This thesis will be comprised of 8 Chapters. The rest of the thesis presents the research about the related algorithms, the rough set algorithm and our proposed approach and solution. The Chapters and their contents are as following:

- Chapter 1: Introduction
- Chapter 2: Review of Related Work - Presents the research about the other contemporary data mining and rule generation techniques like decision trees, naive bayes and genetic algorithm.
- Chapter 3: Review of Rough Set Data Analysis - Presents the detailed analysis of the RSDA.

- Chapter 4: Preliminary Attempts to Improve Rough Set Scalability - This chapter presents our proposed database schema and preliminary attempts to arrive at our final solution for reduct finding in the database.
- Chapter 5: An Improved Reduct Determination Algorithm - This chapter presents the final version of our solution to the attribute reduct finding in the database.
- Chapter 6: Testing and Results - In this chapter we present the tests that we have performed on two different datasets and present the results in comparison with the traditional discernibility matrix reduct finding approach.
- Chapter 7: Discussion on the Results - This chapter discusses the results of the tests done in the chapter 6 and present a comparative discussion on the results for the two data sets.
- Chapter 8: Conclusion and Future Research - This chapter will conclude our thesis and also highlight the areas where future research can be performed to make our proposed solution workable for most of the data sets.

Chapter 2

Review of Related Work

2.1 Introduction

This chapter details a comparison of the RSDA technique with three most widely used contemporary machine learning techniques, i.e., decision tree, naive bayes and genetic algorithm. These three techniques are picked for the comparison with the rough set approach because all these techniques are also capable of producing the human readable if-then type rules. The techniques are explained separately with the emphasis on the rule generation. For a better understanding and consistency, the same example data set is used in the description of all the techniques including the RSDA. The example data is given in the following table:

Case Type	Price	Guarantee	Sound	Screen	d
1	high	24 months	stereo	76	high
2	low	6 months	mono	66	low
3	low	12 months	stereo	36	low
4	medium	12 months	stereo	51	high
5	medium	18 months	stereo	51	high
6	high	12 months	stereo	51	low

Table 2.1: Television sets

The above table shows a simple *Information System I*. An information system $U = \langle U, \Omega, V_q, f_q \rangle_{q \in \Omega}$ is a structure that consists of the following:

1. A finite set U of objects.
2. A finite set Ω of attributes.
3. For each $q \in \Omega$,
   - A set V_q of attribute values.
   - A mapping function $f_q : U \rightarrow V_q$

The information system is represented as a matrix because both U and Ω are finite. The table shows the information system I with 6 objects types (1 to 6) and four attributes namely Price, Guarantee, Sound and Screen. Also in the table there is one more column labeled as "d". This new attribute "d" comes with its own set of values V_d and a information mapping function $d : U \rightarrow V_d$. We relate the values of an object with respect to attributes of Ω to its value with respect to d. Therefore the new attribute is called as dependent or decision attribute while the $q \in \Omega$ are called as independent or conditional attributes. This new structure $D = \langle I, d \rangle$ is called as a *decision system*.

2.2 Decision Trees

Decision tree learning is one of the most widely used technique for inductive inference. Decision tree learning is a method for approximating discrete valued target functions represented as a tree structure. It is important to understand that the decision trees approximate only discrete valued target functions therefore can be represented as a human readable if-then rules. This feature of the decision trees alone has helped this technique to be the most widely used.

The application of the decision trees is broadly ranged from medical diagnosis applications to assessment of the credit risk in the financial industry. There are quite a few decision tree algorithms that have been developed but all of them are based on the core algorithm of a top-down greedy search through the space of the possible decision trees. In this introduction we will explain the two most popular of these algorithms, i.e., ID3 and C4.5.

2.2.1 Decision Tree Representation

As the name suggests, decision tress are represented as "trees". From the set of attributes A, an $a \in A$ is selected as the root node. From the root node the development of the tree starts. Every node in a decision tree specifies a test of

some attribute of the instance and each subtree from that node corresponds to one of the possible values for that attribute. From the root node, an instance is classified by testing the attribute specified by that node. Then moving down that tree branch corresponding to the value of the attribute in the given example. The process is repeated for the subtree rooted at the new node. In general, a decision tree represents a disjunction of conjunctions of constraints on the attribute values of the instances. Each path from the root node to the leaf corresponds to the conjunction of the attribute tests and the tree itself is disjunction of those conjunctions.

The construction of the decision tree starts from the root node and then subsequent nodes are selected for the subtrees. The selection of an attribute at the root node level and then at every subsequent level is dependent on the "information gain" of every attribute [18]. The information gain of an attribute is the measure of the expected reduction in entropy. Given a collection A of all the positive and negative examples of some target attribute, the information gain $Gain(S, A)$ of an attribute A, relative to a collection of examples S is defined as:

$$Gain(S, A) = Entropy(S) - \sum_{v \in values\ (A)} \frac{|S_v|}{|S|} Entropy(S_v) \tag{2.1}$$

where values(A) is the set of all possible values for the attribute A, S_v is the subset of S for which the attribute A has the value v. If the target atribute can take c values then the entropy of S relative to target is given by the following formula:

$$Entropy(S) = \sum_{i=1}^{c} - p_i \log_2 p_i \tag{2.2}$$

In the above formula p_i is the proportion of S belonging to class i that is induced by one of the target attributes.

Consider the television set information system given in Table 2.1. Suppose S is a collection of the training examples described by the attributes in the TV set information system. To select the root node, we have to calculate the information gain for every attribute using the Equation 2.1 and then an attribute with the highest information gain is selected as a root element.

To calculate the information gain of an attribute, first we have to calculate the $Entropy(S)$ of all the whole system according to the decision or target attribute. By using the Equation 2.2, we can do that as follows:

In our TV set information system, with respect to the decision attribute, there are three positive(3+) and three negative(3-) examples. Having equal number of positive and negative examples mean that the entropy will be equal to 1. Lets calculate it using the Equation 2.2.

$$\begin{aligned} Entropy([3+,3-]) &= -(\frac{3}{6})\, log_2\, (\frac{3}{6}) - (\frac{3}{6})\, log_2\, (\frac{3}{6}) \\ &= (\frac{1}{2}) + (\frac{1}{2}) \\ &= 1 \end{aligned}$$

Now lets calculate the information gain of the individual attribute. Here the calculations are shown for the Guarantee attribute. The rest are self explanatory.

$$\begin{aligned} values(Guarantee) &= 6mths, 12mths, 18mths, 24mths \\ S &= [3+,3-],\ Entropy(S) = 1 \\ S_{6mths} &= [0+,1-],\ Entropy(S_{6mths}) = 0 \\ S_{12mths} &= [1+,2-],\ Entropy(S_{12mths}) = 0.9183 \\ S_{18mths} &= [1+,0-],\ Entropy(S_{18mths}) = 0 \\ S_{24mths} &= [1+,0-],\ Entropy(S_{24mths}) = 0 \end{aligned}$$

Now the information gain for the "Guarantee" can be calculated using equation 2.1 as:

$$\begin{aligned} Gain(S, Guarantee) &= 1 - (\frac{1}{6})0 - (\frac{3}{6})0.9183 - (\frac{1}{6})0 - (\frac{1}{6})0 \\ &= 0.5085 \end{aligned}$$

The calculation above shows that the attribute Guarantee's information gain value is 0.5085. Similarly the information gain for all the other attributes i.e. price, screen and sound can be calculated using Equation 2.2. The calculated information gain for other attributes is:

$$\begin{aligned} Gain(S, Price) &= 0 \\ Gain(S, Sound) &= 0 \\ Gain(S, Screen) &= 0.5085 \end{aligned}$$

From the information gain values of all the attributes, we can see that either the Guarantee or the Screen could be selected as the root node. Lets say we

select Guarantee as the root node. The process of selection for the root of then every subsequent sub-tree continues with the remaining attributes and training examples until every node ends up as the leaf node. If we draw the decision tree using the method of calculation given above and for the TV sets information system, it would look like the Figure 2.1. The diamonds represent the leaves while the ovals represent the nodes.

For a customer having a "high" probability of buying a TV set, the decision would look like as:

$$(Guarantee = 18mths) \vee (Guarantee = 24mths) \vee (Guarantee = 12mths \wedge Price = medium) \vee (Guarantee = 12mths \wedge Price = high \wedge Screen = 76) \vee (Guarantee = 12mths \wedge Price = high \wedge Screen = 51 \wedge Sound = stereo)$$

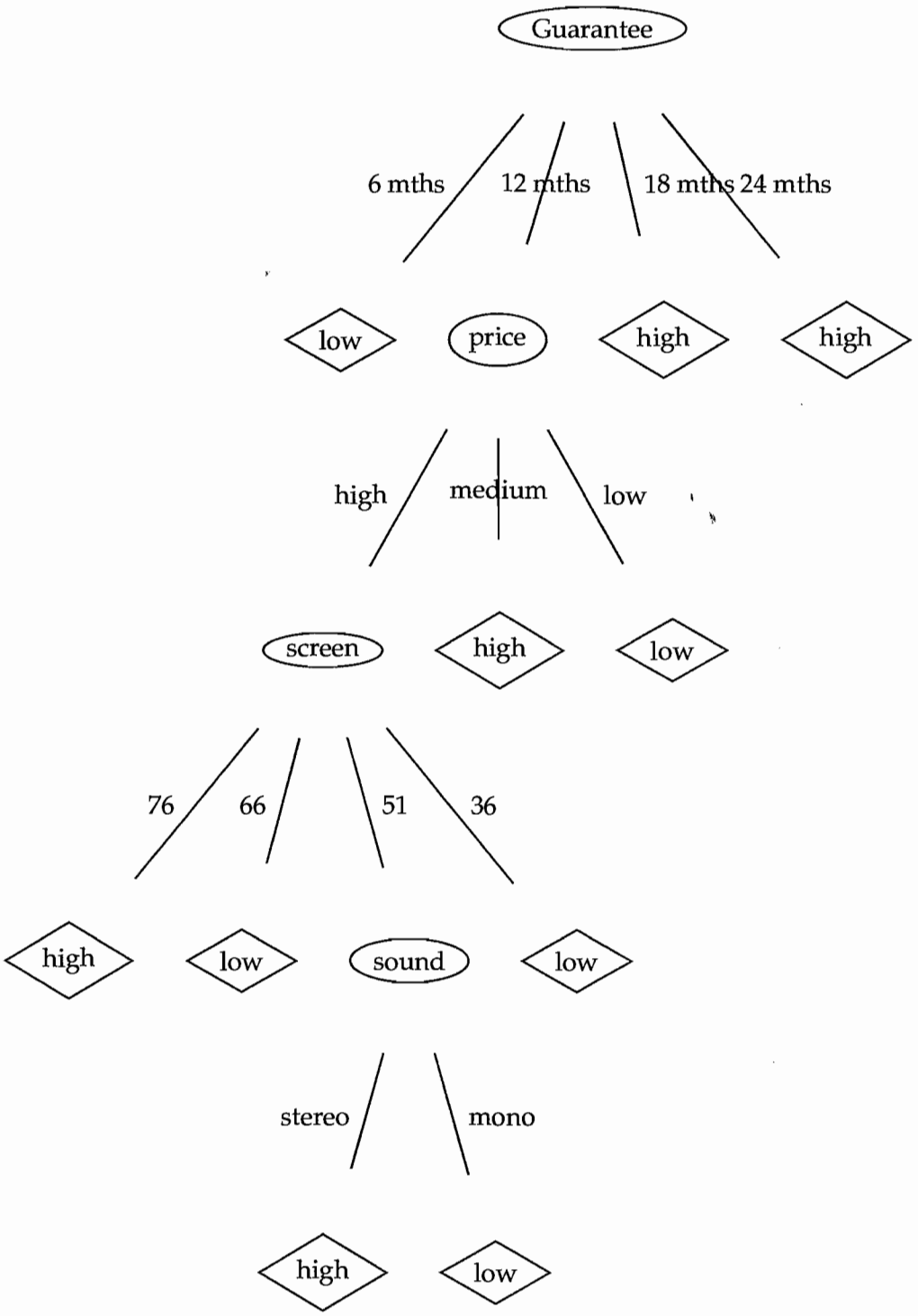

Figure 2.1: Television Sets Decision Tree

2.2.2 ID3

ID3 is based on the core decision tree algorithm that employs a top down greedy search through the space of the possible decision trees [18]. ID3 constructs the top down tree starting with the fundamental question: "Which attribute be the root element?". For the selection of the root element, each element is evaluated to determine how well it would classify the training examples. This evaluation is done by calculating the information gain of every attribute. The attribute with the highest information gain is selected as the root element. A descendant of the root node is then created for each possible values of this attribute, and the training examples are then sorted to the appropriate descendant node.

The process of selecting a new attribute and partitioning the training examples is repeated for each non-terminal descendant node but only training examples that are associated with that descendant node are incorporated instead of using all the training examples. The algorithm does not include those attributes that are already been dealt with higher up in the tree, therefore every attribute appears only once in the tree. This process of excluding the attributes at the higher node and incorporating the next best attribute continues for each new leaf node until one of the following two conditions are met:

1. Every attribute has already been included along this path through the tree.
2. All the training examples associated with this leaf node have the same target value i.e. their entropy is zero.

This process forms a greedy search for an acceptable decision tree in which the algorithm never backtracks to reconsider earlier choices. ID3 like any other inductive learning algorithms is characterized as searching a space of hypotheses for the one that fits the training examples. The hypotheses space of ID3 is the set of all the possible decision trees. In this search, ID3 performs a simple-to-complex hill climbing search starting from an empty tree and the progressively search for a decision tree that correctly classifies the training data. The evaluation function of this hill climbing search is the information gain measure.

Issues in ID3

Despite its popularity and the practical use in many different applications, ID3 decision tree algorithm suffers from some major issues. Some of the most noticeable are the following:

1. Over-fitting of the training data. A hypothesis $h \in H$ is said to over-fit the training data if there exists some alternative hypothesis $h' \in H$, such that h has smaller error than h' over the training examples, but h' has smaller error than h over the entire distribution of instances. The most common causes of over-fitting are:

   - When there is noise in the data or
   - when the number of training examples is too small to produce a representative sample of the true target function

   Common heuristics to avoid over-fitting are:

   - Don't try to fit all examples, stop before the training set is exhausted
   - Fit all examples then prune the resultant tree

2. Choice of the appropriate attribute selection measure
3. Handling missing values
4. Handling attributes with the different cost
5. Scalability.

