

國立政治大學資訊科學系
Department of Computer Science
National Chengchi University

碩士論文

Master's Thesis

以規則為基礎的分類演算法：應用粗糙集
A Rule-Based Classification Algorithm:
A Rough Set Approach

研究生：廖家奇

指導教授：徐國偉

中華民國一百零一年七月

July 2012

以規則為基礎的分類演算法：應用粗糙集

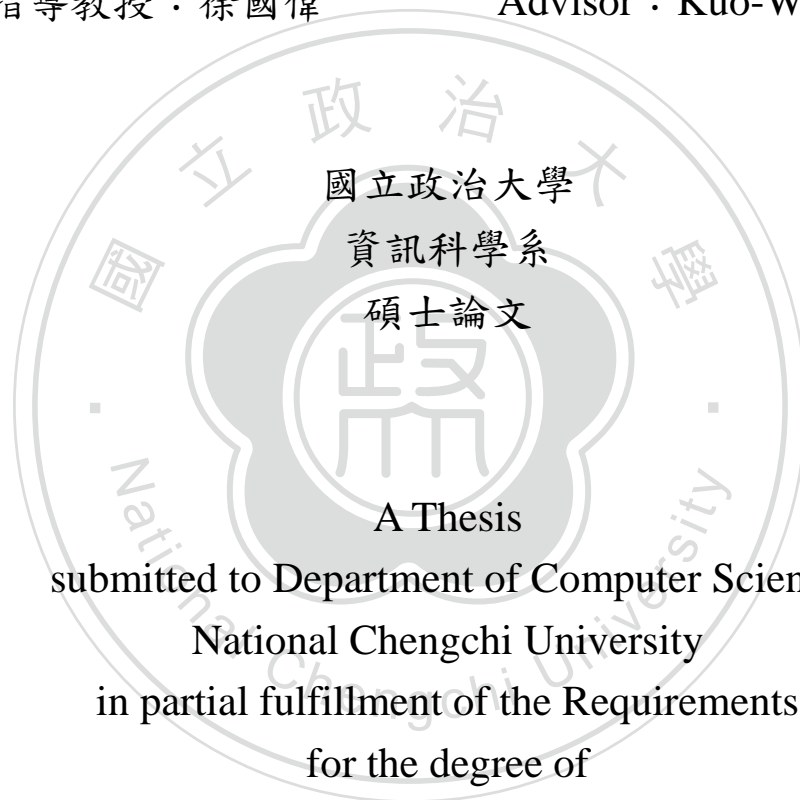
A Rule-Based Classification Algorithm:
A Rough Set Approach

研究生：廖家奇

Student : Chia-Chi Liao

指導教授：徐國偉

Advisor : Kuo-Wei Hsu



A Thesis
submitted to Department of Computer Science
National Chengchi University
in partial fulfillment of the Requirements
for the degree of
Master
in
Computer Science

中華民國一百零一年七月

July 2012

Abstract

In this thesis, we propose a rule-based classification algorithm named ROUSER (ROUgh SEt Rule), which uses the rough set theory as the basis of the search heuristics in the process of rule generation. We implement ROUSER using a well developed and widely used toolkit, evaluate it using several public data sets, and examine its applicability using a real-world case study.

The origin of the problem addressed in this thesis can be traced back to a real-world problem where the goal is to determine whether a data record collected from a sensor corresponds to a machine fault. In order to assist in the root cause analysis of the machine faults, we design and implement a rule-based classification algorithm that can generate models consisting of human understandable decision rules to connect symptoms to the cause. Moreover, there are contradictions in data. For example, two data records collected at different time points are similar, or the same (except their timestamps), while one is corresponding to a machine fault but not the other. The challenge is to analyze data with contradictions. We use the rough set theory to overcome the challenge, since it is able to process imperfect knowledge.

Researchers have proposed various classification algorithms and practitioners have applied them to various application domains, while most of the classification algorithms are designed without a focus on interpretability or understandability of the models built using the algorithms. ROUSER is specifically designed to extract human understandable decision rules from

nominal data. What distinguishes ROUSER from most, if not all, other rule-based classification algorithms is that it utilizes a rough set approach to select features. ROUSER also provides several ways to decide an appropriate attribute-value pair for the antecedents of a rule. Moreover, the rule generation method of ROUSER is based on the separate-and-conquer strategy, and hence it is more efficient than the indiscernibility matrix method that is widely adopted in the classification algorithms based on the rough set theory.

We conduct extensive experiments to evaluate the capability of ROUSER. On about half of the nominal data sets considered in experiments, ROUSER can achieve comparable or better accuracy than do classification algorithms that are able to generate decision rules or trees. On some of the discretized data sets, ROUSER can achieve comparable or better accuracy. We also present the results of the experiments on the embedded feature selection method and several ways to decide an appropriate attribute-value pair for the antecedents of a rule.

