RULEGEN – A RULE GENERATION APPLICATION

USING MULTISET DECISION TABLES

A Thesis
Presented to
The Graduate Faculty of The University of Akron

In Partial Fulfillment
of the Requirements for the Degree
Master of Computer Science

Madhavi Gundavarapu
August, 2005
RULEGEN – AN APPLICATION FOR
LEARNING RULES FROM DATABASES

Madhavi Gundavarapu

Thesis

Approved:  

________________________________________  
Advisor  
Dr. Chien-Chung Chan

________________________________________  
Committee Member  
Dr. Zhong-Hui Duan

________________________________________  
Committee Member  
Dr. Xuan-Hien T. Dang

________________________________________  
Department Chair  
Dr. Wolfgang Pelz

Accepted:  

________________________________________  
Dean of the College  
Dr. Roger B. Creel

________________________________________  
Dean of the Graduate School  
Dr. George R. Newkome

Date
ABSTRACT

Volumes of data are generated constantly by several computer applications running across the globe. This trend of rapid increase in data generation is triggered by the ever-increasing usage of internet and database applications. In any particular application domain, having a large database that represents an equally large collection of facts about the domain, is of little use to experts trying to analyze facts, recognize trends or patterns and make decisions. Extracting useful information and ironing out inconsistencies from raw data is a challenging task and draws the research interests of computer scientists globally.

This study presents a system called RuleGen, for generating rules from data tables stored in databases. It is implemented using SQL (Structured Query Language), a standard language for developing database applications. The underlying algorithm is based on learning program LERS-M (Learning from Examples based on Rough MultiSets). RuleGen provides capability to transform numerical data in data tables into a format that can be easily understood and interpreted by users. In addition, users can define any number of views for a particular database table and choose to materialize a view at the time of definition or at a later point in time. In each view, users can define a different set of condition and/or decision attributes. This provides the users the capability to analyze a single table from different perspectives. This is similar to the slice and dice capability provided by OLAP tools.
ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to Dr. Chan for his continuous support and guidance throughout this project. Dr. Chan was always ready to answer questions and explain concepts when I felt lost.

I received invaluable support from my husband, Dr. Mahesh Kavaturu, as I slogged along this project. He encouraged me and was always there for me when I needed him. I thank my parents and parents-in-law for their love, moral support and encouragement.
DEDICATION

I dedicate this thesis to my parents, Hymavathi and Prjapathi Rao Gundavarapu.
# TABLE OF CONTENTS

LIST OF TABLES ........................................................................................................... ix

LIST OF FIGURES ......................................................................................................... x

CHAPTER

I. INTRODUCTION ........................................................................................................ 1

  Motivation ........................................................................................................... 2
  Background .......................................................................................................... 2
  Objectives ............................................................................................................ 3
  Overview ............................................................................................................. 3

II. ROUGH SET THEORY ............................................................................................ 4

  Information Systems ............................................................................................ 5
  Decision Tables ................................................................................................... 5
  Inconsistent Data Sets .......................................................................................... 6
  Upper and Lower Approximations ...................................................................... 6
  Certain and Probable Rules ................................................................................. 8

III. ROUGH MULTISETS AND INFORMATION MULTISYSTEMS .......................... 9

  Multiset Decision Tables .................................................................................... 10
 IX . CONCLUSIONS ...................................................................................................... 49

GLOSSARY ..................................................................................................................... 51

REFERENCES ................................................................................................................. 53
LIST OF TABLES

Table 2.1 – Decision Table ........................................................................................................... 7

3.1 - Information Multisystem ..................................................................................................... 11
3.2 – MDT for Table 2.1 ........................................................................................................... 12
3.3 – Lower Approximation of D2 ........................................................................................... 13
3.4 – Upper Approximation of D2 ........................................................................................... 13

5.1 – MDT with Sequence Numbers ......................................................................................... 22
5.2 – Table representing Upper (D1) ......................................................................................... 23
5.3 – AVT for Upper (D1) ......................................................................................................... 23

6.1 – Transformations for Credit Rating ................................................................................... 30
6.2 – Transformations for Income ............................................................................................ 30
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1 – Menu Tree</td>
<td>27</td>
</tr>
<tr>
<td>6.2 – Logon Screen</td>
<td>28</td>
</tr>
<tr>
<td>6.3 – Sample Contents for Credit_Applications Table</td>
<td>29</td>
</tr>
<tr>
<td>6.4 – Transformed Values for Credit Rating and Income</td>
<td>31</td>
</tr>
<tr>
<td>6.5 – Example MDT generated by RuleGen</td>
<td>34</td>
</tr>
<tr>
<td>6.6 - Rules</td>
<td>34</td>
</tr>
</tbody>
</table>
CHAPTER I

INTRODUCTION

Large volumes of data are acquired and stored into computer systems every day, in various applications, globally. It is hard to imagine a computer application that does not interface with a database or that does not dump megabytes of data into a database everyday. Data is generated constantly by a variety of applications ranging from basic day-to-day applications like utilities or telephone companies that track customer information to huge scientific research applications. The amount of data that is being acquired is growing exponentially by the minute.

It is hard for a human brain to digest all the data associated with an application and generate useful information or draw conclusions all by itself. This is where the power of machines is harnessed again. Computer scientists have come up with algorithms and techniques that can be deployed to crunch data in a database using very abstract, symbolic approaches like humans do. The field of discovering useful data in databases is a fast growing field in Computer Science and is known as “Knowledge Discovery and Data Mining” (KDD).

KDD can be used to establish trends or patterns in large databases. Properties of a data set can be precisely extracted and presented to the users, who can take the results further ahead to suit their needs. The application at hand concentrates on generating
decision rules for large databases. It allows the users to transform numeric attributes, define views of database tables and generate rules. These rules elicit patterns of the data in a table and can be used to assist experts in understanding and analyzing the trend of data in a table.

Motivation

There is an ever-increasing usage of database applications across the globe resulting in ever-expanding databases. Such databases not only hold raw, unprocessed data but also hold an equally large collection of facts buried within the data. There are few applications that help in cleansing, processing and analyzing data in such databases.

Traditional machine learning systems were not integrated with databases. But recent trends include development of learning systems that are tightly coupled with object-oriented, relational databases. This is a very positive development and results in advantages like mature data analysis, ease of integration of such systems with data warehouses and OLAP (On Line Analytical Processing) tools, to name a few.