2.2.3 C4.5

C4.5 is the descendant of the ID3 algorithm. Like ID3, C4.5 also generates classifiers expressed as a decision trees. However, C4.5 is different from ID3 in the selection of the node. At every level, C4.5 selects one attribute of the sample data that most effectively splits its elements into subsets belonging to one class or another. C4.5 uses normalized information gain known as "gain ratio" which takes number and size of branches into account when choosing an attribute. The gain ratio is calculated by dividing the information gain by the "intrinsic info". The intrinsic info is the calculation of the entropy of distribution of instances into branches.

For example in our TV set example, the intrinsic info of the attribute Guarantee is the sum of the entropies for its different values.

$$\begin{aligned} \mathit{Intrinsic\ Info}(\mathit{Guarantee})\ [1,3,1,1] &= 0 + 0.9183 + 0 + 0 \\ &= 0.9183 \end{aligned}$$

Now the gain ratio for the "Guarantee" will be:

$$\begin{aligned} \textit{Gain Ratio}(\textit{Guarantee}) &= \textit{Information Gain}(\textit{Guarantee})/\textit{Intrinsic Info}(\textit{Guarantee}) \\ &= 0.5085/0.9183 \\ &= 0.5537 \end{aligned}$$

The same calculation follows for the other attributes and the one with the highest gain ratio is selected as a node.

C4.5 has a few base cases.

- All the samples in the list belong to the same class. When this happens, it simply creates a leaf node for the decision tree saying to choose that class.
- None of the features provide any information gain. In this case, C4.5 creates a decision node higher up the tree using the expected value of the class.
- Instance of previously-unseen class encountered. Again, C4.5 creates a decision node higher up the tree using the expected value.

Another major difference and, in fact an enhancement to ID3, is the process of rule-post-pruning in C4.5 [21]. This procedure helps avoid the problem of over-fitting in ID3. The method for rule-post-pruning is given below:

1. Develop the initial (un-pruned) decision tree based on the complete training data even allowing over-fitting.
2. Each path from the root of the tree to the leaf becomes a rule. Convert the initial tree into a set of such rules.
3. Each rule is then simplified or pruned by removing any precondition in the rule. Here I mention that only those preconditions that result in the improvement of the estimated accuracy of the rule are discarded.
4. Sort the pruned rules by their estimated accuracy as a default class of rules in chosen.

2.3 Naive Bayes Classifier

Our second classification algorithm in comparison with RSDA is naive Bayes classifier. Naive Bayes classifiers are among the most practical approaches applied to most of the learning problems. Naive Bayes classifiers became popular because of the fact that it is easy to construct and does not require any complicated iterative parameter estimation schemes. Also the rules produced by the naive Bayes classifiers are easy to interpret. Before I explain naive Bayes classifiers, it is important to understand the underlying Bayesian learning method.

2.3.1 Bayesian Learning

Bayesian learning is based on the assumption that probability distribution governs the quantities of interest and carrying out reasoning about these probabilities together with the observed data can create optimal decisions. According to [18], the basic features of the Bayesian learning methods include the following:

1. Every training example incrementally increase or decrease the estimated probability that a hypothesis is correct.
2. Every Bayesian learning method is required to know the prior probability and the probability distribution for each possible candidate algorithm over the observed data.
3. Bayesian learning methods allow probabilistic predictions about hypotheses.
4. This method also allows that new instances can be classified by combining the predictions of multiple hypotheses, weight by their probabilities.

2.3.2 Bayes Theorem

Bayes theorem provides a way to find the most probable hypothesis, given the data D and any initial knowledge about the hypothesis in the hypotheses space. Mathematically, Bayes theorem is defined as:

$$P(h/D) = \frac{P(D/h)\,P(h)}{P(D)} \tag{2.3}$$

where $P(h/D)$ is called a the *posterior* probability of hypothesis h given the data D. $P(D/h)$ denotes the probability of the observing data D given some space in

which the hypothesis h holds. $P(h)$ is known as the *prior* probability of h which reflects the background knowledge that h is the correct hypothesis. $P(D)$ denotes the *prior* probability about the training data D.

In most of the decision learning problems, the goal is to find the most probable hypothesis $h \in H$ given the observed data D. Such a maximally probable hypothesis is called maximum a posteriori (MAP) hypothesis. These hypotheses can be determined by using the Bayes theorem which calculates the posterior probability of each candidate hypothesis. Mathematically, MAP hypothesis denoted as h_{MAP} is calculated as:

$$\begin{aligned} h_{MAP} &= argmax_{h \in H} \; P(h/D) \\ &= argmax_{h \in H} \; \frac{P(D/h)P(h)}{P(D)} \end{aligned} \tag{2.4}$$

We can remove the $P(D)$ from the Equation 2.4 as it is the probability of the give data D and is a constant independent of h. Therefore the h_{MAP} becomes:

$$h_{MAP} = argmax_{h \in H} \; P(D/h) \; P(h) \tag{2.5}$$

2.3.3 Naive Bayes Classifiers:

Naive Bayes classifiers are based on the Bayesian theorem. These classifiers applies to the learning tasks where each instance x is described as the conjunction of attribute values and where the target function $f(x)$ can take on any value from the finite set V.

When a new instance presented as a tuple of attribute values $< a_1, a_2, ..., a_n >$, the naive Bayes classifier predicts the target value for this new instance based on the learning on the training examples. Using the Bayesian theorem, classification of the new instance described by the attribute values $< a_1, a_2, ..., a_n >$ is given as:

$$V_{MAP} = argmax_{v_j \in V} \; P(v_j | a_1, a_2, ..., a_n) \tag{2.6}$$

Elaborating the equation using the Equation 2.6, we get

$$V_{MAP} = argmax_{v_j \in V} \; \frac{P(a_1, a_2, ..., a_n | v_j) \; P(v_j)}{P(a_1, a_2, ..., a_n)} \tag{2.7}$$

$$V_{MAP} = argmax_{v_j \in V} \; P(a_1, a_2, ..., a_n | v_j) \; P(v_j) \tag{2.8}$$

Naive Bayes classifier simplifies the above equation by assuming that the attribute values are conditionally independent given the target value. Therefore

the probability of conjunction $< a_1, a_2, ..., a_n >$ is just the product of probabilities for the individual attributes i.e. $P(a_1, a_2, ..., a_n|v_j) = \Pi_i \ P(a_i|v_j)$. Using this in Equation 2.8 we get:

$$V_{NB} = argmax_{v_j \in V} \ \Pi_i \ P(a_i|v_j) \ P(v_j) \tag{2.9}$$

where V_{NB} denotes the target value output by the naive Bayes classifier.

In summary, the naive Bayes learning method involves the learning from the training examples in the form of estimation of $P(v_j)$ and $P(a_i|v_j)$. The set of these estimates correspond to the learned hypothesis. This learned hypothesis is then used to classify new instances by applying the Naive Bayes classifier as given in Equation 2.9.

2.3.4 Naive Bayes Classifier Example:

Let us explain the naive Bayes classifier using the television buying decision example. We will use the same data as in the decision tree example. Suppose we have the following instance to classify using the Naive Bayes classifier.
Price = high, Guarantee = 18 months, Sound = stereo, Screen = 51
We will use naive Bayes classifier to predict the target value (*high or low*) of the chances of buying the television. Using the naive Bayes equation, we have

$$V_{NB} = argmax_{v_j \in (high, low)} \ \Pi_i \ P(a_i|v_j) \ P(v_j) \tag{2.10}$$

Which will become as following once we use the data from an instance of the television buying information system:

$$\begin{aligned} V_{high\ or\ low} \ = \ & argmax_{v_j \in (high, low)} \ P(v_j) \ P(price = high|v_j) \\ & P(Guarantee = 18mths|v_j) \ P(sound = stereo|v_j) \\ & P(screen = 51|v_j) \end{aligned}$$

To assign this instance to either *high* or *low* target value, we calculate V_{high} and V_{low} for this particular instance. We will need 10 probabilities that can be estimated from the training data. Let us first calculate the individual $P(v_i)$ i.e.

$$\begin{aligned} P(Buying\ television = low) \ &= \ \frac{3}{6} = 0.50 \\ P(Buying\ television = high) \ &= \ \frac{3}{6} = 0.50 \end{aligned}$$

Now we estimate the conditional probabilities.

$$\begin{aligned}
P(price = high \mid buying\ television = low) &= \frac{1}{3} = 0.33 \\
P(price = high \mid buying\ television = high) &= \frac{1}{3} = 0.33 \\
P(guarantee = 18mths \mid buying\ television = low) &= \frac{0}{3} = 0 \\
P(guarantee = 18mths \mid buying\ television = high) &= \frac{1}{3} = 0.33 \\
P(sound = stereo \mid buying\ television = low) &= \frac{1}{3} = 0.33 \\
P(sound = stereo \mid buying\ television = high) &= \frac{3}{3} = 1 \\
P(screen = 51 \mid buying\ television = low) &= \frac{1}{3} = 0.33 \\
P(screen = 51 \mid buying\ television = high) &= \frac{2}{3} = 0.66
\end{aligned}$$

Next we calculate the V_{high} and V_{low}. We use Equation 2.10 and calculate it for both chances of buying television being "high" and "low". For chances being high, it will be:

$$\begin{aligned}
V_{high} = {} & P(decision = high) * P(price = high \mid high) * \\
& P(guarantee = 18mths \mid high) * P(sound = stereo \mid high) * \\
& P(screen = 51 \mid high)
\end{aligned}$$

We have already calculated these probabilities, therefor inputting their values we get:

$$\begin{aligned}
V_{high} &= 0.5 * 0.33 * 0.33 * 1 * 0.66 \\
&= 0.037
\end{aligned}$$

Similarly for chances being low, the probabilities would be

$$\begin{aligned}
V_{low} = {} & P(decision = low) * P(price = high \mid low) * \\
& P(guarantee = 18mths \mid low) * P(sound = stereo \mid low) * \\
& P(screen = 51 \mid low)
\end{aligned}$$

Inputting the calculated probabilities values we get:

$$\begin{aligned}
V_{low} &= 0.5 * 0.33 * 0 * 0.33 * 0.3 \\
&= 0
\end{aligned}$$

Therefore the naive Bayes classifier will assign this new instance as a "high" chance of buying a television.

2.4 Genetic Algorithm

The genetic algorithms (GA) constitute a learning approach that is based on the simulation of the natural evolution. GAs search for the best hypothesis in the search space using the concept of biological evolution [18]. Hypotheses in genetic algorithms are encoded as strings and their interpretation depends on the application. The search for the appropriate hypothesis starts from an initial population consisting of a collection of hypotheses. This initial population of hypotheses is created randomly with say 100 or more hypotheses. The initial population then gives rise to the next generation by means of the genetic operations like mutation and crossover. At every generation, a fraction of the population members go through a fitness test and the most fit hypothesis have the better chance to participate in the development of the next generation.

2.4.1 Hypothesis Representation

The first step that we have to consider while applying a genetic algorithm to a problem is the representation of the hypothesis. In GAs, the hypotheses are represented as bit strings. The bit strings are used because the genetic operators like mutation and crossover can easily be applied on them. As we are doing the comparative analysis of different techniques with the rough set approach, we need to have a representation of if-then-else type rules. The representation of such rules as strings can be complex. A simple representation of such rules in GA system were given by De Jong et al. [6]. They suggested to represent these rules by choosing an encoding that would allocate specific substrings for each of the rules precondition and the postcondition.

To understand the representation of such rules, let us consider our television set buying information system. The information system consists of the four attributes i.e. Price, Guarantee, Sound and Screen. Let us consider on of these attributes, for example Price. The attribute price can take on three possible values i.e. low, medium and high. This attribute can be represented by a bit string of the length three in which each bit corresponds to one of the possible three values. Placing a 1 in some position shows that the attribute is allowed to take on the corresponding value. Therefore 100 means that the price is low

and 010 means that the price is medium. The is representation can be further generalized to show that the value 011 means that the price can be either medium or high. Using this method of representation, we can represent the conjunction of constraints on multiple attributes by concatenating the corresponding bit strings.

Similarly we can represent the decision variable d as 2 bit string for the possible values of low or high. But here we could possibly face an issue with this type of representation because if for some hypotheses, we end up with 11 representation of decision variable, then it means that regardless of the preconditions, the postcondition does not constraint the target attribute of buying T.V. To avoid this issue, we can use the single bit representation of the decision variable where 1 means *yes* or *high* and 0 means *No* or *Low*. Now if we have a rule like:

If Guarantee = 12 months and Price = medium then decision = high
can be represented as:

Price	*Guarantee*	*Sound*	*Screen*	*d*
010	0100	11	1111	1

And for a rule like:

If Guarantee = 18 months then decision = high
is represented as:

Price	*Guarantee*	*Sound*	*Screen*	*d*
111	0010	11	1111	1

Note that in the above examples, say if a rule does not use all the attributes, its representation will still incorporate the bit strings of all attributes. This is achieved by assigning value 1 for every possible position in the bit string for this attribute. In fact, this assignment indicates that the rule does not care about the value of the attribute. The main advantage of using all the attributes in the representation of the hypothesis as a bit string is that it ensures the fixed length representation of the hypothesis which makes it easier especially in the crossover operation. Given this representation method, we can represent all the hypothesis and start with the initial random population of hypotheses and then by using a GA to search for a good hypothesis.

2.4.2 Genetic Operators

There are two types of genetic operators applied to the hypothesis in GAs mutation and crossover. For mutation we suggest standard one bit mutation i.e. choosing a bit at random and replace it with its complement.

In case of if-then hypothesis rules, we use a two point crossover. This crossover is done by selecting two random points between 1 and the length of the hypothesis. We mark the two parents with these points and then create two offspring from them. The first child hypothesis is created by taking the first unmarked piece of the first parent and the marked piece of the second parent and then add the second unmarked piece of the first parent again. The same process repeats for the second child but starting from the second parent. we illustrate it by the an example. Consider the two rules in section 2.4.1 and we randomly select 2 and 9 as the two points for our crossover. Using these crossover points, the parent hypotheses will look like as:

Price	*Guarantee*	*Sound*	*Screen*	*d*
010	0100	11	1111	1

Price	*Guarantee*	*Sound*	*Screen*	*d*
111	0010	11	1111	1

Now after applying the two point crossover, the first child hypothesis will be:

Price	*Guarantee*	*Sound*	*Screen*	*d*
011	0010	11	1111	1

And the second child hypothesis will be represented as:

Price	*Guarantee*	*Sound*	*Screen*	*d*
110	0100	11	1111	1

2.4.3 The Fitness Function

The fitness function plays the key role in the application of GAs. The fitness function defines the criteria for ranking the potential hypotheses and the probability of selecting them to be included in the next generation. In case of complex procedures like if-then rules representation as bit strings, the fitness function measures the overall performance of the resulting procedure rather than the

performance of the individual rules as in the case of the classification rules. In [18], Mitchell suggested that the fitness for such hypothesis is based on its classification accuracy over the training data. Therefore the fitness for a hypothesis h is:

$$Fitness(h) = (correct(h)^2) \tag{2.11}$$

where correct(h) is the number of all the training examples correctly identified by the hypothesis h.