摘要

在本論文中，我們提出了一個以規則為基礎的分類演算法，名為 ROUSER (ROUgh SEt Rule)，它利用粗糙集理論作為搜尋啟發的基礎，進而建立規則。我們使用一個已經被廣泛利用的工具實作 ROUSER，也使用數個公開資料集對它進行實驗，並將它應用於真實世界的案例。

本論文的初衷可被追溯到一個真實世界的案例，而此案例的目標是從感應器所蒐集的資料中找出與機械故障之間的關聯。為了能支援機械故障的根本原因分析，我們設計並實作了一個以規則為基礎的分類演算法，它所產生的模型是由人類可理解的決策規則所組成，而故障的徵兆與原因則被決策規則所連結。此外，資料中存在著矛盾。舉例而言，不同時間點所蒐集的兩筆紀錄極為相似、甚至相同（除了時間戳記），但其中一筆紀錄與機械故障相關，另一筆則否。本案例的挑戰在於分析矛盾的資料。我們使用粗糙集理論克服這個難題，因為它可以處理不完美知識。

研究者們已經提出了各種不同的分類演算法，而實踐者們則已經將它們應用於各種領域，然而多數分類演算法的設計並不強調演算法所產生模型的可解釋性與可理解性。ROUSER 的設計是專門從名目資料中萃取人類可理解的決策規則。而 ROUSER 與其它多數規則分類演算法不同的地方是

利用粗糙集方法選取特徵。ROUSER 也提供了數種方式來選擇合宜的屬性與值配對，作為規則的前項。此外，ROUSER 的規則產生方法是基於 separate-and-conquer 策略，因此比其它基於粗糙集的分類演算法所廣泛採用的不可分辨矩陣方法還有效率。

我們進行延伸實驗來驗證 ROUSER 的能力。對於名目資料的實驗裡，ROUSER 在半數的結果中的準確率可匹敵、甚至勝過其他以規則為基礎的分類演算法以及決策樹分類演算法。ROUSER 也可以在一些離散化的資料集之中達到可匹敵甚至超越的準確率。我們也提供了內建的特徵萃取方法與其它方法的比較的實驗結果，以及數種用來決定規則前項的方法的實驗結果。

誌謝

人的基因之中，或多或少都存著探索的天性；旅遊可以讓我們探索不同的風景地貌與文化風俗，而我念研究所也是為了探索，為了窺探人們如何進行研究。短短兩年過去了，我看到了冰山一角，卻已經要告別親愛的師長與同學，以及讓我成長的學校。

研究對我而言並不是一件簡單的事，大學四年我望著這個領域充滿好奇，期待有一天我也能對研究略知一二，抱著這份憧憬，大學時代的專題是我第一次對研究的初探，感謝沈錕坤教授在這段時間的悉心指導；進入研究所之後，由於對資料探勘抱持興趣，但學校的訓練無法滿足我對處理實務問題的渴望，因此我很感謝王智中博士能提供我機會接觸真實的案例，帶著鋼盔進入工廠，了解資料如何蒐集，並嘗試找出與故障相關的因果關係。在這段日子裡，為了對背景知識有更多了解，成大圖書館便成了我每周報到的好去處，雖然我沒念過成大，卻也感謝成大對我的間接指導。

研究所的日子進行至此，我已不只是在探索如何進行研究，更是在陌生的工廠與大學之間探險，感謝本人的指導教授徐國偉老師憑著熱忱與經驗，接受我那異於常規的學習路線，不倦地導正我的航向，並給我極大的發揮空間，最後能順利完成學位。

我的好同學林宏哲是我在實驗室中的最佳飯友，而他深刻的哲學思考與商業知識總是能在餐後活絡我那迷失於研究的大腦；而我在研究所修最多的是劉昭麟教授開的課，也很感謝劉教授總是能回應我下課後的怪問題。感謝我的父母與家人這些年來的默默支持，讓我跌倒之後還有爬起的機會，也感謝政大，給我一個良好的環境。非常感謝口試委員們蒞臨指導，學生深感榮幸。

本研究是在國科會計畫 NSC 100-2218-E-004-002 的補助下完成的，特別感謝這份支持，讓所有實驗皆能如期完成。

廖家奇

民國 101 年 7 月 23 日

Table of Contents

CHAPTER 1 INTRODUCTION.....	1
1.1 Classification Problem.....	1
1.2 Tree-Based and Rule-Based Classification Algorithms	2
1.3 Rough Set Based Classification Algorithms.....	3
1.4 Data Mining.....	4
1.5 Thesis Organization.....	5
CHAPTER 2 PRELIMINARY.....	6
2.1 Rule-Based Classification Algorithms	6
2.1.1 The Basics.....	6
2.1.2 Separate-and-Conquer	7
2.1.3 Search Heuristics	7
2.1.4 Pruning and Optimization.....	9
2.2 The Rough Set Theory.....	9
2.2.1 Information System and Decision Table.....	10
2.2.2 Indiscernibility Relation	11
2.2.3 Rough Set	12
2.2.4 Reduct and Core	14
2.2.5 Indiscernibility Matrix.....	17
CHAPTER 3 DESIGN OF THE PROPOSED METHOD.....	18
3.1 Potential Boundary Region and Discernibility Power.....	18
3.2 ROUSER	22
CHAPTER 4 IMPLEMENTATION OF THE PROPOSED METHOD.....	28