Background

RuleGen is an implementation of the rule-generating algorithm, LERS-M as proposed by Chan [1]. This algorithm belongs to the LERS (Learning from Examples based on Rough Sets) class of learning systems [3] which are based on rough set theory [5, 6, 7]. LERS-M algorithm is based on rough multisets and information multisystems
This algorithm uses relational operations and can be tightly coupled with object-relational databases.

**Objectives**

The objective of this work is to thoroughly understand the LERS-M algorithm for learning rules from databases and implement the same into an application using an object-relational database. This application can be used as a powerful data analysis tool. It can be used to elicit hidden patterns or trends in a database table, classify future entries based on the records that already exist in a table. It allows users to identify and understand the dependency relationship between important attributes of a table. It also allows users to analyze a single database table from many perspectives. This capability is similar to the slice and dice capability of OLAP tools.

**Overview**

The remaining chapters of this thesis are organized as follows: The underlying concepts used in rule-generation are based on rough multisets and information multisystems. Chapters 2 and 3 discuss underlying concepts of rough set theory, rough multisystems and information multisystems. Chapter 4 discusses the LERS-M algorithm at length. Chapter 5 presents an example to elucidate the concepts presented in the earlier chapters. Chapter 6 presents an elaborate discussion on the RuleGen application and chapter 7 provides DB2-specific details of the implementation. Chapter 8 gives a brief overview of the components of the application. Chapter 9 contains the conclusions. Glossary of application-specific terms used in the thesis and references are also provided.
CHAPTER II
ROUGH SET THEORY

Introduced by Zdzislaw Pawlak [4] in the early 1980s, rough set theory is a mathematical tool that is devised to deal with vagueness and uncertainty. When there is a huge amount of data, it is very difficult to extract useful information from the data not only because of the sheer volume of data but also because the data might not be complete, i.e. there might be some critical piece of data that is missing. Hence, vagueness and uncertainty are at the very core of trying to extract useful information from very large databases. This makes rough set theory a fundamental approach to several disciplines of artificial intelligence like expert systems, machine learning, and knowledge discovery from databases, among others.

The fundamental assumption in rough set theory is that objects in any given domain are perceived only through values of attributes that can be evaluated on these objects. Since actual objects are not directly accessible, all objects that have same attribute values are indiscernible. Property of indiscernibility is very typical to database tables that capture facts about real world because the only available information about a particular object is in the data values of a record. Hence, when it comes to classification of objects, it is dependent on the available information about objects and not on the actual objects themselves.
Information Systems

To encapsulate knowledge about objects in a domain, the concept of Information Systems was formulated by Pawlak [5]. An Information System (IS) is a pair $S = (U, A)$, where $U$ is a non-empty finite set of objects called the Universe and $A$ is a non-empty finite set of attributes such that $a: U \rightarrow V_a$ for $a \in A$. $V_a$ is called the domain of attribute $a$ and defines the set of values that $a$ can have.

Decision Tables

Decision Tables [1] are a special case of Information Systems. They are characterized by two types of attributes namely, decision and condition attributes. Decision attribute represents the classification or partition of a data table. If there are $n$ distinct values for a decision attribute, it implies that the data table is classified into $n$ different groups or classes. The other set of attributes form the condition attributes whose values as a whole determine the value of the decision attribute.

An example decision table used in [8] is shown in Table 1. The first column represents object identifiers. An identifier is used to uniquely identify each row in the table. This decision table has six attributes namely A, B, C, D, E and F. Of these six attributes, attribute D is the decision attribute with values 1, 2 and 3. The other attributes are the condition attributes. In table 1, the universe $U$ consists of 28 objects. The partition of $U$ based on the values of decision attribute D is

$$X_1 = \{1, 2, 4, 8, 10, 15, 22, 25\}$$
$X_2 = \{3, 5, 11, 12, 16, 18, 19, 21, 23, 24, 27\}$

$X_3 = \{6, 7, 9, 13, 14, 17, 20, 26, 28\}$

$X_i$ is the set of objects whose value for decision attribute $D$ is $I$, for $i = 1, 2, 3$.

**Inconsistent Data Sets**

Data sets that have same condition values but different decision values are known as *inconsistent data sets*. These data sets are also referred to as *noisy data sets*. In table 1, objects 8 and 12 are inconsistent. Both of them have $(1, 1, 1, 1, 1)$ for their condition values but object 8 has value of 1 and object 12 has value of 2 for the decision attribute. Such inconsistencies are quite common in real world data tables. The following subsection discusses how Rough Set Theory got its arms around the problem of inconsistency.

**Upper and Lower Approximations**

Rough Set theory uses the concept of upper and lower approximations to encapsulate inherent inconsistency in real-world objects. Approximations are defined as follows:

Let $A = (U, R)$ be an *approximation space*, where $U$ is a nonempty set of objects and $R$ is an equivalence relation defined on $U$. Let $X$ be a nonempty subset of $U$. The *lower approximation* of $X$ by $R$ in $A$ is defined as

$$RX = \{e \in U \mid [e] \subseteq X\}.$$  

The upper approximation of $X$ by $R$ in $A$ is defined as:

$$RX = \{e \in U \mid [e] \cap X \neq \emptyset\}.$$
Table 2.1 – Decision Table

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>17</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>18</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>19</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>21</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>22</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>23</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>24</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>25</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>26</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>27</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>28</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>

In the above definitions, \([e]\) denotes the equivalence class containing \(e\). The difference \(\overline{RX} - RX\) is called the boundary set of \(X\) in \(A\). A subset \(X\) of \(U\) is said to be \(R\)-definable in \(A\) if and only if \(\overline{RX} = RX\). The pair \((\overline{RX}, RX)\) defines a rough set in \(A\), which is a family of subsets of \(U\) with the same lower and upper approximations as \(RX\) and \(\overline{RX}\). In terms of decision tables, the pair \((U, A)\) defines an approximation space.


When a decision class \( X_i \subseteq U \) is inconsistent, it means that \( X_i \) is not \( A \)-definable. In this case, classification rules can be found from \( AX_i \) and \( \overline{AX}_i \). These rules are called *certain rules* and *possible rules*, respectively. Using approximations, rough set theory, thus, provides the capability of addressing certainty as well as uncertainty inherent in real world data tables.

**Certain and Probable Rules**

The lower and upper approximations of a subset \( X \) of \( U \), the universe, define two types of rules on the subset [9]. Since the lower approximation includes all the objects that belong to \( X \), all rules defined on the lower approximation are called “certain rules”. Since the upper approximation includes objects that have a non-null intersection with \( X \), the rules defined on the upper approximation might not hold good for each and every elementary object of \( X \). Hence, these rules are called “probable rules”.