2.4.4 Selection

There are different types of selection used for the GAs such as roulette-wheel or fitness proportionate selection, rank selection and tournament selection [13]. For the if-then rule hypotheses population, Mitchell [18] suggested to use the proportionate or roulette-wheel selection. As mentioned in the last section that our fitness function is based on the overall performance of the resulting procedure rather than the performance of the individual rules, we employ the same idea here and the selection of the hypothesis will not only be dependent on its own fitness but will also be dependent on the fitness of the competing hypotheses. Therefore the probability of a hypothesis h_i to be selected for the next generation is given as:

$$Pr(h_i) = \frac{Fitness(h_i)}{\sum_{j=1}^{n} Fitness(h_j)} \tag{2.12}$$

This means that the probability of the hypothesis h_i to be selected is directly proportional to its own fitness and inversely proportional to the fitness of its competing hypotheses.

Chapter 3

Review of Rough Set Data Analysis

3.1 Introduction

This chapter continues with our discussion of the comparative analysis of different machine learning and data mining techniques with RSDA. Once again to keep the consistency of our discussion in line with decision trees, naive Bayes and genetic algorithm in the last chapter, we will use the same example data of television buying decision table. The example data is given in the following table:

Case Type	Price	Guarantee	Sound	Screen	d
1	high	24 months	stereo	76	high
2	low	6 months	mono	66	low
3	low	12 months	stereo	36	low
4	medium	12 months	stereo	51	high
5	medium	18 months	stereo	51	high
6	high	12 months	stereo	51	low

Table 3.1: Television sets

3.2 Introduction to Rough Set Data Analysis

The last technique that we will focus in the our discussion on different soft computing and Artificial Intelligence techniques for data mining is the rough set data analysis. Rough set data analysis (*RSDA*) was developed by Pawlak and his co-workers in early 1980s [19]. This technique has since been greatly researched and to date there are more than 1300 papers published related to it. At its guts, RSDA is largely based on three basic operations:

- Attribute reduction
- Rule generation
- Prediction

Until recently RSDA had been considered as part of the "soft computing field", however in [10], it is also regarded as a non-traditional AI technique. Unlike other soft computing techniques that require additional model assumptions like prior probabilities, degrees of belief or fuzzy functions, RSDA does not rely on any assumptions or external parameters. Due to this property, RSDA is considered a a non-invasive data analysis - a data analysis that relies on the data only. A lot of research material has been published on this non-invasive property of RSDA [8, 10, 11]. This property of not relying on the external parameters and assumptions makes the RSDA a technique of choice for rule generation in critical systems like medicine [12, 23, 24] and financial data [4, 16] where a small external assumption can have a big impact in the final outcome of the analysis.

3.2.1 Basic Model of RSDA

The basic model of RSDA is based on assumption (which is not external) that the granularity of the data can be expressed as partitions and their associated equivalence relations on the set of the objects; such a relationship is called as the indiscernability relation. Before we explain RSDA, let us understand the concept of partition, equivalence relation and eventually an indiscernability relation.

Partition

Suppose that U is a non-empty set. A partition is a family P of non-empty subsets of U such that "each element of U contained in exactly one element of P".

Equivalence Relation

A equivalence relation θ on U is a binary relation that is reflexive, symmetric and transitive, i.e., if $x, y, z \in U$, then by definition an equivalence relation θ is represented as:

$$x \theta x \ (reflexitivity)$$
$$x \theta y \text{ implies } y \theta x \ (symmetry) \text{ and}$$
$$x \theta y \text{ and } y \theta z \text{ imply } x \theta z \ (transitivity)$$

Each partition P induces an equivalence relation θ on U as:

$$x \theta y \Leftrightarrow x \text{ and } y \text{ are in the same class of } P.$$

Conversely each equivalence relation θ on U induces a partition P of U with classes θ_x as:

$$\theta_x = y \in U : x \theta y$$

To explain the indiscernibility relation, first we need to explain the representation of data or knowledge in RSDA. The data for RSDA is usually represented in the form of a matrix as what is called as an information system.

Information System

An information system $U = \langle U, \Omega, V_q, f_q \rangle_{q \in \Omega}$ is a structure consists of the following:

1. A finite set U of objects.
2. A finite set Ω of attributes.
3. For each $q \in \Omega$,
   - A set V_q of attribute values.
   - A mapping function $f_q : U \rightarrow V_q$

The information system is represented as a matrix because both U and Ω are finite. To show an example information system, we will consider the television set buying data in Table 3.1 that we have been using in the explanation of other techniques in this chapter. The table shows the information system I with 6 objects types (1 to 6) and four attributes namely Price, Guarantee, Sound and Screen.

Also in the table there is one more column labeled as "d". This new attribute "d" comes with its own set of values V_d and a information mapping function $d : U \rightarrow V_d$. We relate the values of an object with respect to attributes of Ω to its value with respect to d. Therefor the new attribute is called as dependent or decision attribute while the $q \in \Omega$ are called as independent or conditional attributes. This new structure $D = \langle I, d \rangle$ is called as a *decision system*. Now that we have understood an information system, we can define the indiscernibility relation.

Indiscernibility Relation

In an information system, two objects are indiscernible or are in indiscernibility relation if they have same values for the attributes contained in Ω i.e. they are in the same equivalence class of Q.

$$x \equiv_{\theta_Q} y \; if \; and \; only \; if \; a(x) = a(y) \; for \; all \; a \in Q$$

Set Approximation

Set approximation helps us define the crisp partitioning of objects in an information system based on the equivalence relation. In rough set theory our knowledge about the subset $\mathtt{X}$ of U is only limited to its equivalence classes θ and their unions.
So for $\mathtt{X} \subseteq U$, we say

$$\underline{\mathtt{X}}_\theta \stackrel{def}{=} \{x \in Q : \theta_x \subseteq \mathtt{X}\} \tag{3.1}$$

is the lower approximation of $\mathtt{X}$ and

$$\overline{\mathtt{X}}^\theta \stackrel{def}{=} \{x \in Q : \theta_x \cap \mathtt{X} \neq \phi\} \tag{3.2}$$

is the upper approximation of the $\mathtt{X}$. In this context we say that if $\theta_x \subseteq \underline{\mathtt{X}}$, we know for certain that $x \in \mathtt{X}$ and if $\theta_x \subseteq U \setminus \overline{\mathtt{X}}$ we know for certain the $x \notin \mathtt{X}$. The area of uncertainty lies in the region $\overline{\mathtt{X}} \setminus \underline{\mathtt{X}}$.In this area of uncertainty we are not sure about the membership of x since θ_x intersects both $\mathtt{X}$ and $U \setminus \mathtt{X}$.
A rough set of U is a pair $\langle \underline{\mathtt{X}}, \overline{\mathtt{X}} \rangle$ where $\mathtt{X} \subseteq U$. A rough set is called definable if $\underline{\mathtt{X}} = \overline{\mathtt{X}}$.

3.2.2 Attribute Reduction or Reducts

Data dependencies and the reduction of the number of attributes comprise one of the major research interests in the field of data mining. RSDA probably gives the most structural and simplified way of finding the data dependencies and attribute reduction.

Let us consider $D = \langle U, \Omega, V_q, f_q, d \rangle$ be a decision system representing the television buying scenario presented in the Table 3.1. The basic idea of attribute reduction in RSDA [10] is the comparison of the equivalence relation generated by the sets of attributes. For every $Q \subseteq \Omega$, we associate an equivalence relation θ_Q on U. From our definition of indiscernibility, we can see that if $x\ \theta_Q\ y$, that means that x and y are indiscernible with respect to the values of their attributes from Q. Also we can understand that if $P,\ Q \subseteq \Omega$ then

$$P \subseteq Q \Rightarrow \theta_Q \subseteq \theta_P$$

When $\theta_Q \subseteq \theta_P$, we say that P is dependent on Q written as $Q \Rightarrow P$.

Definition of a Reduct

A set $P \subseteq Q \subseteq \Omega$ is called a reduct of Q if:

1. $\theta_P = \theta_Q$
2. For each $R \subset P,\ \theta_R \neq \theta_Q$

A reduct is a minimal by definition which means that the attributes within a reduct are independent and none of them can be omitted. The reducts produce deterministic rules. The intersection of all the reducts of Q is called the **core** of the Q.

Finding a Reduct

There are few different ways of finding the reducts of an information system. Here we will consider a very simple method of finding the core and the reducts by creating a "discernibility matrix". Before we create such a matrix, first we define the discernibility function.

A discernibility function $\delta : U * U \rightarrow 2^{\Omega}$ is defined as:

$$\delta(a, b) \stackrel{def}{=} \{q \in \Omega : f_q(a) \neq f_q(b)\}$$

The discernibility matrix is then created by assigning to each $\langle a, b\rangle \in U^2$ the set $\delta(a, b)$ for all those attributes q for which $q(a) \neq q(b)$. Since the assignment $\langle a, b\rangle \rightarrow \delta(a, b)$ is symmetric and $\delta(a, a) = \emptyset$, we need to record only the upper triangle of the matrix. For the television buying decision system, the discernibility matrix would look as follows:

	2	3	4	5	6
1	Pr,Gu,So,Sc	Pr,Gu,Sc	Pr,Gu,Sc	Pr,Gu,Sc	Gu,Sc
2		Gu,So,Sc	Pr,Gu,So,Sc	Pr,Gu,So,Sc	Pr,Gu,So,Sc
3			Pr,Sc	Pr,Gu,Sc	Pr,Sc
4				Gu	Pr
5					Pr,Gu
6					

Table 3.2: Discernibility Matrix

Now considering the discernibility function and matrix, we propose another definition of the core and the reduct:

1. The core of I is a set

$$\{q \in \Omega : \delta(a, b)\} = \{q\} \text{ for some } a, b \in U\}$$

2. $P \subseteq \Omega$ is a reduct of I if P is minimal with respect to the property

$$P \cap \delta(a, b) \neq \emptyset$$
$$\text{for all } a,\ b \in \Omega,\ \delta(a, b) \neq \emptyset$$

By following the above definition and looking at the discernibility matrix given in Table 3.1 for the television buying information system, we can see that $\{Pr,\ Gu\}$ is the only reduct as well as the **core**.

3.2.3 Decision Reducts

The above section explains how to find the reducts of an information system. However an information system with the decision attribute is a *decisionsystem* as explained in section 3.2.1. For a decision system, we find the decision reducts. A decision reduct is a minimal set of attributes that preserves the decision making power of the original system [5]. Decision reducts are used to create the minimal

decision rules for a decision system. The decision reducts can also be created by creating a discernibility matrix as described in the last section but the matrix is created while considering the value of the decision attribute. A decision-relative discernibility matrix of a decision system defined as $D = \langle I, d \rangle$ is the $n \times n$ matrix O defined as following:
for i and j in 1, ..., n,

$$O_{i,j} = \{a \in A | a(x_i) \neq a(x_j)\}$$

If we create the decision-relative discernibility matrix for the decision system given in section 3.2.1, we can see that $\{Pr, Gu\}$ and $\{Pr, Sc\}$ are the two decision reducts and Pr is the core of the system.

3.2.4 Rule Generation

The reducts are then used to synthesize the minimal decision rules. Once the reducts are computed, the rules can then be constructed by overlaying the reducts over the original decision table and reading off the values.

Once again there are different ways for generating the decision rules from the reducts for an information system. We will discuss here a method that is based on the relations among the classes of the equivalence relations associated with the attribute sets [10]. In this method we assume that the partition induced by θ_Q is $\{\mathtt{X}_1, ..., \mathtt{X}_n\}$ and the one induced by θ_d is $\{\mathtt{Y}_1, ..., \mathtt{Y}_m\}$. For each $\mathtt{X}_i$, we associate a set $\mathtt{M} = \{\mathtt{Y}_j : \mathtt{X}_i \cup \mathtt{Y}_j \neq \phi\}$.

Using the above method one can clearly understand that because sets $\mathtt{Y}_1, ..., \mathtt{Y}_m$ partition U, then

$$if\, x \in \mathtt{X}_i,\ then\ x \in \mathtt{Y}_{j_1}\ or\ x \in \mathtt{Y}_{j_2}\ or\ ....\ or\ x \in \mathtt{Y}_{j_i(j)}$$

where $\{\mathtt{Y}_{j_1}, ..., \mathtt{Y}_{j_i(j)}\} = \mathtt{M}$. We know that each class $\mathtt{X}_i$ corresponds to the feature vector $\langle \mathtt{t}_a \rangle_{a \in Q}$ where $x \in \mathtt{X}_i$ if and only if $f_q(x) = a_1\ and\ ....\ f_{q_n}(x) = a_n$. Similarly $x \in \mathtt{Y}_j$ if and only if $f_d(x) = b_i$ for some $b_i \in V_d$.

This process gives us general form of generating decision rules of the form

$$if\ f_q(x) = a_1\ and\ ....\ f_{q_n}(x) = a_n,\ then\ f_d(x) = b_{j_1}\ or\ ....\ or\ f_d(x) = b_{j_n}$$

We denote these rules as $Q \rightsquigarrow d$.

For our television set buying example, the $\mathtt{X}_i$ are:
Pr:

$$\mathtt{X}_1 = \{a_1, a_6\},\ \mathtt{X}_2 = \{a_2, a_3\},\ \mathtt{X}_3 = \{a_4, a_5\}$$

Gu:

$$\mathtt{X}_4 = \{a_1\},\ \mathtt{X}_5 = \{a_2\},\ \mathtt{X}_6 = \{a_5\},\ \mathtt{X}_6 = \{a_3, a_4, a_6\}$$

and $\mathtt{Y}_j$ would be

$$\mathtt{Y}_1 = \{a_2, a_3, a_6\},\ \mathtt{Y}_2 = \{a_1, a_4, a_5\}\}$$

One can clearly see that $\mathtt{X}_i$ for Guarantee are more deterministic so we start constructing our rules from Guarantee and the incorporate Price in it. The decision rule using this technique would look like as:

$$If\ Guarantee \begin{cases} = 6\ months: & \text{low} \\ = 12\ months: & If\ Price \begin{cases} = medium: & \text{high} \\ = otherwise: & \text{low} \end{cases} \\ \geq 18\ months: & \text{high} \end{cases}$$

In general our decision rules have the form $\alpha \rightsquigarrow \beta$, where α is a positive boolean combination of descriptors of the form $f_{q_i}(x) = a_i$ and β is a disjunction of descriptors of the form $f_d(x) = b_j$.