4.1 WEKA	28
4.1.1 Import Data.....	28
4.1.2 Classifier.....	28
4.1.3 Cross-Validation	29
4.2 Data Structure	29
4.2.1 Rough Set	29
4.2.2 Decision Rule	30
4.3 ROUSER	32
4.3.1 BuildClassifier()	32
4.3.2 OneClassRule().....	32
4.3.3 grow()	34
4.3.4 BestAntd()	35
CHAPTER 5 EXPERIMENT AND RESULTS	36
5.1 Environmental Setting	36
5.2 Data Sets.....	36
5.3 Results	39
5.4 Summary.....	49
CHAPTER 6 CASE STUDY	50
6.1 Introduction	50
6.1.1 Back Ground Knowledge	50
6.1.2 Problem.....	51
6.1.3 Data.....	51
6.2 Model Building.....	52
6.2.1 Data Preprocessing	52

6.2.2 Classification Algorithms	53
6.3 Inspect the Result Rules	54
6.4 Summary.....	56
CHAPTER 7 CONCLUSIONS AND FUTURE WORK.....	57
REFERENCE	58



List of Figures

Figure 1. Cooperative data analysis.....	5
Figure 2. The SEPERATE&CONQUER algorithm.	7
Figure 3. (a) Classic set. (b) Rough set. (c) Rough set for example.....	12
Figure 4. (a) The space of $d=y$. (b) The lower-approximation of $d=y$. (c) The upper-approximation of $d=y$	13
Figure 5. (a) The data space. (b) The positive region of U/D	14
Figure 6. (a) The new decision table with a_3 . (b) Data space of $a_3=1$. (c) Data space of $a_3=2$. (d) Data space of $a_3=3$	15
Figure 7. a_1, a_3 as the reduct.....	16
Figure 8. a_2, a_3 as the reduct.....	16
Figure 9. An example of indiscernibility matrix.	17
Figure 10. The redefined rough set.....	19
Figure 11. (a) Decision table $A' = (U, C \cup D - \{a_2\})$. (b) The new rough set of $d=y$	20
Figure 12. The rough set of $A = (U, C \cup D)$	20
Figure 13. The rough set of $A' = (U, C \cup D - \{a_2\})$	21
Figure 14. The GROW function.	23
Figure 15. The DISCPOW function.	23
Figure 16. The BUILD_CLASSIFIER function.....	26
Figure 17. The example for checking if two records are indiscernible.	26
Figure 18. The data structure of a rough set.	30
Figure 19. The data structure of the antecedent of a rule: RAntd	30
Figure 20. The data structure of a rule: RouserRule	31

Figure 21. A data structure of the rule set: m_Ruleset..... 31

Figure 22. The data structure of ROUSER’s rule model..... 31

Figure 23. The flow chart of **BuildClassifier()**..... 32

Figure 24. The flow chart of **OneClassRule()**..... 33

Figure 25. The flow chart of **grow()**..... 34



List of Tables

Table I. Information system $A=(U,C)$	10
Table II . Decision table $A=(U,C \cup D)$	11
Table III. Type 1 indiscernibility.....	27
Table IV. Type 2 indiscernibility.....	27
Table V. Type 3 indiscernibility.	27
Table VI. Type 4 indiscernibility.	27
Table VII. Original nominal data sets.....	37
Table VIII. Discretized data sets.....	37
Table IX. Details of discretization.....	38
Table X. Number of contradictions in original nominal data sets.....	38
Table XI. Number of contradictions in discretized data sets.....	38
Table XII. Results for original nominal data sets.....	40
Table XIII. Number of contradictions in artificial missing values in data sets.	40
Table XIV. Results for artificial missing values in data sets.	41
Table XV. Results for ordered rule strategy.....	42
Table XVI. Results of replacing Chi-Square value with Information Gain in ROUSER.	43
Table XVII. Results for discretized data sets.	44
Table XVIII. Training time.....	45
Table XIX. Rule set size.....	46
Table XX. The comparison of DiscPow_Chi and CfsSubsetEval.....	48
Table XXI. The comparison of DiscPow_Chi and Information Gain feature selection.....	49

CHAPTER 1

INTRODUCTION

1.1 Classification Problem

In machine learning, a classification task is to classify an unknown data record into a pre-specified category based on values of attributes of the data record. Before the machine is capable to do the classification task, it needs to learn from some training data where each data record has been associated with a category. Each attribute of a given data set corresponds to a domain of continuous values, i.e. real numbers, or a domain of discrete values, i.e. nominal data.

The proposed classification algorithm is specialized to nominal data. Nominal data is common in banking. Most of a customer's personal information, such as gender, marital status, hobbies, and hometown, are nominal data. Banks would like to have rules to utilize a customer's personal information in calculating his or her credit score. In addition, biologists are familiar with nominal data. Gene data is all nominal, and biologists want to study the relationships between gene combinations and a certain disease. Furthermore, although data from sensors is usually real numbers, engineers often need to discretize them into nominal data for further processing.