CHAPTER III

ROUGH MULTISETS AND INFORMATION MULTISYSTEMS

Introduced by Grzymala-Busse [7], information multisystems are represented using multisets instead of single sets. Multisets are defined as those sets in which an element may have several occurrences as opposed to just one in a regular set. The object identifiers that are used to uniquely identify objects in information systems are not used in information multisystems. The very definition of multisets implies the omission of object identifiers. Number of occurrences of each object is denoted in an additional column named $W$. Thus, information multisystems are more compact compared to the original information systems. Table 2 [1] is the multiset representation of the decision table represented in Table 1. Formally, an information multisystem can be defined as a triple, $S = (Q, V, \tilde{Q})$ where $Q$ is a set of attributes, $V$ is the union of domains of attributes $Q$ and $\tilde{Q}$ is a multirelation on $X \times V_q$.

In applying the concepts of lower and upper approximations in rough sets to multisets, the following definitions hold good. Let $M$ be a multiset and let $e$ be an element of $M$ whose number of occurrences in $M$ is $w$. The sub-multiset $\{w \cdot e\}$ will be denoted by $[e]_M$. Thus, $M$ may be represented as union of all $[e]_{M^e}$ where $e$ is in $M$. A multiset $[e]_M$ is called an elementary multiset in $M$. The empty multiset is elementary. A finite union of elementary multisets is called a definable multiset in $M$. Let $X$ be a submultiset of $M$. Then, the lower approximation of $X$ in $M$ is the multiset defined as $\underline{X} = \{e \in M \mid [e]_M \}$. 
The upper approximation of \( X \) in \( M \) is the multiset defined as \( X = \{ e \in U \mid \[ e \]_M \cap X \neq \emptyset \} \), where the operations on sets are defined by multisets. Therefore, a *rough multiset* in \( M \) is the family of all sub-multisets of \( M \) having the same lower and upper approximations in \( M \).

**Multiset Decision Tables**

Derived from an information multisystem, a Multiset Decision Table (MDT) [2] represents distinct decision values and their distributions in separate columns. The contents of this table are generated dynamically based on a decision table. In general, an MDT has twice as many additional columns as that of distinct decision values. MDT columns are divided into two sections namely, LHS (Left Hand Side) and RHS (Right Hand Side). LHS columns contain all the condition attributes and the number of occurrences of each row (\( W \) column). RHS columns consist of the distinct decision-value columns and their corresponding distributions. Decision-value columns have either 0 or 1 as values. A value of 1 in column \( Dv \) indicates that the given combination of condition attributes has the value of \( v \) for its decision attribute \( D \).

```plaintext
procedure generateMDT

Inputs: information multisystem \( S \)

Outputs: Decision Table (MDT) with sequence numbers

begin

Create a Multiset Decision Table (MDT) from \( S \) with sequence numbers;

for each decision value \( d_i \) of \( D \) do
```

10
begin

find the upper approximation $UPPER(d_i)$ of $d_i$;

Generate rules for $UPPER (d_i)$;

end;

end;

MDT for the information multisystem shown in Table 2 is represented in Table 3 [1]. LHS columns are A, B, C, D, E, F and W. Decision attribute D in Table 1 has three distinct values, 1, 2, and 3. Hence, the MDT has six RHS columns. D1, D2, D3 represent the boolean columns and w1, w2 and w3 represent the weights of the distinct decision values. Since column W represents the number of occurrences of each row, $w1 + w2 + w3 = W$.

**Rough Set Theory on MDTs**

Concepts of upper and lower approximations discussed in Section 2 can be extended to MDTs using relational operations. In terms of the LHS columns, lower approximation can be defined as those rows of the MDT where $D_i = 1$ and $W = w_i$. Upper approximation can be defined as those rows in MDT where $D_i = 1$ and $W \geq w_i$. Simply stated, upper approximation can be defined as those rows where $D_i = 1$.

Tables 3.3 and 3.4 represent the lower and upper approximations of decision-value pair (D, 2). The lower approximation is derived by selecting those rows where $D2 = 1$ and $W = w2$. 

11
Table 3.1 - Information Multisystem

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>E</th>
<th>F</th>
<th>D</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.2 – MDT for Table 2.1

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>E</th>
<th>F</th>
<th>W</th>
<th>D1</th>
<th>D2</th>
<th>D3</th>
<th>w1</th>
<th>w2</th>
<th>w3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
### Table 3.3 – Lower Approximation of D2

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>E</th>
<th>F</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

### Table 3.4 – Upper Approximation of D2

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>E</th>
<th>F</th>
<th>W</th>
<th>D1</th>
<th>D2</th>
<th>D3</th>
<th>w1</th>
<th>w2</th>
<th>w3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
CHAPTER IV

LERS-M ALGORITHM

This section presents the LERS-M algorithm [1] in detail. LERS-M takes as input a table with condition attributes \( C_1, C_2, \ldots, C_n \) and decision attribute \( D \). It then generates an MDT using basic SQL commands on the database table. The MDT generated also has sequence numbers to uniquely identify each row of the MDT. For each value \( d_i \) of the decision attribute \( D \), the upper approximation, \( \text{Upper}(d_i) \) is computed. A set of rules is generated for each \( \text{Upper}(d_i) \). The basic algorithm for LERS-M follows:

procedure \( LERS-M \)

Inputs: a table \( S \) with condition attributes \( C_1, C_2 \ldots, C_n \) and decision attribute \( D \).

Outputs: a set of production rules.

begin

Create a Multiset Decision Table (MDT) from \( S \) with sequence numbers;

for each decision value \( d_i \) of \( D \) do

begin

find the upper approximation \( \text{UPPER}(d_i) \) of \( d_i \);

Generate rules for \( \text{UPPER}(d_i) \);

end;

end;
Rule Generation Strategy

The actual implementation of the LERS-M algorithm presented above first generates an AVT (Attribute – Value pairs Table). AVT contains all the attributes in Upper \((d_i)\) along with the distinct values for each attribute, \((attribute, value)\) pairs. A grouping criterion is specified for AVT and the same criterion is used to rank the groups. In this implementation, number of rows in Upper \((d_i)\) with the same \((a, v)\) pair is used as the grouping criterion. Once the groups are identified in AVT, conjunctions of \((a, v)\) pairs with the same rank are generated, known as \textit{intra-group conjuncts}. \textit{Inter-group conjuncts} are generated by taking natural join over \((a, v)\) pairs of different groups. Conjuncts generated in the above step constitute the LHS of the rules. Validity of conjuncts generated also has to be tested. Generation and validation of conjuncts is encapsulated in a module called \textit{GenerateAndTestConjuncts}. A valid conjunct is minimized by dropping redundant conjuncts.