3.2.5 Rule Significance

Like any inference system, we have to consider that any rule constructed using the rough set approach is not due to chance. This consideration is especially important when rules are to be used for prediction or classification of unseen data. A good method of finding rule significance is mentioned in the book [10] that presents a statistical hypothesis testing solution to this problem. The hypothesis test is done to test the claim that "a rule $Q \rightsquigarrow d$ using rough set is due to a chance". Care has to be taken that the no specific model assumptions are used which are not justified by the data at hand. The hypothesis test is as follows:

H_o: The rule $Q \rightsquigarrow d$ is due to a chance.

H_a: The rule $Q \rightsquigarrow d$ is not due to a chance.

The level of significance in our case is 0.05. The test statistics is the approximation quality γ. The test will be done by comparing γ to the approximation quality of the rules obtained by reordering randomly the feature vectors for objects in the information system I, while keeping the decision values constant.

To find the significance of the approximation quality of the rules obtained by randomly reordering the feature vectors for objects, we let $\sum$ be the set of all the permutations of U, and $\sigma \in \gamma$. we define a new attribute function a^σ by:

$$a^\sigma(x) = \begin{cases} a(\sigma(x)), & \text{if } a \in Q \\ a(x), & \text{otherwise} \end{cases}$$

The resulting information system I_σ permutes the Q-columns according to σ, while leaving the d-columns constant. We also let Q^σ be the result of the permutation in the Q-columns, and $\gamma(Q^\sigma \rightsquigarrow d)$ be the approximation quality of the new rule $Q^\sigma \rightsquigarrow d$ in the information system I_σ.

Next we measure the test statistics value as:

$$\alpha = p(\gamma(Q \rightsquigarrow d|H_o)) = \frac{|\gamma(Q^\sigma \rightsquigarrow d) \geq \gamma(Q \rightsquigarrow d):\ \sigma \in \sum|}{|U|!}$$

If $\alpha < 0.05$, the rule $Q \rightsquigarrow d$ is significant and we reject H_o(null hypothesis), otherwise the rule $Q \rightsquigarrow d$ is casual and we fail to reject H_o.

Conditional Significance:

We find the conditional significance if we have to determine the influence of one attribute on the classification success. Traditionally in rough set theory, the effect on approximation quality by omitting an attribute within reduct is the indicator of the importance of that attribute. But this approach suffers from one problem i.e. we are not sure if the decline of approximation quality is due to a chance. To test the conditional significance we again do a hypothesis test as before:

H_o: The influence of q on $\gamma(Q \rightsquigarrow d)$ is due to a chance.

H_a: The influence of q on $\gamma(Q \rightsquigarrow d)$ is not due to a chance.

Again if σ is a permutation of U and $a \in Q$, we define

$$a^{\sigma,q}(x) = \begin{cases} a(\sigma(x)), & \text{if } a = q \\ a(x), & \text{otherwise} \end{cases}$$

The test statistics will be calculated as follows:

$$\alpha = p(\gamma(Q \rightsquigarrow d|H_o, q)) = \frac{|\gamma(Q^{a_q} leadstod) \geq \gamma(Q \rightsquigarrow d)\colon \sigma \in \sum|}{|U|!}$$

Again if $\alpha < 0.05$, we reject H_o(null hypothesis), otherwise we fail to reject H_o and call q as *conditional casual.*

3.2.6 Discretization

Discretization is a process of collecting numerical values into classes such as intervals or ranges of values. For discrete numerical values, a very simple yet efficient discretization technique was presented by Powloski and Skowron [20]. This technique transforms an information system I into a new binarised information system I^*. The general idea for binarisation of an information system is as follows:

Given an information system $I = \langle U, \Omega, (V_q)_{q\in\Omega} \rangle$, for each attribute $q \in \Omega$ let V_q^+ be the set attribute values which are actually taken by some $x \in U$. For each $t \in V_q^+$, we define a new attribute function: $q^t : U \rightarrow 0, 1$ as:

$$q^t(x) = \begin{cases} 1, & \text{if } q(x) = t, \\ 0, & \text{otherwise} \end{cases}$$

we let $\Omega_q = q^t : t \in V_q^+$, and I_B be the information system with attribute $\Omega^+ = \bigcup_{q\in\Omega} \Omega_q$.
Once the system is binarised, we do the following:

1. For each $q \in \Omega$ and each $a \in V_d^+$, find all the $t \in V_q^+$ for which

$$\forall_x \in U\, [q^t(x) = 1\ implies\ d(x) = a]$$

2. Let $M^{q,a}$ be the set of all these binary attributes, and define a new binary attribute $m^{q,a}$ by

$$\begin{aligned} m^{q,a}(x) = 1 \quad &\Leftrightarrow \quad q^t(x) = 1\ for\ some\ t \in M^{q,a} \\ &\Leftrightarrow \quad max_{t\in M^{q,a}} q^t(x) = 1 \end{aligned}$$

Then we replace all the attributes $q^t, t \in M^{q,a}$ simultaneously by the attribute $m^{q,a}$. This step I found is the most important and efficient step in this technique as it collects all the attribute values of q that do not split the decision class belonging to a into a single attribute value.

This practice gives us a new information system $I^* = \langle U, \Omega^*, (V_q^*)_{q\in\Omega} \rangle$ where

$$V_q^* = \left\{m^{q,a} : a \in V_d^+\right\} \cup V_q \bigcup_{a \in V_d^+} M^{q,a}$$

Also in this new information system, for each $x \in U$, there is exactly one $t_x \in V_q^*$ such that $q^t(x) = 1$ by definition of V_q^*, and we set $q^*(x) = t_x$.
By considering the minimal reducts of this new binarised information system before collecting the attribute values can significantly reduce the number of attribute values even further but care should be taken as it is an expensive procedure.

Next the effect of this binarisation technique on the dependency structure of the original system is verified. This is done by checking if the binarisation has an effect on the significance of attributes. The verification can be done by testing two things:

1. $\gamma(Q \rightsquigarrow d) = \gamma(Q^* \rightsquigarrow d)$
   This can be proved as following:
   Suppose that $P(Q)$ and $P(d)$ is the set of classes of θ *and* θ_d. Now

   $$\gamma(Q \rightsquigarrow d) = \frac{\sum |\{X : X \text{ is a } d\text{–deterministic class of } \theta_Q\}|}{U}$$

   If Y is a class of $P(d)$, the

   $$Z = \bigcup \{X \in P(Q) : X \subseteq Y\}$$

   contains only those elements of U that contributed to the Q-deterministic part of Y. Since Z is a class of Q^*, and every d-deterministic class of θ_{Q^*} has the same form, it proves our claim.

2. $p(\gamma(Q \rightsquigarrow d|H_o)) \geq p(\gamma(Q^* \rightsquigarrow d|H_o))$
   This can be proved as following:
   We know that attribute values are identified in the filtration process, for each $R \subseteq \Omega$, every class of θ_{R^*} is a union of classes of θ_R. Thus, given any $\sigma \in \sum$, the rule $Q^\sigma \rightarrow d$ will have at least as many deterministic classes as $Q^{*\sigma} \rightarrow d$. That means $\gamma(Q^\sigma \rightsquigarrow d) \geq \gamma(Q^{*\sigma} \rightsquigarrow d)$. Similarly for every $\sigma \in \sum$, we have $\gamma(Q^{\sigma*} \rightsquigarrow d) \geq \gamma(Q^* \rightsquigarrow d)$ which means

   $$\gamma(Q^\sigma \rightsquigarrow d) \geq \gamma(Q^{*\sigma} \rightsquigarrow d) \geq \gamma(Q^{\sigma*} \rightsquigarrow d) \geq \gamma(Q^* \rightsquigarrow d) = \gamma(Q \rightsquigarrow d)$$

   which proves our claim.

The above two proofs verify that the approximation quality of a binarised information system is same as the original system and the rule significance is not worse than before.

3.2.7 Rough Entropy

Rough entropy measures proposed in [9] is a very different way of measuring the prediction quality of the prediction rules. The rationale behind this concept is that the reduct based prediction rules are only relative to the chosen reduct which only measures the uncertainty of the prediction and do not take into account the predictor variables. In order to measure to measure the prediction success of the set of predictor attributes unconditionally, the authors proposed to combine:

1. The complexity $H(Q)$ of coding the hypothesis Q.
2. The conditional coding complexity $H(d|Q)$ of d, given by the values of attributes in Q

into one measure as $H(Q \rightarrow d)$. This will be done by suitable entropy functions. Let P be a partition of U with classes X_i, $i \leq k$ each having the cardinality r_i. Here the assumption is that the elements of U are randomly distributed within the classes of P, so that the probability of element x being in class X_i is just $\frac{r_i}{n}$.

The entropy is defined as:

$$H(P) = \sum_{i=1}^{k} \frac{r_i}{n} \cdot \log_2(\frac{n}{r_i})$$

For the purpose of expression, if θ is an equivalence relation on U and P is its induced partition, then we will write $H(Q)$ instead of $H(P)$. Furthermore, if Q is a set of attributes, then we will write $H(Q)$ instead of $H(\theta_Q)$.

Entropy actually estimates the mean number of comparisons minimally necessary too retrieve the equivalence class information of a randomly chosen element $x \in U$. So entropy $H(P)$ actually measures the granularity of the partition P. If there is only one class the $H(P) = 0$ and if P corresponds to identity ω, then $H(P)$ has the maximum value $\log_2(|U|)$.

Now in order to find entropy of the set of attribute Q - $H(Q)$, we suppose that X_i, $1 \leq i \leq t$, are the classes θ_Q each having cardinality r_i. Furthermore, let $|U|$ = n and $\hat{\pi}_i$ be the probability measure associated with θ_Q, i.e.

$$\hat{\pi}_i = \frac{|X_i|}{|U|} = \frac{r_i}{n}$$

The entropy of θ_Q is

$$H(Q) = \sum_{i=1}^{t} \hat{\pi}_i \cdot \log_2 \frac{1}{\hat{\pi}_i} = \sum_{i=1}^{t} \frac{r_i}{n} \log_2 \frac{n}{r_i}$$

In order to keep only the deterministic classes of θ_Q, a new equivalence relation θ_Q^+ on the U is defined as:

$x \equiv_{\theta_Q^+} y$ if and only if x = y or there exists some $1 \leq i \leq c$ such that $x, y \in X_i$

The probability measure $\hat{\psi}_i$ associated with θ_Q^+ is:

$$\hat{\psi}_i = \begin{cases} \hat{\pi}_i, & \text{if } 1 \leq i \leq c, \\ \frac{1}{n}, & \text{otherwise} \end{cases}$$

Now we can define formally the rough entropy with respect to $Q \rightsquigarrow d$ rule as:

$$H(Q \rightsquigarrow d) = \sum_i \hat{\psi}_i \cdot log(\frac{1}{\hat{\psi}_i})$$

$H(Q \rightsquigarrow d)$ will have maximum value $log_2(n)$ when:

- θ_Q is the identity relation, and everything can be explained by Q, or
- $\gamma(Q \rightsquigarrow d) = 0$, everything is guessing

Another type of rough entropy is the normalized rough entropy(NRE) formally defined as $S(Q \rightsquigarrow d)$. For this purpose there are two situations:
- If θ_d be identity ω, so the $H(d) = log_2(n)$. Then

$$S(Q \rightsquigarrow d) = \begin{cases} 1, & \text{if } \theta_Q = \omega, \\ 0, & \text{otherwise} \end{cases}$$

- else if $H(d) < log_2(n)$, then

$$S(Q \rightsquigarrow d) = 1 - \frac{H(Q \rightsquigarrow d) - H(d)}{log_2(n) - H(d)}$$

Here S is an unconditional measure as both the complexity of the rules and the uncertainty of the predictions are combined into one measure. So if $S(Q \rightsquigarrow d) = 1$, it is the perfect prediction result while if $S(Q \rightsquigarrow d) = 0$, it is the worst case. More specifically if $S(Q \rightsquigarrow d)$ has a value near 1, the entropy is low and chosen attribute combination is favorable and if it has a value near 0, it indicates casualness.

Chapter 4

Preliminary Attempts to Improve Rough Set Scalability

4.1 Introduction

This chapter talks about the scalability in general in the field of data mining and then specifically about the scalability in the rough set approach in particular. In the last section of this chapter, I will present the proposed solution to improve the scalability in rough sets. Since its inception in the mid 80s, rough set theory has been regarded as a very powerful data mining and knowledge discovery technique. Rough Set being a non-invasive data analysis technique [10] has an edge over other data mining and knowledge discovery techniques because this feature on non-invasiveness is very critical for sensitive data like financial [12, 23, 24] and medical data [4, 16]. Also other features of the Rough Set theory like attribute reduction, discretization and output as the human readable rules lead to its overall high prediction accuracy. Despite all of its prediction accuracy and ease of use, Rough Set theory has not been very successful in its usage in the commercial data mining and knowledge discovery applications and tools. The main reason for its commercial failure is the scalability of Rough Set algorithm because Rough Set Algorithms slow down proportionally with the increase in records. Let us first discuss what scalability is and what it means in case of data mining and knowledge discovery algorithms.

4.2 Scalability

Scalability is the property of a program or system that if the complexity of inputs from a specified set of inputs considered increases the runtime behavior of the program increases just moderately, e.g. polynomial with a low degree. In case of data mining algorithms, scalability means that if the data processed by the algorithm increases, its performance should also scale accordingly or at least does not slow down exponential to the data increase. Data mining algorithms can encounter two different types of increase in the input data:

1. Column Scalability: In this case the number of attributes or columns remains static but the number of rows or records increases. If the algorithm is row scalable, then by increasing the number of rows will result in the scaling of the algorithm too and its performance will not decline proportionally to the number of rows.

2. Column Scalability: The column scalability of an algorithm means that the performance of the algorithm scales proportionally with the increase in the number of columns. Unlike the row scalability, in case of column scalability the number of rows do not stay static and they can increase too. Column scalability is more difficult to achieve than the row scalability.

4.3 Scalability of the Rough Set Algorithm

The rough set algorithm has not been commercially used in rule engines and data mining applications because of its poor scalability. Classical rough set algorithm suffers from the both row and column scalability issues. The performance related issues of classical rough set algorithm are primarily due to its integral step of attribute reduction. In the commercial world, the decision systems consist of data sets that can sometimes reach millions of rows e.g. in case of bank or credit card customer bases. Finding a reduct using a discernibility matrix in the classical rough set algorithm involves comparison of each record in the decision system with all the other records in order to create their partitions or equivalence classes. This comparison operation is the most time and space consuming operation and is the cause of poor scalability of the rough set algorithm. Increasing the number of records would result in more comparison; decreasing the performance of the algorithm. Research has found that finding all the reducts in an information or

decision system is an NP complete [7, 15] problem. Let us discuss time and space issues regarding rough set.