To monitor a machine and check if it is stable or not, for example, engineers may want to use data of real numbers from sensors in the machine to train a classification model, or a classifier, in which the underlying classification algorithm is based on complex mathematical

methods. Examples of such classification algorithms include Support Vector Machines (SVMs) [4] or Artificial Neural Networks (ANNs) [2]. Engineers may obtain high accuracy from the trained classification models but learn little from them. Engineers need to know the possible causes of a fault or a problem in order to perform fault diagnosis and resolve the problem, but they will have difficulty in identifying the possible causes from complex mathematical expressions given by SVMs or ANNs.

The goal of the classification algorithm in this thesis is to extract human understandable decision rules from nominal data. A decision rule is a function mapping a data space (a space of data records) to a class space (a space of categories or class labels). A human understandable decision rule is helpful for domain experts, such as engineers in the above example, to learn the causes and effects from data, and it is also important for scientists who intend to acquire knowledge from data.

1.2 Tree-Based and Rule-Based Classification Algorithms

Decision tree learning is one of the most widely used and practical methods for inductive inference [13], and a model learned by the method is a discrete-valued function, which can be represented by a decision tree. A classification algorithm which adapts decision tree learning method is called a tree-based classification algorithm.

Unlike the tree-based algorithms, the hypotheses learned by a rule-based classification algorithm are sets of if-then rules, which is the most expressive and human-understandable. One way to learn a set of rules is re-representing a learned decision tree by a set of mutual-exclusive rules, one rule for one path from the root to the leaf in the tree. Another widely used method for rule learning is separate-and-conquer (sequential covering). First-order rules are rules with variables, and they are more powerful in representation than

decision tree or propositional rules in some special cases. However in this thesis we focus on learning propositional rules.

Researchers have proposed several classification algorithms with the ability of rule generation. Rule-based classification algorithms like RIPPER (Repeated Incremental Pruning to Produce Error Reduction) [3] can generate rules directly, while tree-based classification algorithms like ID3 [17] and C4.5 [18] can also generate rules after transformation. C4.5 is one of the most popular classification algorithms [21], while RIPPER represents the state-of-the-art rule-based classification algorithms [10] [11]. What makes the classification algorithm proposed in this thesis different from the rule-based and tree-based classification algorithms is that it decides an attribute-value pair for the antecedents of a rule according to the rough set theory.

1.3 Rough Set Based Classification Algorithms

Rough set theory is a mathematical tool to describe imprecise knowledge. The most successful application of rough set is feature selection. Based on the features selected by rough set approach, researchers attempt to develop classification algorithms. Indiscernibility matrix [14] is widely adapted by these researchers to generate rules from data. In most classification algorithms that are based on the rough set theory, the indiscernibility matrix is used to generate all possible reducts (each of which is a subset of attributes) in nominal data and then generate rules from reducts. However, the computational cost of the indiscernibility matrix is high. There exist speed-up methods, but most of them are still based on the indiscernibility matrix [1][5]. The classification algorithm proposed in this thesis adopts the separate-and-conquer strategy [7] rather than the indiscernibility matrix, for rule generation.

1.4 Data Mining

Data mining is relatively a young field in computer science, and the goal is to capture knowledge from data in human-understandable structure for further use [24]. To achieve this goal, different disciplines like artificial intelligence, machine learning, statistics, and database systems are fused together.

In a practical application of data mining, some customizations are required to fit a client's need. Although the goal of data mining is to capture human-understandable knowledge, sometimes the discovered knowledge is still too hard for clients to understand. In order to bridge the gap between the client and engineers, we follow a cooperative data analysis method. Here we introduce how we implement data mining techniques in cooperative data analysis. In Figure 1, the User is the client or the person who has the need of data analysis, and the Miner is the one who analysis the data. When User hands the data to Miner and introduces the background, the cooperation has begun. First the Miner enters the stage of improvement, understanding the user's need, being familiar with the data, making clear the way to settle the problem, and seeking out the suitable algorithms. After that, the Miner enters the stage of model building. At this stage, Miner preprocesses the data, and builds models from the data. Followed by the stage of building rules, Miner summarizes results from the models and makes them understandable to user. The cycle of the cooperation now turns to the user's side, and enters the stage of results inspecting. The user must spend time to inspect the results and judge them by professional knowledge. We hope that no matter the results coming from miner is useful or not, User can make some feedback to Miner, since User is the one who is mostly sensitive to the case and is professional to the background knowledge. This is the stage of feedback. Receiving the feedback from the User, Miner can think and improve the analysis methods, and make the cooperative analysis a positive cycle.