The RuleGen algorithm is an iterative one. The procedure starts off with Upper \((d_i)\) as the initial target set for the rules. After each iteration, a set of rules is generated and the instances covered by these rules are removed from target set. The algorithm terminates when all instances in Upper \((d_i)\) are covered. The algorithm is guaranteed to stop because upper approximations generated using the rough set theory are always finite.

In the procedures presented below, \([a, v]\) denotes the extension of an \textit{a-v} pair \((a, v)\), defined as the set of instances covered by the \textit{a-v} pair. This set will be a subset of the sequence numbers in the original \textit{MDT}. Nonempty finite set of \textit{a-v} pairs is defined as a
conjunct. The extension of a conjunct is the intersection of extensions of all the $a$-$v$ pairs in the conjunct where as the extension of a group of conjuncts is the union of extensions of all the conjuncts in the group, and the extension of an empty group of conjuncts is an empty set.

Procedure $RULE\_GEN$ [1] is the primary module that generates rules for each Upper ($d_i$). It uses two other procedures, namely, $GroupAVT (G, TargetSet)$ and $GenerateAndTestConjuncts$.

procedure $RULE\_GEN$

Inputs: an upper approximation of a decision value $d_i$, $UPPER (d_i)$ and an $MDT$.

Outputs: a set of rules for $UPPER (d_i)$.

begin

$TargetSet: = UPPER (d_i);$  

$Ruleset: = empty set;$  

$grouping criteria G: = degree of relevance;$  

Create an $a$-$v$ pair table $AVT$ containing all $a$-$v$ pairs in $UPPER (d_i);$  

while $TargetSet$ is not empty do

begin

$AVT: = GroupAVT (G, TargetSet);$  

$NewRules := GenerateAndTestConjuncts(AVT, UPPER(di));$  

$Ruleset := Ruleset + NewRules;$  

$TargetSet: = TargetSet – [NewRules];$  

end;
minimalCover(Ruleset); // Drop redundant rules from rule set
end; // RULE_GEN

Procedure GroupAVT (G, TargetSet) takes a grouping criterion and a subset of
Upper (d_i) to generate equivalent a-v pairs relevant to target set.

procedure GroupAVT

Inputs: a grouping criterion such as degree of relevance
a subset of the upper approximation of a decision value d_i.

Outputs: a list of groups of equivalent a-v pairs relevant to the target set.

begin

Initialize the AVT table to be empty;

Select a sub-table T from the target set where decision value = d_i;

Create a query to get a vector of condition attributes from the sub-table T;

for each condition attribute do // Generate distinct values for each attribute

begin

Create query string to select distinct values;

for each distinct value do

begin

Create a query string to select count of occurrences;

relevance := count of occurrences;

if (relevance > 0)

Add the condition-value pair to AVT table;
end; // for each distinct value
end; // for each condition attribute

Select the list of distinct values of the relevance column;
Sort the list of distinct values in descending order;
Use the list of distinct values to generate a list of groups of a-v pairs;
end; // GroupAVT

GenerateAndTestConjuncts procedure takes as input a list of equivalent AVT entries and Upper \((d_i)\). The outcome of running this procedure is the rules corresponding to the Upper \((d_i)\).

procedure GenerateAndTestConjuncts

Inputs: a list AVT of groups of equivalent a-v pairs
the upper approximation of decision value \(d_i\).

Outputs: a set of rules.

begin

RuleList := \(\emptyset\);

CarryOverList := \(\emptyset\); // a list of groups of a-v pairs

CandidateList := \(\emptyset\); // a list of rules

TargetSet := UPPER\((d_i)\);

// Generate Candidate List

repeat
\[ L := \text{getNext}(AVT); \]  // L is a list of equivalent a-v pairs

if (L is empty)
then break;

if (\text{conjunct}(L) \not\subset \text{TargetSet})
then Add \text{conjunct}(L) to \text{CandidateList};

// \text{conjunct}(L) returns a conjunction of all a-v pairs in L

if (\text{CarryOverList} is empty)
then Add all a-v pairs in L to \text{CarryOverList}
else begin

\text{FilterList} := \emptyset;

Add join (\text{CarryOverList}, L) to \text{FilterList};

/*@join is a function that creates new lists of a-v pairs by
taking and joining one element each from the
\text{CarryOverList} and L */

\text{CarryOverList} := \emptyset;

for each list in \text{FilterList} do

if ([\text{list}] \not\subset \text{TargetSet})
then Add list to \text{CandidateList}
else Add list to \text{CarryOverList};

end;

until (\text{CandidateList} is not empty);
// Test CandidateList
for each list in CandidateList do
begin
  // Drop conjuncts for minimal list
  list := minimalConjunct(list, UPPER(d_i)); Add list to RuleList;
end;
return RuleList;
end; // GenerateAndTestConjuncts

Procedure minimalConjunct is in turn called by GenerateAndTestConjuncts procedure to eliminate redundant conjuncts. It takes Upper (d_i) and a list of conjuncts generated by the calling procedure as inputs and returns a minimal conjunct list.

procedure minimalConjunct
Inputs: the upper approximation of decision value d_i,
a list conjuncts such that [list] ∈ UPPER(d_i)
Outputs: minimal conjunct list
begin
  TargetSet := UPPER(d_i);
  // Generate minimal conjunct list
  for each conjunct in list do
  begin
    newList := list – conjunct;
  end;
end;
if (newList ∈ TargetSet)
then begin
  list := newList;
  minimalConjunct(list, UPPER(d_i));
end;
else break;
end;
return list;
end;  // minimalConjunct
CHAPTER V

EXAMPLE

This section presents an example [1] to elucidate the procedures presented in the previous section. The following table represents an MDT of Table 3.2, with sequence numbers.