Due to the inherent feature of attribute reduction, the rough set algorithm requires comparison of every record in the decision table with every other record for a subset of conditional attributes to find the reduct. This process consumes a lot of time as finding a reduct has exponential complexity. Especially for large data sets the algorithm can get really slow.

Also if the comparison operation of the records in the decision table is done in the data structures inside the memory of the machine, then it would require large amount of memory and resources. The data structures used for storing the equivalence classes of the objects for a subset of conditional attributes can grow very rapidly and would require large memory.

4.4 Proposed Solutions for Better Scalability

In this research, our focus is to improve the scalability of the rough set algorithm by improving on the reduct generation step of the algorithm which is the main cause of its poor performance for the large data sets. As finding all the reducts for a data set is an NP complete [7, 15] problem, we will try to show that for some of the experimental data sets, we can improve the reduct finding process and give sub exponential performance. To improve the performance of reduct finding, we propose the following:

- Faster access of the data from the database.
- Processing of the reduct generation inside the database by defining a different schema and queries instead of finding them in the memory.

By implementing the above mentioned steps, we propose that we can help bring down the time and space needed in the reduct generation which in turn will improve the over all scalability of the rough set algorithm.

4.4.1 Faster Data Access

Every data mining algorithm and knowledge discovery technique deals with the data usually in the form of the tables with large number of records. Rough set is no different as it is also applied to the data stored in tables in the form of a decision system. Processing of a huge data sets has been an area of research for a long time. In 1996 SLIQ [17] was presented as a decision tree classifier for

the large data sets but due to the use of a memory-resident data structure that scales with the size of the training set, SLIQ had an upper limit on the number of records. In the same year 1996, the creators of SLIQ developed an improved algorithm SPRINT [22] for the processing of very large data sets in the IBM Almaden Research Center. SPRINT deals with the data sets resident on the disks and introduced parallelization but its performance was equivalent to that of a classical decision tree algorithm.

Modern data database systems like Oracle and SQL Server makes it possible to access that data lot faster with their indices and smart query optimizing engines but it is also a known fact that too many indices slow down the data access than speeding it up. Column based databases like Infobright [3]; which are more favorable for data mining due to their faster data access and intelligent optimizers can increase the performance of our proposed approach. For the purpose of this research, we used the MYSQL database.

4.4.2 Reduct Generation in Database

The second part of the solution for improving the scalability of the rough set algorithm proposes the reduct generation in the database instead of being done in memory. As we discussed earlier, reduct generation is the main cause of the poor scalability of the rough set algorithm. Reduct finding in the traditional rough set algorithm is achieved by creating a discernibility matrix and finding a reduct in the matrix. The discernibility matrix is created in memory by comparing every record in the dataset with every other record which at best could result in the $O(n^2)$ complexity. This complexity order would not be a big issue for the smaller data sets like 1000 records in that table. In that case the maximum number of comparisons would be $1000 * 1000 = 1,000,000$ comparisons. However for the large datasets like 1 million records in the table, it would be very slow to create the discernibility matrix. Also after the matrix is created, we have to verify that every entry in the dataset has an element in common with the given set we are currently checking. That has to be done for every set to find all the reducts.

This research proposes that the processing of the reduct finding should be done in the database by storing the data in the suitable schema and defining the SQL queries on it. Reduct finding, if done in the database would result in the following advantages:

- Faster processing as the network latency during accessing and retrieving the data from the database is removed.

- Less CPU and memory resources usage.
- Easy maintenance. If the whole reduct generation process occurs in the database then its maintenance or update would be lot easier as the only place to make the changes would be in the database schema or queries while the code for the rough set algorithm remains unchanged.

4.5 The Challenge

The challenge with the solution proposed above is that for its successful outcome, its implementation is not sufficient but it has to be implemented in such a way that the reduct finding process is scalable for most of the typical databases too. Shifting the reduct finding process from the memory to the database is not the complete solution because it just moves the time and space costly mechanism of reduct finding to the database. Implementation of this mechanism in a way that for the larger datasets, the reduct finding process does not get slow is the real ask for this solution.

4.6 Attempts in Reduct Determination in Databases

The second part of our proposed solution requires the process of reduct determination to be done in the database. To keep the consistency and easier understanding we used the same television buying data set that we used in the chapter 1 and 2. The data set presents a decision making scenario in which a user decides to purchase a television set based on the four attributes i.e. Price, Guarantee, Sound and Screen. The task is to find the attribute reducts for this data set in the database.

Case Type	Price	Guarantee	Sound	Screen	d
1	high	24 months	stereo	76	high
2	low	6 months	mono	66	low
3	low	12 months	stereo	36	low
4	medium	12 months	stereo	51	high
5	medium	18 months	stereo	51	high
6	high	12 months	stereo	51	low

Table 4.1: Television sets

In our example of the television buying decision system mentioned above, we need to find decision reducts using a database schema and queries.

4.6.1 Initial Attempt

STEP 1

We create a new schema, in which tables are defined for every attribute of the original decision system, and the columns for these tables are based on the possible values vector of each attribute. The character "Y" represent that the value is present for the object and the value "N" represents that it is not the case for a particular object.

Oid	High	Low	Medium	d
1	Y	N	N	high
2	N	Y	N	low
3	N	Y	N	low
4	N	N	Y	high
5	N	N	Y	high
6	Y	N	N	low

Table 4.2: Price Table

Oid	6 mos	12 mos	18 mos	24 mos	d
1	N	N	N	Y	high
2	Y	N	N	N	low
3	N	Y	N	N	low
4	N	Y	N	N	high
5	N	N	Y	N	high
6	N	Y	N	N	low

Table 4.3: Guarantee Table

Oid	Stereo	Mono	d
1	Y	N	high
2	N	Y	low
3	Y	N	low
4	Y	N	high
5	Y	N	high
6	Y	N	low

Table 4.4: Sound Table

Oid	36″	51″	66″	76″	d
1	N	N	N	Y	high
2	N	N	Y	N	low
3	Y	N	N	N	low
4	N	Y	N	N	high
5	N	Y	N	N	high
6	N	Y	N	N	low

Table 4.5: Screen Table

STEP 2

After defining all the tables we will execute following queries on the each table. In each query the results are stored in a temporary table i.e. HOLDUP and then a query on the "count" on this temporary table HOLDUP is made to ascertain if the conditional variable represented in the table is a decision reduct or not.

Test for Price

```
SELECT high,low,medium INTO HOLDUP
FROM PRICE
WHERE d = 'high'
INTERSECT
SELECT high,low,medium
FROM PRICE
WHERE d = 'low';

Next we check the COUNT of the HOLDUP table:
```

```
SELECT COUNT(*) FROM HOLDUP;

COUNT = 1 - {NOT A DECISION REDUCT}
```

Test for Guarantee

```
SELECT sixmonths,twelvemonths,eighteenmonths,
       twentyfourmonths INTO HOLDUP
FROM GUARANTEE
WHERE d = 'high'
INTERSECT
SELECT sixmonths,twelvemonths,eighteenmonths,twentyfourmonths
FROM GUARANTEE
WHERE d = 'low';
```

Next we check the COUNT of the HOLDUP table:

```
SELECT COUNT(*) FROM HOLDUP;

COUNT = 1  - {NOT A DECISION REDUCT}
```

Test for Sound

```
SELECT stereo,mono INTO HOLDUP
FROM SOUND
WHERE d = 'high'
INTERSECT
SELECT stereo,mono
FROM SOUND
WHERE d = 'low';
```

Next we check the COUNT of the HOLDUP table:

```
SELECT COUNT(*) FROM HOLDUP;

COUNT = 1 - {NOT A DECISION REDUCT}
```

Test for Screen

```
SELECT thirtysixinch,fiftyoneinch,sixtysixinch,
       seventysixinch INTO HOLDUP
FROM SCREEN
WHERE d = 'high'
INTERSECT
SELECT thirtysixinch,fiftyoneinch,sixtysixinch,
       seventysixinch
FROM SCREEN
WHERE d = 'low';
```

Next we check the COUNT of the HOLDUP table:

```
SELECT COUNT(*) FROM HOLDUP;
```

```
COUNT = 1 - {NOT A DECISION REDUCT}
```

These results indicate that in our example a single attribute is not a decision reduct.

STEP 3

Now we define views as the join of two attribute tables and then execute the same intersect query and process used in step 2 to find the decision reduct.

```
CREATE VIEW [PRICE AND GUARANTEE] AS
SELECT PRICE.high,PRICE.low,PRICE.medium,
       GUARANTEE.sixmonths,GUARANTEE.twelvemonths,
       GUARANTEE.eighteenmonths,GUARANTEE.twentymonths,GUARANTEE.d
FROM PRICE JOIN GUARANTEE
ON PRICE.oid=GUARANTEE.oid
```

The query will create a view given in table 4.6.

Oid	high	low	medium	6 mos	12 mos	18 mos	24 mos	d
1	Y	N	N	N	N	N	Y	high
2	N	Y	N	Y	N	N	N	low
3	N	Y	N	N	Y	N	N	low
4	N	N	Y	N	Y	N	N	high
5	N	N	Y	N	N	Y	N	high
6	Y	N	N	N	Y	N	N	low

Table 4.6: Price and Guarantee View

We run the following query on this view:

```
SELECT high,low,medium,sixmonths,twelvemonths,
        eighteenmonths,twentyfourmonth INTO HOLDUP
FROM [PRICE AND GUARANTEE]
WHERE d = 'high'
INTERSECT
SELECT high,low,medium,sixmonths,twelvemonths,
       eighteenmonths,twentyfourmonths
FROM [PRICE AND GUARANTEE]
WHERE d = 'low;
```

Next we check the COUNT of the HOLDUP table:

```
SELECT COUNT(*) FROM HOLDUP;

COUNT = 0 - {DECISION REDUCT}
```

We adopt the same routine for the other combinations of the two attributes.

```
CREATE VIEW [PRICE AND SOUND] AS
SELECT PRICE.high,PRICE.low,PRICE.medium,
       SOUND.stereo,SOUND.mono,SOUND.d
FROM PRICE JOIN SOUND
ON PRICE.oid=SOUND.oid
```

The PRICE and SOUND view will look like as:

Oid	high	low	medium	stereo	mono	d
1	Y	N	N	Y	N	high
2	N	Y	N	N	Y	low
3	N	Y	N	Y	N	low
4	N	N	Y	Y	N	high
5	N	N	Y	Y	N	high
6	Y	N	N	Y	N	low

Table 4.7: Price and Sound View

We run the following query on this view:

```
SELECT high,low,medium,stereo,mono INTO HOLDUP
FROM [PRICE AND SOUND]
WHERE d = 'high'
INTERSECT
SELECT high,low,medium,stereo,mono
FROM [PRICE AND SOUND]
WHERE d = 'low';
```

Next we check the COUNT of the HOLDUP table:

```
SELECT COUNT(*) FROM HOLDUP;
```

```
COUNT = 1 - {NOT A DECISION REDUCT}
```

Using the same idea for the combination of the other attributes.

```
CREATE VIEW [PRICE AND SCREEN] AS
SELECT PRICE.high,PRICE.low,PRICE.medium,
       SCREEN.thirtysixinch,SCREEN.fiftyoneinch,
       SCREEN.sixtysixinch,SCREEN.seventysixinch,SCREEN.d
FROM PRICE JOIN SCREEN
ON PRICE.oid=SCREEN.oid
```

The PRICE and SCREEN view will look like as:

Oid	high	low	medium	36″	41″	66″	76″	d
1	Y	N	N	N	N	N	Y	high
2	N	Y	N	N	N	Y	N	low
3	N	Y	N	Y	N	N	N	low
4	N	N	Y	N	Y	N	N	high
5	N	N	Y	N	Y	N	N	high
6	Y	N	N	N	Y	N	N	low

Table 4.8: Price and Screen View

We run the following query on this view:

```
SELECT high,low,medium,
       thirtysixinch,fiftyoneinch,sixtysixinch,seventysixinch
INTO HOLDUP
FROM [PRICE AND SCREEN ]
WHERE d = 'high'
INTERSECT
SELECT high,low,medium,
       thirtysixinch,fiftyoneinch,sixtysixinch,seventysixinch
FROM [PRICE AND SCREEN ]
WHERE d = 'low';
```

Next we check the COUNT of the HOLDUP table:

```
SELECT COUNT(*) FROM HOLDUP;
```

```
COUNT = 0 - {DECISION REDUCT}
```

Now we create the view of Guarantee and Sound attributes.

```
CREATE VIEW [GUARANTEE AND SOUND] AS
SELECT GUARANTEE.sixmonths,GUARANTEE.twelvemonths,
       GUARANTEE.eighteenmonths,GUARANTEE.twentyfourmonths,
       SOUND.stereo,SOUND.mono,SOUND.d
FROM GUARANTEE JOIN SOUND
ON GUARANTEE.oid=SOUND.oid
```

The GUARANTEE and SOUND view will look like as:

Oid	6 mos	12 mos	18 mos	24 mos	stereo	mono	d
1	N	N	N	Y	Y	N	high
2	Y	N	N	N	N	Y	low
3	N	Y	N	N	Y	N	low
4	N	Y	N	N	Y	N	high
5	N	N	Y	N	Y	N	high
6	N	Y	N	N	Y	N	low

Table 4.9: Guarantee and Sound View

We run the following query on this view:

```
SELECT sixmonths,twelvemonths,eighteenmonths,twentyfourmonths,
       stereo,mono
INTO HOLDUP
FROM [GUARANTEE AND SOUND ]
WHERE d = 'high'
INTERSECT
SELECT high,low,medium,thirtysixinch,fiftyoneinch,
       sixtysixinch,seventysixinch
FROM [GUARANTEE AND SOUND ]
WHERE d = 'low';
```

Next we check the COUNT of the HOLDUP table:

```
SELECT COUNT(*) FROM HOLDUP;
```

```
COUNT = 1 - {NOT A DECISION REDUCT}
```

Next we create the view of Guarantee and Screen attributes.

```
CREATE VIEW [GUARANTEE AND SCREEN] AS
SELECT GUARANTEE.sixmonths,GUARANTEE.twelvemonths,
       GUARANTEE.eighteenmonths,GUARANTEE.twentyfourmonths,
       SCREEN.thirtysixinch,SCREEN.fiftyoneinch,
```

```
      SCREEN.sixtysixinch,SCREEN.seventysixinch,SCREEN.d
FROM GUARANTEE JOIN SCREEN
ON GUARANTEE.oid=SCREEN.oid
```