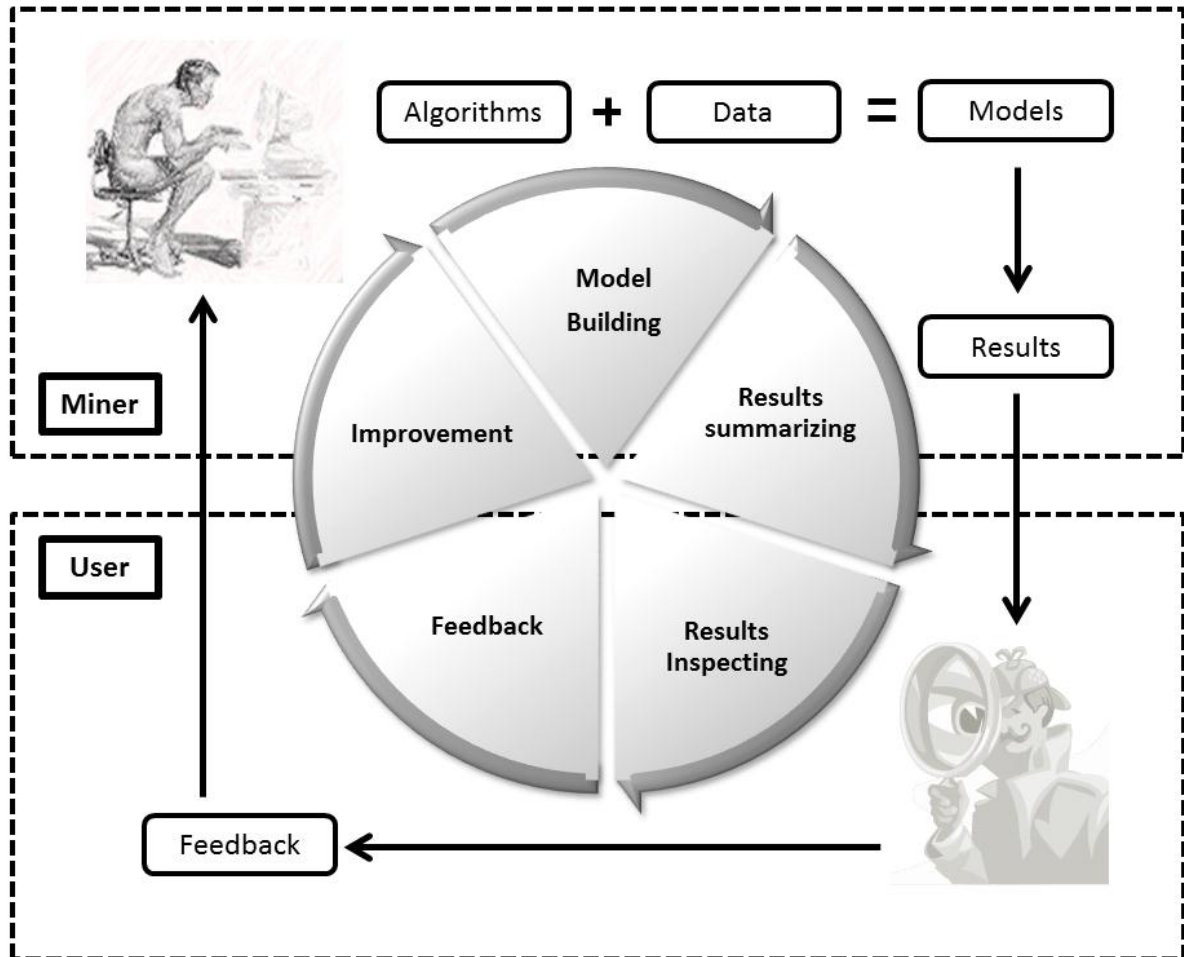


Figure 1. Cooperative data analysis.

1.5 Thesis Organization

The rest of this thesis is organized in the following way: Chapter 2 will give the preliminaries, and the proposed classification algorithm will be introduced in Chapter 3. Chapter 4 will be the implementation of the proposed algorithm. The experimental results are presented in Chapter 5. A case study is given in Chapter 6. The thesis will be concluded in Chapter 7 with potential directions for future work.

CHAPTER 2

PRELIMINARY

2.1 Rule-Based Classification Algorithms

2.1.1 The Basics

Rule induction is to learn rules from the given training data, and a rule-based classification algorithm uses the learned rules to classify unseen data records. For classification, a decision rule is a logic statement with the following form:

$$\text{condition1} \wedge \text{condition2} \wedge \dots \rightarrow \text{class}$$

where a condition is usually an attribute-value pair, indicating a certain value of certain attribute that is required to trigger the condition.

If a training data record matches all conditions of the rule, we say that the rule **covers** the data record; if the rule covers a data record and classify the data record to the right class, we say that the rule **explains** the data record. Given a rule set \mathbf{R} , for every possible data record, if there exists a rule which is able to cover the record, we say that the set of rules are **exhaustive**. If no two rules in \mathbf{R} cover the same data record, we say that the rule set is **mutually exclusive**. If the rule set is not mutually exclusive, a data record can be covered by several rules and lead to contradicting results. Generally there are two approaches to overcome this problem: Ordered rules and unordered rules. Ordered rules rank the rules by a certain criteria (e.g. accuracy, coverage, description length), so only one rule will be chosen to classify a data record. Unordered rules allow multiple rules to be triggered to classify a single

data record through voting or weighting methods.

RIPPER [3] is a popular rule-based classification algorithm. It has two stages: The generation stage and the optimization stage. The classification algorithm proposed in this thesis competes with it in the generation stage.