Table 5.1 – MDT with Sequence Numbers

<table>
<thead>
<tr>
<th>Seq</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>E</th>
<th>F</th>
<th>W</th>
<th>D1</th>
<th>D2</th>
<th>D3</th>
<th>w1</th>
<th>w2</th>
<th>w3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Upper approximation of D1 (i.e. the rows where decision attribute D has a value of 1) can be derived from the above table. It consists of rows that satisfy the condition of (D1 = 1). The following sub-table represents the upper approximation of (D1).
Table 5.2 – Table representing Upper (D1)

<table>
<thead>
<tr>
<th>Seq</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>E</th>
<th>F</th>
<th>W</th>
<th>D1</th>
<th>D2</th>
<th>D3</th>
<th>w1</th>
<th>w2</th>
<th>w3</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Attribute-Value pairs Table (AVT) generated for Upper (D1), using degree of relevance as the grouping criterion, is represented by Table 5.3. AVT represented below is sorted by the degree of relevance.

Table 5.3 – AVT for Upper (D1)

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
<th>Relevance</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>F</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>E</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>F</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

The following groups of a-v pairs are generated by procedure GroupAVT, using the degree of relevance as the grouping criterion:

\{(C, 1)\}
\{(B, 0), (E, 0), (F, 1)\}
{(A, 1)}
{(A, 0)}
{(B, 1), (E, 1), (F, 0)}
{(C, 0)}

GenerateAndTestConjuncts procedure uses the above list of AVT groups to generate and validate conjuncts. Target Set for Upper (D1) is a set of sequence numbers from Table 7 that is the set \{3, 4, 5, 7, 8, 11, 13\}. Intra-group group conjuncts are generated if a group has more than one \( (a, v) \) pair. Inter-group conjuncts are generated if the extension of the conjuncts at a degree of relevance is not a subset of Upper (D1).

The first group, \{(C, 1)\}, has a single \( (a, v) \) pair and hence, intra-group conjunct is not generated. Extension of \{(C, 1)\} is the set \{3, 4, 5, 8, 11, 12, 13\} which is not a subset of Upper (D1). Hence, inter-group join has to be generated with the next group \{(B, 0), (E, 0), (F, 1)\}. Inter-group and Intra-group conjuncts generated are shown below, along with their respective extensions.

\[
\begin{align*}
\{(C, 1), (B, 0)\} &= \{3, 4, 5, 8\} \\
\{(C, 1), (E, 0)\} &= \{3, 4, 5, 8, 11\} \\
\{(C, 1), (F, 1)\} &= \{4, 5, 8, 13\} \\
\{(B, 0), (E, 0), (F, 1)\} &= \{4, 7, 8\}
\end{align*}
\]

All the above conjuncts are valid since their extensions are indeed subsets of Upper (D1). Procedure minimalConjunct is used to minimize the conjuncts. This procedure is used to drop conjuncts in a list from left to right sequentially and test for validity of the remaining conjuncts. The result of executing this procedure on the above
list of valid conjuncts is that the conjunct (B, 0) in the intra-group conjunct {(B, 0), (E, 0), (F, 1)} is redundant since [{(E, 0), (F, 1)}] = {4, 7, 8}, which is a subset of Upper (D1).

The extension of all the new rules found so far is the set {3, 4, 5, 8, 11, 12, 13} which covers all the instances of Upper (D1). Therefore, the procedure GenerateAndTestConjuncts terminates. The rules generated are listed below:

(C, 1) & (B, 0) → (D, 1)

(C, 1) & (E, 0) → (D, 1)

(C, 1) & (F, 1) → (D, 1)

(E, 0) & (F, 1) → (D, 1)
CHAPTER VI

RULEGEN APPLICATION

RuleGen application generates rules from database tables for the users. This application was coded for IBM’s DB2 UDB database, using J2EE technology and Apache-Tomcat application server. Even though the application is tied to a specific database, the concept can be generalized and extended to any other relational database. Section 7 discusses details of the application that are DB2 specific. Since Java is used as the programming language, portability of the source code is not an issue at all.

Navigation

Different capabilities of the RuleGen application are accessible to the users from the drop down menu bar that is present at the top of the screen. The main menu options are listed left to right, following the logical order of events in the application, although the user is free to jump to any menu option at any time. Figure 1 displays the menu tree.

User Interface Details

After a menu option is chosen, the flow of logic takes a few screens before to complete the intended operation. If the user jumps to another menu option before completion of the currently running step, the chosen operation is ended abruptly. Successful completion of a logical process is communicated to the user with a message at
the end of the process. User can then move on to another logical step in the application. Errors are communicated to the user in case something goes wrong at the database level.

Since a logical process spans across multiple screens, taking different inputs from the user at each screen, users can navigate to the next screen by clicking on the “Next” button at the bottom of the screen. Input entered so far can be reset by clicking on the “Reset” button at the end of the screen.

**RuleGen Logon**

RuleGen’s home page prompts the users to log on to a DB2 UDB database. Once the user logs in successfully, various options of the application are presented to the user in the form of drop down menus at the top of the screen. There are three main capabilities for the user to choose from, namely, Transformations, View Management and Rules.
To gain a better understanding of the capabilities of RuleGen, an example table will be very helpful. Let us consider “Credit Applications Table” that holds data about credit applications. Possible columns in this table would be First Name, Last Name, Mother’s Maiden Name, Social Security Number, Credit History, Credit Rating, Annual Income and Approved (i.e. if the credit application is approved or not). Let us assume that valid values for Credit Rating column range from 1 through 10; rating of 1 being Bad and 10 being Excellent. This example table will be referred to in the following subsections wherein the capabilities of RuleGen are discussed in detail.

Transformations

Many tables contain numeric columns, i.e. columns that contain only numeric values (integers or real numbers). In the Credit_Applications example table, Credit
Rating and Income are the numeric columns. Consider the following rule generated using these numeric columns in the example table:

\[(\text{Credit Rating}, 5) \& (\text{Annual Income}, 35000) \rightarrow (\text{Approved}, \text{No}).\]

This is where transformations come in handy. Transformations allow the users to transform or map numeric values in a database table into more user friendly values. Using this capability in RuleGen, users can define ranges for numeric values and assign a name to each range. Considering the Credit Applications table and its two numeric columns, Credit Rating and Income, respectively, a user can specify the following ranges:
Table 6.1 – Transformations for Credit Rating

<table>
<thead>
<tr>
<th>Range Name</th>
<th>Lower Limit</th>
<th>Upper Limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>High</td>
<td>6</td>
<td>10</td>
</tr>
</tbody>
</table>

Table 6.2 – Transformations for Income

<table>
<thead>
<tr>
<th>Range Name</th>
<th>Lower Limit</th>
<th>Upper Limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>2500</td>
<td>55000</td>
</tr>
<tr>
<td>R2</td>
<td>5500</td>
<td>90000</td>
</tr>
<tr>
<td>R3</td>
<td>9000</td>
<td>13500</td>
</tr>
</tbody>
</table>

RuleGen compares each individual value within the column to the ranges input by the user. Once it finds a range which encompasses the value, it assigns the name of that range to that value. Using these range names in place of the actual numeric values, RuleGen then builds a view of the underlying table. Figure 4 shows sample contents of such a view generated by RuleGen. In this example, Credit Rating and Income columns have been transformed.