The GUARANTEE and SCREEN view will look like as:

Oid	6 mos	12 mos	18 mos	24 mos	36"	41"	66"	76"	d
1	N	N	N	Y	N	N	N	Y	high
2	Y	N	N	N	N	N	Y	N	low
3	N	Y	N	N	Y	N	N	N	low
4	N	Y	N	N	N	Y	N	N	high
5	N	N	Y	N	N	Y	N	N	high
6	N	Y	N	N	N	Y	N	N	low

Table 4.10: Guarantee and Screen View

We run the following query on this view:

```
SELECT sixmonths,twelvemonths,eighteenmonths,twentyfourmonths,
thirtysixinch,fiftyoneinch,sixtysixinch,seventysixinch
INTO HOLDUP
FROM [GUARANTEE AND SCREEN ]
WHERE d = 'high'
INTERSECT
SELECT sixmonths,twelvemonths,eighteenmonths,twentyfourmonths,
thirtysixinch,fiftyoneinch,sixtysixinch,seventysixinch
FROM [GUARANTEE AND SCREEN ]
WHERE d = 'low';
```

Next we check the COUNT of the HOLDUP table:

```
SELECT COUNT(*) FROM HOLDUP;
```

```
COUNT = 1 - {NOT A DECISION REDUCT}
```

And finally we create the view of Sound and Screen attributes.

```
CREATE VIEW [SOUND AND SCREEN] AS
SELECT SOUND.stereo,SOUND.mono,
      SCREEN.thirtysixinch,SCREEN.fiftyoneinch,
      SCREEN.sixtysixinch,SCREEN.seventysixinch,SCREEN.d
FROM SOUND JOIN SCREEN
ON SOUND.oid=SCREEN.oid
```

The SOUND and SCREEN view will look like as:

Oid	stereo	mono	36″	41″	66″	76″	d
1	Y	N	N	N	N	Y	high
2	N	Y	N	N	Y	N	low
3	Y	N	Y	N	N	N	low
4	Y	N	N	Y	N	N	high
5	Y	N	N	Y	N	N	high
6	Y	N	N	Y	N	N	low

Table 4.11: Sound and Screen View

We run the following query on this view:

```
SELECT stereo,mono,thirtysixinch,fiftyoneinch,
       sixtysixinch,seventysixinch
INTO HOLDUP
FROM [SOUND AND SCREEN ]
WHERE d = 'high'
INTERSECT
SELECT stereo,mono,thirtysixinch,fiftyoneinch,
       sixtysixinch,seventysixinch
FROM [SOUND AND SCREEN ]
WHERE d = 'low';
```

Next we check the COUNT of the HOLDUP table:

```
SELECT COUNT(*) FROM HOLDUP;
```

```
COUNT = 1 - {NOT A DECISION REDUCT}
```

These results indicate that in our example *Price, Guarantee* and *Price, Screen* are the decision reducts.

STEP 4

Next we do the JOIN of three attributes but we know that *PRICE, GUARANTEE* and *PRICE, SCREEN* are already found as reduct so in our combination of the three attributes, we will ignore those attributes as it will result in a combination that already has a reduct. Therefore

```
{PRICE, GUARANTEE, SOUND} - IGNORED
{PRICE, GUARANTEE, SCREEN} - IGNORED
{PRICE, SOUND, SCREEN} - IGNORED
```

We will only test *GUARANTEE, SOUND, and SCREEN* as below.

```
CREATE VIEW [GUARANTEE AND SOUND AND SCREEN] AS
SELECT GUARANTEE.sixmonths,GUARANTEE.twelvemonths,
       GUARANTEE.eighteenmonths,GUARANTEE.twentyfourmonths,
       SOUND.stereo,SOUND.mono,SCREEN.thirtysixinch,
       SCREEN.fiftyoneinch,SCREEN.sixtysixinch,
       SCREEN.seventysixinch,SCREEN.d
FROM GUARANTEE
JOIN SOUND
ON GUARANTEE.oid=SOUND.oid
JOIN SCREEN
ON GUARANTEE.oid=SCREEN.oid
```

The GUARANTEE, SOUND and SCREEN view will look like 4.12:

Oid	6 mos	12 mos	18 mos	24 mos	stereo	mono	36"	41"	66"	76"	d
1	N	N	N	Y	Y	N	N	N	N	Y	high
2	Y	N	N	N	N	Y	N	N	Y	N	low
3	N	Y	N	N	Y	N	Y	N	N	N	low
4	N	Y	N	N	Y	N	N	Y	N	N	high
5	N	N	Y	N	Y	N	N	Y	N	N	high
6	N	Y	N	N	Y	N	N	Y	N	N	low

Table 4.12: Guarantee and Sound and Screen View

We run the following query on this view:

```
SELECT sixmonths,twelvemonths,eighteenmonths,twentyfourmonths,
       stereo,mono,thirtysixinch,fiftyoneinch,sixtysixinch,
       seventysixinch INTO HOLDUP
FROM [GUARANTEE SOUND AND SCREEN]
WHERE d = 'high'
INTERSECT
SELECT sixmonths,twelvemonths,eighteenmonths,twentyfourmonths,
       stereo,mono,thirtysixinch,fiftyoneinch,sixtysixinch,
       seventysixinch
FROM [GUARANTEE SOUND AND SCREEN]
WHERE d = 'low';
```

Next we check the COUNT of the HOLDUP table:

```
SELECT COUNT(*) FROM HOLDUP;

COUNT = 1 - {NOT A DECISION REDUCT}
```

4.6.2 Issues with the Initial Attempt

The initial attempt seems to work fine and is able to find the decision reducts correctly for a given data set but this approach will not be practical for the data sets that have the attributes with a wide range of possible values. Let us take an example the attribute age of the bank account holder. The range of this attribute could be 18 - 95 may be. If we follow the above approach, the AGE table could have 78 columns including the decision column. Handling tables with such a wide range of possible columns would not only be practically challenging but also will affect performance badly too. In fact our, this attempt will not be possible to implement for the data set in which the attributes have infinite range of values like integer or real numbers. That is why we have dropped the idea presented in the initial attempt and began thinking for a more feasible and practical solution.

4.6.3 Second Attempt

Keeping in mind the reason for the failure of the initial attempt regarding finding the decision reducts using database schema and queries, we are proposing a lot simpler and more practical solution to find the decision reducts inside the database.For this solution, once again we use the television buying example.

Case Type	Price	Guarantee	Sound	Screen	d
1	high	24 months	stereo	76	high
2	low	6 months	mono	66	low
3	low	12 months	stereo	36	low
4	medium	12 months	stereo	51	high
5	medium	18 months	stereo	51	high
6	high	12 months	stereo	51	low

Table 4.13: Television sets

STEP 1

We create a new schema, that includes two tables i.e. TV_HIGH and TV_LOW. TV_HIGH table contains all the records that have the decision class (d) as "high "and TV_LOW table contains all the records that have the decision class (d) as "low ". The new tables' structure is given below:

Field	Type	Null
Price	varchar(8)	Yes
Guarantee	int(11)	Yes
Sound	varchar(8)	Yes
Screen	int(11)	Yes
Decision_Class	varchar(8)	Yes

Table 4.14: TV_HIGH and TV_LOW Tables

STEP 2

After defining the tables now, we run the following queries. These queries get the count of the rows by doing the inner join of TV_HIGH and TV_LOW tables using a set of columns. The columns for the JOIN are the subset of all the columns in the original tables. First we check for the one attribute:

```
SELECT COUNT(*) FROM TV_HIGH
INNER JOIN TV_LOW USING (price)

COUNT = 1 - NOT A DECISION REDUCT

SELECT COUNT(*) FROM TV_HIGH
INNER JOIN TV_LOW USING (guarantee)

COUNT = 2 - NOT A DECISION REDUCT

SELECT COUNT(*) FROM TV_HIGH
INNER JOIN TV_LOW USING (sound)

COUNT = 6 - NOT A DECISION REDUCT

SELECT COUNT(*) FROM TV_HIGH
INNER JOIN TV_LOW USING (screen)

COUNT = 2 - NOT A DECISION REDUCT
```

Now we test for the combination of two attributes:

```
SELECT COUNT(*) FROM TV_HIGH
INNER JOIN TV_LOW USING (price,guarantee)

COUNT = 0 - DECISION REDUCT

SELECT COUNT(*) FROM TV_HIGH
INNER JOIN TV_LOW USING (price,sound)

COUNT = 1 - NOT A DECISION REDUCT

SELECT COUNT(*) FROM TV_HIGH
INNER JOIN TV_LOW USING (price,screen)

COUNT = 0 - DECISION REDUCT

SELECT COUNT(*) FROM TV_HIGH
INNER JOIN TV_LOW USING (guarantee,sound)
```

```
COUNT = 2 - NOT A DECISION REDUCT

SELECT COUNT(*) FROM TV_HIGH
INNER JOIN TV_LOW USING (guarantee,screen)

COUNT = 1 - NOT A DECISION REDUCT

SELECT COUNT(*) FROM TV_HIGH
INNER JOIN TV_LOW USING (sound,screen)

COUNT = 2 - NOT A DECISION REDUCT
```

Now that we know that the *Price, Guarantee* and *Price, Screen* are found as reducts we ignore those subsets of three attributes that contain the *Price, Guarantee* and the *Price, Screen*. This is to avoid finding the super set of reducts. The only set that remains then is checked.

```
SELECT COUNT(*) FROM TV_HIGH
INNER JOIN TV_LOW USING (guarantee,sound,screen)

COUNT = 1 - NOT A DECISION REDUCT
```

FINAL STEP

The above schema and the queries are going to find all decision reducts. There is no chance of finding the super reducts. All the reduct finding logic works in the database outside the memory (Java). In the final step we combined the all the above queries and created a store procedure which will produce the decision reducts by calling all the queries in a loop.

For this store procedure to work, we created another table -"TV_SUBSET" that contains all the subset except empty set for the tv table column names. The table also contains another column -"reduct_flag" with a default value "N". This column is updated in the stored procedure if a particular subset is found to be a reduct. The store procedure also set the value "Y" for all the other subsets that have the reducts as a subset. The data in the new table "TV_SUBSET" is given below:

Id	Subset	Reduct_Flag
1	price	N
2	guarantee	N
3	sound	N
4	screen	N
5	price,guarantee	N
6	price,sound	N
7	price,screen	N
8	guarantee,sound	N
9	guarantee,screen	N
10	sound,screen	N
11	price,guarantee,sound	N
12	price,guarantee,screen	N
13	price,sound,screen	N
14	guarantee,sound,screen	N
15	price,guarantee,sound,screen	N

Table 4.15: TV_SUBSET Table

Also we define another table *TV_REDUCT* which will contain all the reducts after the store procedure is finished. This table structure is following:

Field	Type
id	int(11)
reduct	varchar(500)

Table 4.16: TV_REDUCT Table

Now we define a store procedure that makes use of the *TV_SUBSET* and the *TV_REDUCT* table. In case the subset is a reduct, the store procedure inserts a record in the *TV_REDUCT* table and updates the reduct_flag in the *TV_SUBSET* table for that and all the other records that have the last found reduct as a subset to avoid testing them and. Below is the syntax of the store procedure which can be run in any database.

```
DROP PROCEDURE IF EXISTS GetAllTvReducts;
DELIMITER //
PROCEDURE GetAllTvReducts (IN c INT)
READS SQL DATA
```

```
BEGIN
DECLARE done INT DEFAULT 1;
DECLARE arg VARCHAR(1000);
DECLARE sum VARCHAR(1000);
hist_loop: LOOP
    SELECT subset INTO arg from test.TV_SUBSET
      WHERE id = done AND reduct_flag = 'N';
      set @sql =concat('select COUNT(*) INTO @total
                       FROM test.TV_HIGH
                       INNER JOIN test.TV_LOW
                       USING (', arg, ')');
      PREPARE stmt1 FROM @sql;
      EXECUTE stmt1;
      IF @total = 0 THEN
       SET sum = REPLACE(arg,',','\%');
         INSERT into test.TV_REDUCT(reduct) values (arg);
         UPDATE test.TV_SUBSET
         SET reduct_flag = 'S'
         WHERE subset LIKE concat('\%',sum,'\%');
         COMMIT;
      END IF;
      DEALLOCATE PREPARE stmt1;
      SET @total = 0;
      SET done = done + 1;
      IF done > c
       THEN LEAVE hist_loop;
      END IF;
      END LOOP hist_loop;
END
DELIMITER ;
```

4.6.4 Issues with the Second Attempt

The solution presented above as a second attempt is workable and finds the reducts correctly. However the JOIN operation is very expensive and requires the full table scan. Due to the full table scan the *"@sql"* query in the above store procedure takes a long time and results may not be as good as expectations.

Chapter 5

An Improved Reduct Determination Algorithm

5.1 Introduction

In this chapter we present an improved reduct determination algorithm based on the second attempt toward the solution that we presented in the last chapter. This algorithm is based on the same schema what we presented in the last chapter i.e. separate tables for each decision class, a table of all the subsets for the conditional attributes and a table to insert the reducts found. There are however few changes that we made to the second solution from the section 4.6.3. These changes are following:

- The results of the *"@sql"* query in the store procedure presented in the Section 4.6.3 are limited to 1 only. Therefore an addition of LIMIT 0,1 is added to the query. The reason for this addition is that we only need to know if there is one record common in two tables based on the combination of the variables fed from the subset table.

- The second change that we made to our query is that instead of doing a JOIN, we opted to test each attribute in the WHERE clause i.e. change our query to:

```
@sql = concat('select 1 INTO @total
```

```
FROM test.TV_HIGH X, test.TV_LOW Y
WHERE X.PRICE = Y.PRICE AND X.SOUND = Y.SOUND LIMIT 0,1)
```

which will test for PRICE,SOUND combination. Other combinations are also tested in the same manner.

- The change in the last point prompts a change in the entries of our SUBSET table. Now we make entries for all the subset combinations but in the form of *X.ATTRIBUTE1 = Y.ATTRIBUTE1 AND X.ATTRIBUTE2 = Y.ATTRIBUTE2* and so on. We will name this new column as *SQL_SUBSET*.

Due to the changes that we mentioned above, we made a slight change in the subset table. We added another column *SQL_SUBSET* in the table that contains the entries of the subset but in the format mentioned in the point 3 above. The entries for this additional column are presented in table 5.1 .