2.1.2 Separate-and-Conquer

The separate-and-conquer strategy, or sequential covering, first builds a rule that explains a part of the training data, separates them, and conquers the rest recursively until no data remains. It ensures that every data record is at least covered by one rule. Figure 2 gives the separate-and-conquer algorithm, the core of the proposed classification algorithm in this thesis. Before the algorithm begins, one of the classes is chosen. POSITIVE chooses the data that should be classified to the chosen class, and NEGATIVE chooses the others. Every rule is empty in the beginning, and continues to grow until no negative data is covered by it.

```
Class = CHOOSE(ClassSet)
SEPERATE&CONQUER(Class,TrainData):
  RuleSet =  $\emptyset$ 
  while POSITIVE(TrainData) $\neq\emptyset$ 
    Rule=[null $\rightarrow$ Class]
    Covered=COVER(TrainData,Rule)
    while NEGATIVE(Covered) $\neq\emptyset$ 
      GROW(Rule,Covered)
      Covered= COVER(Covered ,Rule)
    RuleSet=RuleSet  $\cup$  {Rule}
    TrainData=TrainData \ Covered
  return RuleSet
```

Figure 2. The **SEPERATE&CONQUER** algorithm.

2.1.3 Search Heuristics

Search heuristics are used to evaluate the found hypotheses. The GROW function in the

separate-and-conquer algorithm given in Figure 2 searches from the covered data a suitable attribute and the corresponding value in order to grow a rule. Examples of search heuristics include Entropy and used in ID3 [17] and C4.5 [18].

Entropy

Entropy is the weighted average of information content of each class and originates from the ID3 decision tree learning system [7]. Given a set S , the Entropy of the set S is defined as:

$$E(S) = - \sum_{j=1}^N Pr(j) \log_2 Pr(j)$$

where N is the number of different values of an attribute in S , and $Pr(j)$ is the proportion of the value j in the set S .

The definition of Entropy above is suitable for decision trees. To be suitable for a rule-based classification algorithm, the Entropy can be defined as:

$$E(S) = - \frac{p}{p+n} \log_2 \frac{p}{p+n} - \frac{n}{p+n} \log_2 \frac{n}{p+n}$$

where p is the number of positive instances covered by a given rule r , and n is the number of positive instances covered by the given rule r . It is obvious that this definition is a special binary case of the original definition.

Information Gain

Information Gain measures the expected reduction in Entropy caused by partitioning the instances according to an attribute [13]. The definition of Information Gain is:

$$IG(S, a) = E(S) - \sum_{v \in Values(a)} \frac{|S_v|}{|S|} E(S_v)$$

where a is the attribute, and S_v is the subset of S for which attribute a has value v .

2.1.4 Pruning and Optimization

We believe that a generated rule might be overfitting, which means that a rule is grown too precisely to achieve high accuracy, while few data records are explained by this strict rule. To avoid overfitting, pruning methods were introduced to shorten the rule. In general there are two categories of pruning methods: **Pre-pruning** and **post-pruning**. Pre-pruning methods stop the growing of the rule by implementing some stopping criteria, such as Purity, Minimum Description Length, significance, etc. Post-pruning methods drop part of the conditions from a grown rule by testing if the pruned rule performs better than the original rule on some criteria or not. Currently the proposed ROUSER adapts no pruning methods, while implementing a pruning method suitable for ROUSER will be part of the future work.

Rules generated through pruning stage are usually perform well, and experiments show that the whole rule sets are significantly improved on both the size and the performance through global **optimization**, which is a post-induction optimization method on the whole rule set. Currently ROUSER adapts no optimization methods, while investigating an optimization method suitable for ROUSER will be part of future work.

2.2 The Rough Set Theory

The rough set theory is first introduced by Zdzisław I. Pawlak in 1982 as a mathematical tool to characterize imprecise knowledge [15][16]. The main difference between a rough set and a classic set is the appearance of a boundary “region” (not just a boundary), where the uncertain elements exist, in a rough set. The fuzzy set theory [22] is another tool to characterize imprecise knowledge. The main difference between a fuzzy set and a rough set is that a fuzzy set needs a predefined function to decide the “membership degree” of each element. . Practically speaking, such a membership function is defined under some assumptions and on a

case-by-case basis. Nevertheless, a rough set needs no membership function, since the uncertain elements are located in the boundary region in a rough set.

2.2.1 Information System and Decision Table

An information system A is a pair, denoted by $A = (U, C)$, where U is the universe, and C is the set of attributes. When we deal with classification or clustering issues, the elements of U can be considered as instances. For each attribute $a \in C$, the value set is V_a . For each instance $x \in U$, it contains $|C|$ attribute values, and the value of attribute a in instance x is denoted by $a(x)$. The information system $A = (U, C)$ in Table I below, $U = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8\}$, $C = \{a_1, a_2\}$, $V_{a_1} = \{1, 2, 3, 4\}$, $V_{a_2} = \{1, 2, 3, 4\}$.

Table I. Information system $A=(U,C)$

U	a_1	a_2
x_1	1	2
x_2	2	1
x_3	2	2
x_4	3	2
x_5	3	2
x_6	3	3
x_7	3	4
x_8	4	3

A Decision Table [S15] is a special case of an information system with the form $A = (U, C \cup D)$, where $d \in D$ is a decision attribute, called *decision*, and $d \notin C$, while each $a \in C$ is called *condition*. The value set of d is V_d . d is also the *class* in a classification problem. The value of decision d in instance x is denoted by $d(x)$. For example, the decision table $A = (U, C \cup D)$ in Table II below, $U = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8\}$, $C = \{a_1, a_2\}$,

$D = \{d\}$, $V_{a_1} = \{1, 2, 3, 4\}$, $V_{a_2} = \{1, 2, 3, 4\}$ and $d = \{y, n\}$, $d(x_1) = y$, $d(x_5) = n$.