Using these transformed values for generating rules, the following rule, (Credit Rating, 5) & (Annual Income, 35000) → (Approved, No) will be transformed as follows: (Credit Rating, Low) & (Annual Income, R3) → (Approved, No). The rule using transformed values has more clarity compared to the rule generated using numeric values alone.
Not all columns in a table play an important role in generating rules. Some columns might be more informational while others might be more critical. Considering the columns in the Credit Applications table, the most important columns used in processing a credit application will be Credit Rating, Credit History and Annual Income. These columns decide the value of the Approved column. Columns like First Name and Last Name, SSN, Mother’s Maiden Name, etc, do not play a role in the decision process of approving or denying a credit application.

Even among the important columns of a table, analysts might want to study how the behavior of one particular column or certain combination of columns affects the decision making process. For example, analysts might want to understand how Credit History affects the Approval process or how Credit Rating and Credit History together affect the Approval process.
Taking this fact into account, RuleGen provides the capability of selectively including columns for rule generation. To achieve this functionality, RuleGen allows users to define “views” for a database table. These views are virtual views, i.e. they are not actual database views. Therefore, this capability does not create additional load on the database. While defining these views, users can selectively include columns from the underlying database table to be used for rule generation. Any number of such views can be defined for a single database table. Users can select any particular view they defined and use that for rule generation. The rules thus generated elicit the relationship between the columns contained in the view alone. This does not eliminate the capability of selecting the original database table itself for rule generation.

Users can define two types of views namely, Base View and Decision View. Base View contains all the required columns specified by the user from the underlying table. A Decision View is defined over a base view. In a decision view, the decision column has been clearly identified. A decision column is the one whose value depends upon the values of the remaining columns in the view or the table. Columns other than the decision column are known as the condition columns.

Since View Management allows users to define any number of base views for a database table and any number of decision views over a base view, it is in essence similar to the slice and dice capabilities of the OLAP tools. This capability allows the users to thoroughly analyze the dependencies between columns of a table.
Rules

Rules elicit the dependency relationship between the columns of a table and are the final deliverable of RuleGen application. A rule is of the format:

$$(CC1, V1) \land (CC2, V2) \land \Rightarrow DC, VD$$

The above rule can be interpreted as follows: When Condition Column CC1 has a value of V1 and Condition Column CC2 has a value of V2, then Decision Column, DC, will have a value of VD. The left hand side of the rule could have one or many condition columns, depending on the dependency relationship that the algorithm comes up with.

Rules thus generated for a particular database table represent patterns among the existing records of the table. They classify the records of the table into categories based on the dependency relationships. Using these classifications as the base, it is possible to classify future entries of the table. For example, if there is a new credit application with a low Credit Rating, from the rules already generated, it is easily predictable that the application will not be approved.

Since the rules represent only main classification of records of a table, experts will have a smaller set of data to review compared to all the records of the table. Thus, rules can be used to summarize values.
Figure 6.5 – Example MDT generated by RuleGen

Figure 6.6 - Rules
CHAPTER VII

DB2-SPECIFIC IMPLEMENTATION DETAILS

RuleGen application was implemented on IBM DB2 Universal Database (Version 7). There are certain aspects of the implementation that are specific to the database used. This section gives an overview of the database-specific sections of the implementation.

System Data

During the process of defining transformations, base views and decision views, RuleGen application displays tables and/or views owned by the user. Each table or view is displayed as a hyperlink, clicking on which launches another browser window that displays columns and data types of the corresponding table or view. Information about tables and/or views, columns and their data types is maintained automatically by the UDB system in the system catalog tables. These tables contain metadata, i.e. details about data stored in the database. These catalog tables are defined under SYSIBM schema.

Two views are defined on the tables of this schema, namely, SYSCAT and SYSSTAT. SYSCAT catalog views are not updatable while SYSSTAT catalog views are updatable by authorized users and can be used to change statistics used for query optimization. For RuleGen application, we are concerned only with SYSCAT schema.
Every table in SYSIBM schema has a corresponding view in SYSCAT schema. All the data required to display details about tables, views, columns and data types is obtained from views in the SYSCAT table.

Tables / Views

The name of the SYSCAT view that stores information about tables and views is *Tables*. Each row of this view represents a table, view or alias. Both system as well as user-defined tables and views are represented in this view. Details like schema, name, type (table, view or alias), number of columns and rows, creation time are stored about each entry, along with other details. RuleGen queries this view to display a list of tables and/or views to the user.

Columns

SYSCAT view that stores information about columns of a table or view is *Columns*. Columns of catalog tables and views as well as user-defined tables and views are listed in this view. Details like table schema, table name, column name and number, data type details, default value, null values allowed or not are stored about each column, among other details. RuleGen application uses this view to display columns and their data types to the user. It also uses this view to determine columns on which transformations can be defined because only columns with numeric data type can be transformed.
CHAPTER VIII
IMPLEMENTATION

RuleGen application uses DB2 Universal Database as the backend and Java Servlet Pages (JSPs) and Servlets as the front-end. JSPs were used to display dynamic web content and servlets were used to provide dynamic interaction with the database server. Apache-Tomcat 3.2.4 was chosen as the Application Server but it could have been any Servlet container that supports JSPs and servlets. All JSPs and servlets are deployed to Tomcat server, which was configured to work as a standalone Web server. To store the application details like transformations, base and decision views, five DB2 tables are created by the application. The following subsections give a brief overview of the DB2 tables, JSP and servlet components of this application.

JSP Components

This section lists all the JSP components used in RuleGen application. These components are listed in alphabetical order and this order is not necessarily in line with the order in which these components are called by the application.

BaseViewAttributes.jsp

From the screen that displays a list of base views, each as a hyperlink, this page displays the corresponding base view attributes in a new window when the users click on any particular base view. This page serves as a window to the base view, giving the users
more details about the base views, before they can select any particular base view for further processing, i.e. for creating a decision view.

BaseViewColumns.jsp

Users are navigated to this JSP page when they choose to display the attributes of a Base View.