Id	SQL_SUBSET	Reduct_Flag
1	X.PRICE = Y.PRICE	N
2	X.GUARANTEE = Y.GUARANTEE	N
3	X.SOUND = Y.SOUND	N
4	X.SCREEN = Y.SCREEN	N
5	X.PRICE = Y.PRICE and X.GUARANTEE = Y.GUARANTEE	N
6	X.PRICE = Y.PRICE and X.SOUND = Y.SOUND	N
7	X.PRICE = Y.PRICE and X.SCREEN = Y.SCREEN	N
8	X.GUARANTEE = Y.GUARANTEE and X.SOUND = Y.SOUND	N
9	X.GUARANTEE = Y.GUARANTEE and X.SCREEN = Y.SCREEN	N
10	X.SOUND = Y.SOUND AND X.SCREEN = Y.SCREEN	N
11	X.PRICE = Y.PRICE AND X.GUARANTEE = Y.GUARANTEE AND X.SOUND = Y.SOUND	N
12	X.PRICE = Y.PRICE AND X.GUARANTEE = Y.GUARANTEE AND X.SCREEN = Y.SCREEN	N
13	X.PRICE = Y.PRICE AND X.SOUND = Y.SOUND AND X.SCREEN = Y.SCREEN	N
14	X.GUARANTEE = Y.GUARANTEE AND X.SOUND = Y.SOUND AND X.SCREEN = Y.SCREEN	N
15	X.PRICE = Y.PRICE AND X.GUARANTEE = Y.GUARANTEE AND X.SOUND = Y.SOUND AND X.SCREEN = Y.SCREEN	N

Table 5.1: TV_SUBSET Table

5.2 Improved Approach

We revisited the solution provided in the second attempt. For a subset of attributes to be a reduct, it is required that there should not be one record that is common in the both *TV_HIGH* and *TV_LOW* tables. Keeping that in mind, we put the limit to one on the number of records retrieved. The improved version of the store procedure is following:

```
DROP PROCEDURE IF EXISTS GetAllTvReducts;
```

```
DELIMITER //
PROCEDURE GetAllTvReducts (IN c INT)
READS SQL DATA
BEGIN
    DECLARE done INT DEFAULT 1;
    DECLARE sql_arg VARCHAR(1000);
    DECLARE sum VARCHAR(1000);
    hist_loop: LOOP
      SET @total = 0;
      SELECT sql_subset, red_subset INTO sql_arg, red_arg
      from test.TV_SUBSET
      WHERE id = done AND reduct_flag = 'N';
      SET done = done + 1;
      IF done > c
       THEN LEAVE hist_loop;
      END IF;
      IF sql_arg <> '' THEN
       set @sql = concat(select 1 INTO @total
                  FROM test.TV_HIGH X, test.TV_LOW Y
                  WHERE ',sql_arg,' LIMIT 0,1);
       PREPARE stmt1 FROM @sql;
       EXECUTE stmt1;
       DEALLOCATE PREPARE stmt1;
       IF @total = 0 THEN
         INSERT into test.TV_REDUCT(reduct) values (red_arg);
         SET sum = REPLACE(red_arg,',','\%');
         UPDATE test.TV_SUBSET
         SET reduct_flag = 'S'
         WHERE subset LIKE concat('\%',sum,'\%');
         COMMIT;
       END IF;
      END IF;
      SET sql_arg = '';
    END LOOP hist_loop;
END
DELIMITER ;
```

The above store procedure finds the reducts in the dataset very efficiently and quickly. Because all of the reduct finding process (which is the bottleneck in the rough set algorithm) is done in the database, it improves the scalability of the overall rough set algorithm a great deal. The store procedure will work for the data set with the decision variable with two classes. If however the decision attribute contains more than 2 classes, a slight change in the schema and the store procedure will make it workable. For the schema we will have to add a new table that contains all the binary combinations of the decision classes presented in Table 5.2:

Field	Type
id	int(11)
left_table	varchar(500)
right_table	varchar(500)

Table 5.2: Decision Tables

Now the store procedure will be changed as following:

```
DROP PROCEDURE IF EXISTS GetAllTvReducts;
DELIMITER //
PROCEDURE GetAllTvReducts (IN c INT)
READS SQL DATA
BEGIN
    DECLARE done INT DEFAULT 1;
    DECLARE tab_cnt INT DEFAULT 1;
    DECLARE sql_arg VARCHAR(1000);
    DECLARE sum VARCHAR(1000);
    hist_loop: LOOP
      SELECT sql_subset, red_subset INTO sql_arg, red_arg
      from test.TV_SUBSET
      WHERE id = done AND reduct_flag = 'N';
      SET done = done + 1;
      IF done > c
       THEN LEAVE hist_loop;
      END IF;
      table_loop: LOOP
```

```
        set @a = tab_cnt;
        set @st1 := concat( 'SELECT id,left_table, right_table
                            INTO @left_tab,@right_tab
                            FROM test.DECISION_TABLES
                            WHERE id = ? ');
        PREPARE stm1 FROM @st1;
        EXECUTE stm1 using @a;
        DEALLOCATE PREPARE stm1;
        SET tab_cnt = tab_cnt + 1;
        IF tab_cnt > t THEN LEAVE table_loop; END IF;
        IF @left_tab <> '' THEN
           set @sql = concat('select 1 INTO @total FROM
                     test.', @left_tab,' X, test.',  @right_tab,' Y
                     WHERE  ',sql_arg,' LIMIT 1');
           PREPARE stmt1 FROM @sql;
           EXECUTE stmt1;
           DEALLOCATE PREPARE stmt1;
           IF @total > 0 THEN LEAVE table_loop; END IF;
         END IF;
         SET @left_tab = '';
         SET @right_tab = '';
       END LOOP table_loop;
       IF @total IS NULL OR @total = 0 THEN
         INSERT into test.TV_REDUCT(reduct) values (red_arg);
         SET sum = REPLACE(red_arg,',','\%');
         UPDATE test.TV_SUBSET
         SET reduct_flag = 'S'
         WHERE red_subset LIKE concat('\%',sum,'\%');
         COMMIT;
        END IF;
      END IF;
      SET red_arg = '';
SET @total = 0;
    END LOOP hist_loop;
END
DELIMITER ;
```

Chapter 6

Testing and Results

6.1 Introduction

In this chapter we will test the solution that we proposed in the final section of the previous chapter. We will apply the store procedure on two different data sets to find the decision reducts in order to confirm the validity of our approach. The results from these tests will be compared against the results from the traditional discernibility matrix approach using the same data set. For the traditional discernibility approach, we will use the RSDA implementation in the Rseslib [2]. We have implemented these tests on an *Intel(R) Core(TM)2 Duo CPUP8800 @ 2.66GHz* machine with 4GB of RAM and running a *Microsoft Windows XP Professional* operating system. The comparison of the results will be presented to understand that how the scalability of the reduct finding is improved by using our proposed approach. The results are presented to view that our proposed approach of reduct finding performs better in terms of:

- Horizontal Scalability: In this case the number of records or rows are kept fixed and the number of attributes or columns are increased gradually from lower to maximum number of attributes.

- Vertical Scalability: In this case the number of attributes or columns are fixed while the number of records or rows are increased gradually from the lower to the maximum number of rows in the data set.

For testing of our algorithm and the traditional discernibility matrix approach, we downloaded two data sets from the UCI Machine Learning Repository [1]. These data sets are Adult Data Set and the Person Activity Data Set. A brief description of these data sets is given in the tests sections below.

6.2 First Test

In this test we will apply both our proposed stored procedure approach and the traditional discernibility approach on the Adult Data Set. For better readability and understanding, the results for both the vertical and horizontal scalability tests are presented in the form of the tables.

6.2.1 Adult Data Set

The Adult Data Set [1] is a multivariate classification type of data set. The data was extracted from the US Census data 1994. The data set based on the 14 attributes identifies whether a person makes over 50K or less than 50K a year. The attributes for this data set and their domain values are as following:

- **age:** continuous.
- **workclass:** Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked.
- **fnlwgt:** continuous.
- **education:** Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool.
- **education-num:** continuous.
- **marital-status:** Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse.
- **occupation:** Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.
- **relationship:** Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.

- **race:** White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.
- **sex:** Female, Male.
- **capital-gain:** continuous.
- **capital-loss:** continuous.
- **hours-per-week:** continuous.
- **native-country:** United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinadad&Tobago, Peru, Hong, Holand-Netherlands.
- **decision-class:** >50K, <= 50K.

There are 48842 records in this table with 11687 records have the decision class >50K and 37155 have the decision class <= 50K. This means that in our proposed solution we create two table i.e. ADULT_GRTR_FIFTY table that contains 11687 records and the ADULT_LESSEQ_FIFTYK with 37155 records in it. Both tables have exact same columns and column names.

Field	Type
id	int(11)
age	int(11)
workclass	varchar(45)
fnlwgt	int(11)
education	varchar(45)
edu_num	int(11)
marital_status	varchar(45)
occupation	varchar(45)
relationship	varchar(45)
race	varchar(45)
sex	varchar(10)
capital_gain	int(11)
capital_loss	int(11)
hours_per_week	int(11)
native_country	varchar(45)
decision_class	varchar(10)

Table 6.1: ADULT_GRTR_FIFTYK and ADULT_LESSEQ_FIFTYK

Using the ADULT_GRTR_FIFTYK and ADULT_LESSEQ_FIFTYK and applying the first store procedure mentioned in the section 5.2, we got the following results which is compared with the traditional discernibility matrix approach applied on the same data set.

6.2.2 Results for Horizontal Scalability

In this case the number of rows or records is static at 32561 and the results are based on increasing the number of columns from 4 to maximum number of conditional attributes i.e. 14. The original table consists of following conditional attributes.

- *AGE, WORKCLASS, FNLWGT, EDUCATION, EDU_NUM, MARITAL_STATUS, OCCUPATION, RELATIONSHIP, RACE, SEX, CAPITAL_GAIN, CAPITAL_LOSS, HOURS_PER_WEEK, NATIVE_COUNTRY*

There are two decision reducts that are found by both the approaches. These reducts are:

1. *AGE, WORKCLASS, FNLWGT, EDUCATION, OCCUPATION, RELATIONSHIP, CAPITAL_GAIN, HOURS_PER_WEEK*

2. *AGE, WORKCLASS, FNLWGT, EDU_NUM, OCCUPATION, RELATIONSHIP, CAPITAL_GAIN, HOURS_PER_WEEK*

Attributes	Time Taken (Traditional Approach)	Proposed Solution		Reducts
		Subsets	Time Taken	
4	5 min	15	0.03 sec	N
5	6 min	31	0.06 sec	N
6	6 min	63	0.11 sec	N
7	6 min	127	0.47 sec	N
8	7 min	255	0.91 sec	N
9	8 min	511	1.73 sec	N
10	9 min	1023	3.36 sec	N
11	9 min	2047	22.77 sec	N
12	10 min	4095	46.42 sec	N
13	10 min	8191	1 min	Y
14	10 min	16383	6 min	Y

Table 6.2: Horizontal Scalability Results

The results clearly shows that our proposed solution outperforms the traditional discernibility approach for different number of attributes while using the complete set of records in the data set. Also by increasing the number of attributes discretely starting from 4, the time taken to find the reducts in the case of our proposed approach does not increase proportionally while the traditional approach scales poorly in that case. Therefore we can say that our algorithm is more horizontally scalable for this data set than the traditional discernibility matrix approach.

6.2.3 Results for Vertical Scalability

In this case the number of conditional attributes or columns is static at 14 and the results are based on increasing the number of rows or records from 10,000 to maximum number of records in the data set. The original table consists of following conditional attributes.

- *AGE, WORKCLASS, FNLWGT, EDUCATION, EDU_NUM, MARITAL_STATUS, OCCUPATION, RELATIONSHIP, RACE, SEX, CAPITAL_GAIN, CAPITAL_LOSS, HOURS_PER_WEEK, NATIVE_COUNTRY*

No. of Attributes	No. of Records	Time Taken (Traditional Approach)	Time Taken (Proposed Approach)
14	10,000	1 min	5 min
14	15,000	2 min	5 min
14	20,000	3 min	5 min
14	25,000	6 min	5 min
14	32,561	10 min	6 min
14	48,842	24 min	7 min

Table 6.3: Vertical Scalability Results

There are two decision reducts that are found by both the approaches for the full data set i.e. all the attributes and the complete set of records are:

1. *AGE, WORKCLASS, FNLWGT, EDUCATION, OCCUPATION, RELATIONSHIP, CAPITAL_GAIN, HOURS_PER_WEEK*

2. *AGE, WORKCLASS, FNLWGT, EDU_NUM, OCCUPATION, RELATIONSHIP, CAPITAL_GAIN, HOURS_PER_WEEK*

The results for the vertical scalability are different and less convincing from the ones for the horizontal scalability. However they also show that by increasing the number of records gradually starting from 10,000, the time taken to find the reducts in the case of our proposed algorithm does not increase proportionally while the traditional approach scales poorly in that case. For the less number of rows, the traditional approach does perform better than our proposed algorithm but for the larger number of records, our solution starts performing much better than the traditional approach. Therefore we can say that our algorithm is more vertically scalable too for this data set than the traditional discernibility matrix approach.

6.3 Second Test

In this test we will apply both our proposed stored procedure approach and the traditional discernibility approach on the Person Activity Data Set. The details about this data set are given in the next section. For this data set we only collected the results for the vertical scalability tests. The results of our approach in comparison with the traditional discernibility matrix approach are presented in the form of the tables for readability and understanding.

6.3.1 Person Activity Data Set

The Person Activity Data Set [1] is a multivariate classification type of data set. The data contains recordings of five people performing different activities. Each person wore four sensor tags (ankle left, ankle right, belt and chest) while performing the same scenario five times. The data set consists of 7 attributes identifies the person activity defined as one of the 11 decision classes. The attributes for this data set and their domain values are as following:

- **Sequence Name:** {A01,A02,A03,A04,A05,B01,B02,B03,B04,B05,C01, C02,C03,C04,C05, D01,D02,D03,D04,D05,E01,E02,E03,E04,E05} Nominal - A, B, C, D, E = 5 people

- **Tag identificator:** {010-000-024-033,020-000-033-111,020-000-032-221, 010-000-030-096}
  Nominal
  - ANKLE_LEFT = 010-000-024-033
  - ANKLE_RIGHT = 010-000-030-096
  - CHEST = 020-000-033-111
  - BELT = 020-000-032-221

- **timestamp :** Numeric.

- **date:** date = dd.MM.yyyy HH:mm:ss:SSS Date

- **x coordinate of the tag:** Numeric.