Table II . Decision table $A=(U, C \cup D)$

U	a_1	a_2	d
x_1	1	2	y
x_2	2	1	y
x_3	2	2	y
x_4	3	2	y
x_5	3	2	n
x_6	3	3	n
x_7	3	4	n
x_8	4	3	n

2.2.2 Indiscernibility Relation

Indiscernibility relation is an equivalence relation mathematically, but the meaning is different. When we say that two objects are indiscernible, we mean that the two objects have exact the same value on every attribute and hence we cannot distinguish the two objects. However, we still cannot say that the two objects are the same, due to the limit of knowledge (attributes). A formal definition of indiscernibility relation is given below.

For every instance $x, y \in U$, x, y are indecernable if and only if for every $a \in C$, $a(x) = a(y)$. For each subset $C' \subseteq C$, C' makes a partition on U , denoted by U/C' , and $C'(x) \in U/C'$ denotes the block of the partition containing instance x , which means $x \in C'(x)$. For each $y \in C'(x)$, $a(y) = a(x)$, which means that instances in the same block of partition are indiscernible. C' forms an indiscernibility relation and $I(C')$ defines as follows:

$$x I(C') y \text{ if and only if } a(x) = a(y) \text{ for every } a \in C'.$$

For example, consider the decision table in Table II above. All partitions are given below:

$$U/D = \{\{x_1, x_2, x_3, x_4\}, \{x_5, x_6, x_7, x_8\}\},$$

$$U/C = \{\{x_1\}, \{x_2\}, \{x_3\}, \{x_4, x_5\}, \{x_6\}, \{x_7\}, \{x_8\}\},$$

$$U/\{a_1\} = \{\{x_1\}, \{x_2, x_3\}, \{x_4, x_5, x_6, x_7\}, \{x_8\}\},$$

$$U/\{a_2\} = \{\{x_2\}, \{x_1, x_3, x_4, x_5\}, \{x_6, x_8\}, \{x_7\}\}.$$

2.2.3 Rough Set

The main difference between a rough set and a classic set is the appearance of a boundary “region” (not just a boundary), as shown in Figure 3 (a), (b). Given a decision table $A=(U, C \cup D)$, as shown in Table II, where $U=\{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8\}$ is the universe or the training data, $C=\{a_1, a_2\}$ is the condition or the attribute set of the training data, and $D=\{d\}$ is the decision or the set of class labels of the training data. A rough set of $d=y$ is shown in Figure 3 (c). Since there is no difference between the condition of x_4 and that of x_5 , they are in the boundary region.

The visualized rough set of $A = (U, C \cup D)$ is shown in Figure 3 (c).

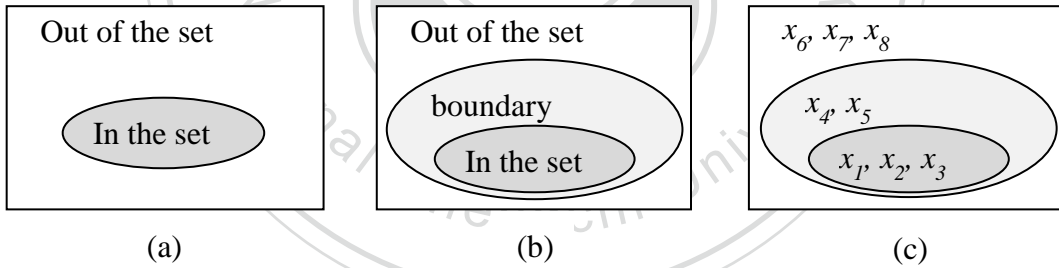


Figure 3. (a) Classic set. (b) Rough set. (c) Rough set for example.

We give an example to help understand a rough set. The set Y corresponding to the set of $d = y$ is $\{x_1, x_2, x_3, x_4\}$, as shown in Figure 4 (a), where the set is mapped to a 4x4 data space of $C = \{a_1, a_2\}$. If we want to define Y precisely through C , we find that elements x_4 and x_5 are indiscernible on C , or $x_4 I(C) x_5$, since both of them satisfy $a_1 = 3$ and $a_2 = 2$, and hence Y cannot be defined precisely through the known attributes. It is easy to see that x_1, x_2, x_3 are certain to belong to Y . We are not sure if x_4, x_5 belong to Y or not, but we are sure

that x_6, x_7, x_8 do not belong to Y . Hence we can characterize the set Y by two crisp set, $\{x_1, x_2, x_3\}$ and $\{x_1, x_2, x_3, x_4, x_5\}$, the lower-approximation and upper-approximation of Y , respectively, as shown in Figure 4 (b) and (c). This example gives a sense to a rough set: A rough set is actually a combination of several traditional sets (crisp sets).