BaseViewList.jsp

This page displays the list of baseviews to the user as hyperlinks. Clicking on a base view opens another browser window that displays attributes of the base view.

BaseViewTables.jsp

A list of tables is displayed to the user as the first step in the process of defining base views by this page.

DecisionAttributes.jsp

This servlet prompts the user for decision view name and decision attribute for defining a decision view based on user’s selection of base view in the previous screen.

DecisionTableList.jsp

This page displays decision tables, i.e. decision views materialized as views in the database to the user. The user is prompted to select one decision table to for defining a MDT from the Rules ➔ Generate MDT menu path.

DecisionViewAttributes.jsp

Decision and condition attributes are displayed to the user in a separate browser window by this JSP page, when the user clicks on a decision view.
DecisionViewList.jsp

A list of decision views is displayed to the user as a list of hyperlinks. When the user clicks on a decision view, the details of the decision view

Errorpage.jsp

This JSP page is called from all the servlets described in the following subsection, in case an error is flagged while processing the user’s request. Servlets check for several possible errors and an appropriate error message is passed on to this JSP page. This page then displays the error to the user.

Frames.jsp

After successful login, this is the page that is displayed to the user. This page, as the name indicates, contains two frames. The first frame has the menu for the application and is a standard frame that the user sees always. The source of this frame is an HTML page called menu.htm. The second frame of this particular page is the welcome page and provides details of the application to the user. The source of this page is again another HTML page called home.htm. This frame is the varying frame and becomes the target frame in which data pertaining to different menu options is displayed, as selected by the user.

Index.jsp

This is the first page of the RuleGen application. Here the user is prompted to provide login information for the database.
MDTList.jsp

This JSP page displays a list of MDTs to the user, from the Rules → Learn menu path so that (s)he can select one MDT for rule generation.

Rules.jsp

Actual rules for a selected MDT are displayed to the user from this page. The generation of rules is done in the Rules/Rules servlet.

TableColumns.jsp

This page displays list of columns of a table and corresponding datatype of each column, when the users click on a hyperlink of a table. This page can be invoked from the list of tables displayed for defining transformations and base views.

TransColumns.jsp

TransColumns displays transformable columns in the table selected by the user to define transformations, i.e. numeric columns, their actual data types and minimum and maximum values. All the columns are displayed with check boxes against them and a text field to let the user enter number of transformations (s)he wishes to define for that column. If the user wishes to define transformations on a particular column, the user must check the box against the column and enter number of transformations in the corresponding text box.

TransRanges.jsp

This page displays columns that the user chose to transform in the previous screens along with minimum and maximum values. The user is prompted to enter range
name and range limits for each of the number of ranges per column input by the user in
the previous screen.

TransTables.jsp

This page displays a list of tables to the user when they navigate to the
Transformations → Define menu option. Each table is displayed as a clickable hyperlink,
clicking on which provides users with information about attributes in a different browser
window.

Servlets

DB2Connectivity/Login.java

This is the first servlet that is to be executed in the application. It reads in the user
name, password and database name from the first screen of the application. It then
retrieves the name of the database server and port from the configuration file called
Login_Config.properties, residing in the same directory, DB2Connectivity. After
establishing a successful connection to the database using the user name and password
provided by the user, the application creates five tables that are transparent to the users,
to store application-specific details. These tables are discussed in the RuleGen Tables
subsection. After successful creation of these tables, Frames.jsp is called from this
servlet.

Baseviews/Columns.java

This servlet retrieves all the columns and their associated data types from the
SYSCAT.Columns view, of a table selected by the user in the previous screen, in the
process of defining a base view. In addition to retrieving the regular columns, all those
columns of the chosen table that have transformations defined are also retrieved. This
information is stored in a vector and passed on to the BaseViewColumns.jsp page.
Baseviews/Definition.java

This servlet creates the base view and saves the details in the RuleGen
application’s tables, according to the specifications provided by the user in the
BaseViewColumns.jsp screen. The name of the base view is saved in called
RuleGen.Meta_Base and the columns in the base view are saved in the
RuleGen.Meta_Base_Columns. A message is displayed to the user indicating the
successful creation of the base view.

Decisionviews/BaseViewAttributes.java

This servlet retrieves the columns of a base view either selected by the user by
clicking on the radio button next to the base view or by clicking on the base view name
displayed as a hyperlink and passes on this information to different JSP pages,
accordingly. In the former case, DecisionViewAttributes.jsp page is used to obtain the
information from the suer for defining a decision view. In the latter case,
BaseViewAttributes.jsp page is used to display details of the base view to the user in a
different browser window.

Decisionviews/BaseViewList.java

This servlet retrieves a list of base views from RuleGen.META_BASE table and
passes on this list to the BaseViewList.jsp page to be displayed to the user. This servlet is
invoked when the user goes to View Management → Define → Decision View menu path.
Decisionviews/DecisionViewAttributes.java

    When a user clicks on a decision view displayed as a hyperlink in the View Management → Create View menu path or on a decision table in the Rules → Learn menu path, this servlet retrieves the decision and condition attributes of a decision view or table accordingly. This information is passed on to the DecisionViewAttributes.jsp page.

Decisionviews/DecisionViewList.java

    This servlet is invoked from two different menu paths namely View Management → Create View and Rules → Generate MDT. In the former menu path, a list of decision views that have not been materialized is retrieved from the application’s META_DECISION table and passed on to the DecisionViewList.jsp page. In the latter menu path, those decision views defined and materialized by the user, i.e. those that have been defined as views in the database are retrieved and passed on to the DecisionTableList.jsp page.

Decisionviews/Definition.java

    This servlet creates the decision view and saves the details in the RuleGen application’s table, META_DECISION and displays a message to the user indicating the successful creation of the decision view.

Decisionviews/Materialize.java

    After the user chooses a particular decision view to be materialized, i.e. to be created as an actual view in the database from the menu path View Management →
Create View, this servlet creates the view and then displays a message to the user upon successful creation of the view. A materialized decision view is called a decision table.

Exceptions/InvalidInputException.java

This is an exception class defined to capture invalid input entered by users. This exception is called from different servlets that read user input and validate it before further processing. In case the input is not valid in that particular context of processing, the servlet uses raises this exception to notify the user.

Rules/AVT.java

This servlet represents the Attribute-Vector Table (AVT) class used in the process of generating rules for a particular MDT. AVT class is instantiated in the Rules/Rules servlet.

Rules/MDT.java

A Multiset Decision Table (MDT) is created by this servlet based on the decision table name selected by the user from the Rules \( \rightarrow \) Generate MDT menu path. Once the MDT is created, a message is displayed to the user accordingly.