- **y coordinate of the tag:** Numeric.

- **z coordinate of the tag:** Numeric.

- **decision_class:** {walking,falling,lying down,lying, sitting down,sitting,standing up from lying,on all fours, sitting on the ground,standing up from sitting, standing up from sitting on the ground} Nominal

There are 164860 records in this data set with 7 conditional attributes and one decision attribute. The decision attribute comprised of 11 different decision classes. This data set was selected to show how our proposed algorithm perform for a dataset with large number of records and high number of decision classes. Using our proposed approach presented before, we created 11 tables, one for each decision class. We will show the structure of one table which is same for the other tables. Using the 11 different tables, one for each decision class and applying

Field	Type
sequence_name	varchar(45)
tag_identifier	varchar(45)
timestamp	varchar(45)
date	varchar(45)
x_coord	varchar(45)
x_coord	varchar(45)
x_coord	varchar(45)
decision_class	varchar(45)

Table 6.4: Person Activity Table(For each activity)

the second store procedure mentioned in the section 5.2, we got the following results, compared against the traditional discernibility matrix approach applied on the same data set.

6.3.2 Results for Vertical Scalability

As mentioned before, we only did the tests for vertical scalability for this data set. This is because of the less number of conditional variables. For such a small number of conditional attributes, testing for horizontal scalability would not make much sense. In this case the number of conditional attributes or columns is static at 7 and the results are based on increasing the number of rows or records from 20,000 to maximum number of records in the data set. The original table consists of following conditional attributes.

- *SEQUENCE_NAME, TAG_IDENTIFIER, TIMESTAMP, DATE, X_COORD, Y_COORD, AND Z_COORD*

No. of Attributes	No. of Records	Time Taken (Traditional Approach)	Time Taken (Proposed Approach)
7	20,000	3 min	3.84 sec
7	40,000	15 min	7.92 sec
7	60,000	33 min	9.31 sec
7	80,000	87 min	10.05 sec
7	100,000	96 min	12.34 sec
7	120,000	142 min	14.31 sec
7	140,000	195 min	16.75 sec
7	164,860	276 min	19.34 sec

Table 6.5: Vertical Scalability Results

There are FIVE decision reducts that are found by both the approaches for the full data set i.e. all the attributes and the complete set of records are:

1. *TIMESTAMP*
2. *DATE*
3. *X_COORD, Y_COORD*
4. *X_COORD, Z_COORD*
5. *Y_COORD, Z_COORD*

The results for the vertical scalability clearly shows that our proposed reduct finding approach performs much better than the traditional discernibility matrix approach.

Chapter 7

Discussion of the Results

In this chapter we are going to discuss the results that we obtained from the testing of the two data sets i.e. Census Data Set and the Person Activity Data Set performed and presented in chapter 6. The results of the tests are very encouraging for our proposed approach for finding the attribute reducts but they also highlight the fact that more research can be performed to improve our proposed approach. The results show that our approach performs far better than the traditional reduct finding approach using the discernibility matrix for the two data sets but we can also derive from the results that for certain other data sets, our approach may not perform very well. In the next chapter we will identify the types of those data sets and the scenarios where our proposed approach will not perform as well and there is a possibility of future research. Here we will discuss the results obtained in the last chapter and do a comparison of our approach with the traditional rough set reduct finding approach.

7.1 Census Data Set

This data set was comprised of 14 conditional and 1 decision attribute with 2 decision classes. For this data set we performed two different tests.

- The first test in section 6.2.2 was performed to test the horizontal scalability i.e. keeping the number of records constant while increasing the number of conditional attributes gradually in the tests. The results show that for fewer number of attributes, our proposed algorithm finds the reducts very fast but

as increase the number of the conditional attributes, our algorithm starts getting slower and takes longer times to find the reducts. This is because of the fact that our algorithm depends heavily on the number of subset combinations to be tested as the reducts. So as we increase the number of conditional attributes, the number of subsets increase exponentially i.e. 2^n resulting in slowing down of the proposed algorithm. We can see from the results in the Table 6.2 that for fewer attributes, our proposed approach performs much faster than the traditional rough set algorithm but as we increase the number of attributes, our proposed approach starts performing just as good as the traditional discernibility matrix approach.

- In contrast to the horizontal scalability, the vertical scalability test in section 6.2.3 proved the strength of our proposed approach. In these tests we keep the number of conditional attributes constant at maximum 14 while increase the number of records to be tested gradually starting from 10,000 records and increasing them in 5,000 chunks. The results of this approach in the Table 6.3 show that for fewer number of records i.e. 10,000 to 20,000 the traditional discernibility approach performs better than our proposed approach, however as the number of the records increase form 25,000 and up, the traditional algorithm slows down while our algorithm keeps performing more or less same. This is the strength of our proposed algorithm that by increasing the number of records, our algorithm scales very well while the traditional discernibility approach slows down proportional to the increase in the number of records.

7.2 Person Activity Data Set

This data set comprised of 7 conditional attributes and 1 decision attribute. The data set contained 164,860 records. This data set was selected for two reasons. The first reason was the number of records. Using the data set with high number of records helps us testing how our proposed approach performs for higher number of records. The second reason was the number of decision classes for the decision attribute. This is very important as this would prove that for the data sets with more decision classes, our algorithm performs just as good as the data set with only two decision classes. As presented in Chapter 6, our proposed approach divides the data set into its respective decision class table. So in this case the test were done with data coming from 11 different tables.

For this data set we only performed the tests for testing the vertical scalability i.e. keeping the number of conditional attributes constant at maximum 7 while increasing the number of records gradually from 20,000 to the maximum. The result presented in Table 6.5 shows that our proposed approach performs far better than the traditional discernibility approach throughout. The results of our approach stayed in seconds for few of records i.e. 20,000 as well as for the maximum number rows. The traditional approach started in minutes and as we increased the number of rows to be tested the traditional approach started taking hours with the results for the maximum number of rows took it close to 5 hours to finish.

The results for both the data sets are very encouraging and shows that our proposed algorithm of finding the attribute reducts in the database with right schema can improve the performance of the rough set algorithm and make it more scalable to be accepted commercially in data mining algorithms.

Chapter 8

Conclusion and Future Research

8.1 Conclusion

In this thesis we highlighted the issue of the scalability in the traditional rough set algorithm. We identified that the root cause of the poor scalability of the traditional rough set algorithm is the attribute reduction step. In the traditional rough set algorithm, the attribute reduction is achieved by creating a discernibility matrix as described in section 3.2.3 in the memory. This matrix is then used to find the attribute reducts. We proposed a completely different approach towards the reduct finding by moving the entire process into the database. In our proposed approach, we defined a schema to store the data in the Information System and then defining queries on them (combined as a store procedure) outlined in the section 5.2. The advantages of our proposed solution are two-fold i.e. not only that it is faster than the traditional discernibility approach as demonstrated in the sections 6.2.2, 6.2.3, 6.3.2 but it also increases the overall efficiency of the RSDA and rule creation process by moving the resource costly process of reduct finding to a separate database server.

Like any new concept and algorithm, our proposed algorithm also suffers from some issues. Analyzing the results for the adult data set in section 6.2.1, couple of big issues with our proposed approach are highlighted. For the adult data set, the results for horizontal scalability in section 6.2.2 shows that for the less

number of attributes, our algorithm outperforms the traditional discernibility matrix approach but as we increase the number of conditional attributes, our algorithm starts slowing down and performs just as good as the traditional discernibility matrix approach. The reason for this poor performance is that our algorithm depends on the number of subsets of the conditional attributes to be tested. As we increase the number of conditional attributes, the number of subsets increase exponentially i.e. 2^n resulting in exponential slowing down of our algorithm.

Another issue that is also evident from the results of the adult data set given in the sections 6.2.2, 6.2.3 is that if the number of number of attributes in a reduct is close to the total number of attributes, then our proposed approach starts behaving slowly. The reason for this behavior is again linked to our approach being dependent on the subsets of the attributes. Say if the reduct comprised of n attributes, then to find that reduct form the table of all the subsets of the attributes, our proposed algorithm will have to test 2^{n-1} subsets combinations. Therefore if we increase the "n", our proposed approach will take longer to find the reduct because the reduct is deep down in the subset combination table while our algorithm checks each combination sequentially. This behavior is evident from the results for the adult data set, where the first reduct found consists of 8 attributes out of 14 conditional attributes. That is why in the case of adult data set, our algorithm performed slowly. In contrast to the adult data set, we can observe from the person activity data in section 6.3.1 set that our algorithm performed much faster because the first reduct consisted of 2 attributes out of the total 7 conditional attributes. The effect of finding the first reduct then decreases the number of tests on the subsets as all the other subset combinations that contain that reduct are discarded because they are super set of the reduct. This discarding or pruning increases the overall performance of our proposed algorithm.

In conclusion we can say that our proposed approach can perform better than the traditional discernibility matrix approach for some of the typical data sets. However, it has an issue to deal with the number of attributes in the data set and also to handle the issue with the number of attributes in the reduct found. The worst case for our proposed algorithm will be if there is no decision reduct in a data set. These issues can be addressed in future research on our proposed algorithm. Some of the suggestions for the future research are given in the next section.

8.2 Future Research

As identified in the previous section, our algorithm suffers from two major issues. First, it slows down if the number of conditional attributes in the data set are increased. Second, it sequentially checks the subset combination to find the reduct which results in slowness if the reduct is deep down in the subset combination table. For the issue of the number of conditional attributes, we suggest that we can perform the column division of the original data set table into two or more tables. For example, the adult data set presented in section 6.2.1, we can divide the data set into two tables each containing the data for 7 attributes and a decision attribute. These two tables can then be used to find the decision reducts. In the end a smart merging algorithm will be required to combine the results from the two tables and find the final decision reducts for the complete data set.

For the second issue with our algorithm where the reducts could be buried deep in the subset table, we suggest to use the genetic algorithm [18] or the heuristic search algorithms like hill climbing or simulated annealing which aim to find the reduct based on the fitness function for the subsets. The fitness function will optimize for the subset combination say the one with less number of attributes for the starters. We believe that employing heuristic search optimization algorithms can improve this issue in our proposed approach.

We also believe that the performance of our algorithm can be increased if we remove the noise from the data set which is common for almost every data set. A unique way of removing the noise is suggested by Xiaohua Hu [14]. He proposed the data generalization technique which is performed inside the database. The technique removes noise from the data by performing attribute removal and attribute-oriented concept tree ascension. We believe if we combine the data generalization with our proposed algorithm to find the reducts in the database, then we can also generate the decision rules as proposed again by Xiaohua Hu [14]. This way the whole RSDA can be performed inside the database.

Bibliography

[1] *UCI Machine Learning Repository*, 2010 (Last accessed April, 2011). `http://archive.ics.uci.edu/ml/datasets.html`.

[2] *Rseslib*, 2011 (Last accessed April, 2011). `http://rsproject.mimuw.edu.pl/`.

[3] *Infobright*, 2011 (Last accessed July, 2011). `http://www.infobright.com/`.

[4] You-Shyang Chen and Ching-Hsue Cheng. Forecasting pgr of the financial industry using a rough sets classifier based on attribute-granularity. *Knowl. Inf. Syst.*, 25:57–79, October 2010.

[5] Chris Cornelis, Germán Hurtado Martín, Richard Jensen, and Dominik Slezak. Feature selection with fuzzy decision reducts. In *RSKT*, pages 284–291, 2008.

[6] Kenneth A. De Jong, William M. Spears, and Diana F. Gordon. Using genetic algorithms for concept learning. *Mach. Learn.*, 13:161–188, November 1993.

[7] Juzhen Dong, Ning Zhong, and Setsuo Ohsuga. Using rough sets with heuristics for feature selection. In *Proceedings of the 7th International Workshop on New Directions in Rough Sets, Data Mining, and Granular-Soft Computing*, RSFDGrC '99, pages 178–187, London, UK, 1999. Springer-Verlag.

[8] Ivo Düntsch and Günther Gediga. Non–invasive data analysis. In *Proc. 8th Ireland Conference on Artificial Intelligence, Derry, (1997)*, pages 24–31, 1997.

[9] Ivo Düntsch and Günther Gediga. Feature selection and data prediction by rough entropy. In Hans-Jürgen Zimmermann, editor, *6th European Congress on Intelligent Techniques and Soft Computing EUFIT'98*, pages 81–85, Aachen, Germany, 1998.

[10] Ivo Düntsch and Günther Gediga. *Rough set data analysis: A road to non-invasive knowledge discovery*. Methoδos Publishers (UK), Bangor, 2000.

[11] Ivo Düntsch and Günther Gediga. Maximum consistency of incomplete data via non–invasive imputation. *Artificial Intelligence Review*, 19:93–107, 2003.

[12] Aboul ella Hassanien and Dominik Ślęzak. Rough neural intelligent approach for image classification: A case of patients with suspected breast cancer. *Int. J. Hybrid Intell. Syst.*, 3:205–218, December 2006.

[13] David E. Goldberg and Kalyanmoy Deb. A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of Genetic Algorithms*, pages 69–93. Morgan Kaufmann, 1991.

[14] Xiaohua Hu. *Knowledge Discovery in Databases: An Attribute Oriented Rough Set Approach*. PhD thesis, University of Regina, 1995.

[15] Ron Kohavi and Brian Frasca. Useful feature subsets and rough set reducts, 1994.

[16] Chiun-Sin Lin, Gwo-Hshiung Tzeng, and Yang-Chieh Chin. Combined rough set theory and flow network graph to predict customer churn in credit card accounts. *Expert Systems With Applications*, 38:8–15, 2011.

[17] Manish Mehta, Rakesh Agrawal, and Jorma Rissanen. Sliq: A fast scalable classifier for data mining. pages 18–32, 1996.

[18] Thomas Mitchell. *Machine Learning*. McGraw-Hill Education (ISE Editions), 1997.

[19] Z. Pawlak. Rough sets. *International Journal of Computer and Information Science*, 11:341–356, 1982.

[20] L. Polkowski and A. Skowron. *Rough Sets and Current Trends in Computing*. Springer Verlag, Berlin, 1998.

[21] J. Ross Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[22] John Shafer, Rakeeh Agrawal, and Manish Mehta. Sprint: A scalable parallel classifier for data mining. pages 544–555. Morgan Kaufmann, 1996.

[23] Shusaku Tsumoto. Mining diagnostic rules from clinical databases using rough sets and medical diagnostic model. *Inf. Sci.*, 162:65–80, May 2004.

[24] Alicja Wakulicz-Deja and Piotr Paszek. Applying rough set theory to multi stage medical diagnosing. *Fundam. Inf.*, 54:387–408, June 2002.