Rules/MDTList.java

A list of Multiset Decision Tables is retrieved by this servlet and passed on to the MDTList.jsp page to be displayed to the user, from the Rules \( \rightarrow \) Learn menu path. Each MDT is displayed as a hyperlink, clicking on which launches another browser window with the attribute details of the underlying decision table. This is achieved by the decisionviews/DecisionViewAttributes servlet.
Rules/Rules.java

This servlet is the servlet that generates rules for a MDT. It uses Rules/AVT class to create instances of AVT objects and methods from Rules/Utilities servlet class in the process of generating rules. The rules are passed on to the Rules.jsp page to be displayed to the user.

Rules/Utilities.java

This servlet contains several utility methods used by the Rules servlet. These methods include methods for building an AVT for a given MDT, sorting and AVT, dropping redundant conditions, among others.

Transforms/TableColumns.java

A list of columns along with the corresponding data types is retrieved to be displayed to the user by this servlet, when the user clicks on a table. The list of tables is presented to the user from different menu paths like Transformations ➔ Define and View Management ➔ Base Views ➔ Define. This information is passed on to the TableColumns.jsp, that displays the details in a separate browser window.

Transforms/Tables.java

This servlet retrieves a list of tables that the user owns from the SYSCAT.Tables view and builds a vector of these tables. It is invoked from two different menu paths. One way is when the user is in the process of defining transformations from the Transformations ➔ Define menu path. The other way is when the user is in the process of defining a base view from the View Management ➔ Define ➔ Base View menu path. In the course of first menu path, the vector of tables is passed on to the TransTables.jsp
and in the second menu path, it is passed on to the BaseViewTables.jsp, to be displayed to the user.

Transforms/TransColumns.java

This servlet determines the table selected by the user in the previous screen wherein the user is displayed a list of tables to select from and retrieves numeric columns of the table from the SYSCAT.Columns view. These columns, their data types along with minimum and maximum values are stored in a vector and passed to TransColumns.jsp.

Transforms/TransRanges.java

Defined in the Form Action tag of TransColumns.jsp, this servlet reads in the values entered by the user for each of the transforming columns and writes that information to RuleGen application’s table that saves transformation details like transformation id, table name and schema, column name and number of ranges – RuleGen.Meta_Trans.

Transforms/TransTransform.java

In this servlet, all the range details are read and saved into RuleGen applications table that holds range information – RuleGen.Meta_Ranges. Once the ranges are successfully defined, a message accordingly is displayed to the user.

RuleGen Tables

After the user logs in successfully, five tables are created to store application-specific data. All these tables are created under the schema named RULEGEN, to
distinguish the application’s tables from other user-owned tables. All the id columns in these tables are generated automatically when a row is saved to these tables.

META_TRANS and META_RANGES tables are used to save details about transformations and ranges. META_TRANS is used to store details about a transformation. The columns of the table include transformation id, transformation name, column name, table name and schema to which the column belongs to and the number of ranges specified by the user. META_RANGES is the table that actually contains the details of the ranges for each column specified in META_TRANS table. This table’s columns include range id, transformation id, range name, lower and upper limits of the range. Transformation id is the foreign key to this table and is derived from the parent table, META_TRANS.

META_BASE, META_BASE_COLUMNS and META_DECISION tables are used to save details about views of the application. META_BASE has just two columns and stores name of the base view along with the base view id. META_BASE_COLUMNS contains actual columns of the base view defined in the above table. It’s columns include base view id from the parent table META_BASE, column name, table name and schema to which the column belongs to and transformation id, in case the column is not a regular column but a transformation column.

META_DECISION is used to store the details of a decision view and has four columns namely, base view id which is derived from the META_BASE table, decision view id, decision view name and a flag called “Materialized”. This column takes only one of the two values, Y or N. Y indicates the decision view has been materialized, i.e. a real
view has been created in the user’s database. If it has a value of N, it indicates that the view has not yet been materialized.
CHAPTER IX

CONCLUSIONS

RuleGen application implements the LERS-M algorithms for generating rules from database tables, as proposed by Dr. C-C. Chan. This algorithm belongs to the LERS (Learning from Examples based on Rough Sets) class of algorithms and uses rough multisets and information multisystems. Using this application, one can understand trends or patterns in a database table. Such an application can be used as a tool in the domain of KDD (Knowledge Discovery and Data mining), for discovering useful information in large databases.

RuleGen provides several different capabilities. Users can view their tables and define transformations on numeric columns converting the numbers into more user-friendly values. A long list of numbers can be divided into ranges with meaningful names using this functionality. For example, values in “Sales” column can be divided into ranges named Low, Medium and High.

Each table could have several attributes, many of which may not influence the actual value of the decision attribute. So, RuleGen enables users to determine what columns or attributes they would like to include in the process of generating rules. Users can define base views to include important attributes of a table and then define one or more decision views, indicating a decision attribute, based on a base view. This provides
the users with a high level of analyzing capabilities as there is no restriction on the

number ways of views that can be defined on a single table.

Decision views can be materialized i.e. created as actual views in the database and the users can learn or generate rules for such materialized decision views. The rules show the dependency of the decision attribute values on the condition attribute values. Thus, RuleGen provides the users with a full suite of capabilities to analyze and establish trends or patterns in database tables, making it a valuable tool for data mining.
GLOSSARY

Base View
A subset of attributes or columns from an actual database table that are deemed important in rule generation and are grouped together as a base view, under a common name.

Condition Attributes
Attributes that are on the left hand side of a rule and that together determine the value of the dependent or decision attribute.

Decision Attribute
Attribute that is on the right hand side of a rule. The value of this attribute is dependent on the values of attributes on the right hand side of the rule.

Decision Table
A decision view that has been materialized.

Decision View
A base view in which the decision and condition attributes have been clearly identified.

Materialized View
A view definition that is created as an actual view in the database. The data in the view contains data from the table as of the time the view is materialized. Any changes to
the data in the underlying table that are done after the view creation are not captured in the view.

Multiset Decision Table (MDT)

Multiset Decision Table (MDT) represents distinct decision values and their distributions in separate columns.

Rule

A logical implication of attributes of a table, of the form \( C \rightarrow D \), defining the dependency of the values of attributes on the right to the values of attributes on the left.

Transformations

Definition of ranges and range names for numerical values of a column so that these numeric values are mapped into range names in rules, making them more intuitive to the end users.
REFERENCES