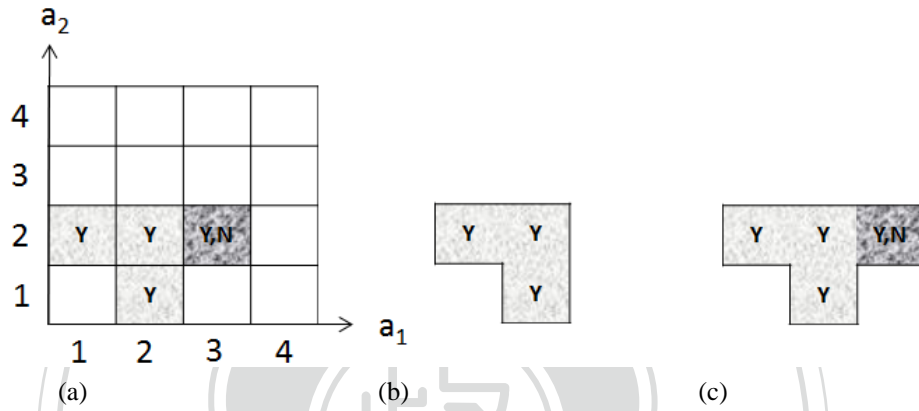


Figure 4. (a) The space of $d=y$. (b) The lower-approximation of $d=y$. (c) The upper-approximation of $d=y$.

Here we give a formal definition to a rough set. Consider a decision table $\mathbf{A} = (U, C \cup D)$, where D forms a partition U/D and indiscernibility relation $I(D)$. For each subset $C' \subseteq C$, C' forms a partition U/C' and indiscernibility relation $I(C')$. When dealing with a classification problem, $I(D)$ must be approximated by $I(C')$. For each block of partition $X \in U/D$, the C' -lower approximation of X is as follows:

$$\underline{C'}(X) = \{x \in U : C'(x) \subseteq X\}$$

The C' -upper approximation of X is as follows:

$$\overline{C'}(X) = \{x \in U : C'(x) \cap X \neq \emptyset\}$$

If $\underline{C'}(X) = \overline{C'}(X)$, we say that X is C' -definable. The rough set theory defines the set X by both $\underline{C'}(X)$ and $\overline{C'}(X)$. If X is C' -definable, we say X is crisp, otherwise X is rough.

The **positive region** of the partition U/D with respect to C' is expressed as $POS_{C'}(D)$, which is a union of every block's lower-approximation of the partition U/D . The definition is given below:

$$POS_{C'}(D) = \bigcup_{X \in U/D} \underline{C'}(X)$$

There are no contradicting data records in $POS_{C'}(D)$. An example of a positive region is given in Figure 5.

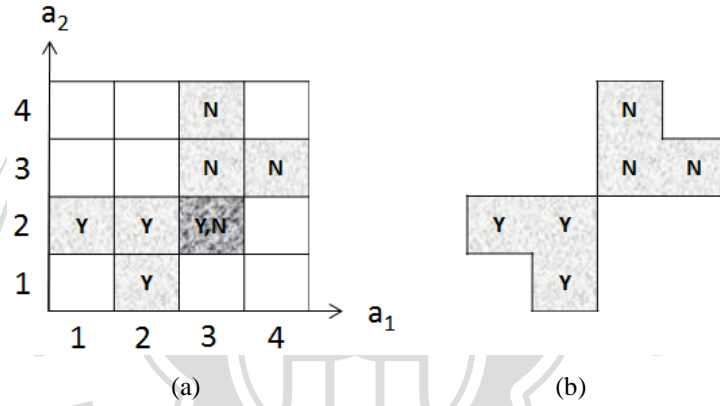


Figure 5. (a) The data space. (b) The positive region of U/D .

The dependency degree of D respect to C' is defined below:

$$\gamma_{C'}(D) = \frac{card(POS_{C'}(D))}{card(U)}$$

If $\gamma_{C'}(D) = 1$ we said that A is consistent on C' , which means that there are no contradicting data records.

2.2.4 Reduct and Core

Given a decision table $A = (U, C \cup D)$, an attribute $a \in C$ is said to be *dispensable* if $\gamma_{C-\{a\}}(D) = \gamma_C(D)$. A subset $C' \subseteq C$ is a **reduct** of C with respect to D if no attribute $a \in C'$ is dispensable. There can be more than one reduct of C , and the set of reducts is denoted by $Red_C(D)$. The core of C with respect to D is defined as below:

$$Core_C(D) = \bigcap_{R \in Red_C(D)} R$$

Consider the new example in Figure 6, where a new attribute a_3 is given, and two partitions are shown as follow:

$$U/\{a_1, a_3\} = \{\{x_1\}, \{x_2, x_3\}, \{x_4\}, \{x_5\}, \{x_6, x_7\}, \{x_8\}\}$$

$$U/\{a_2, a_3\} = \{\{x_2\}, \{x_1, x_3\}, \{x_4\}, \{x_5\}, \{x_6, x_8\}, \{x_7\}\}$$

It is easy to understand that both $\{a_1, a_3\}$ and $\{a_2, a_3\}$ are reducts of the new decision table, and $\{a_3\} = \{a_1, a_3\} \cap \{a_2, a_3\}$ is the core. Graphs for visualization are given in Figure 7 and Figure 8.

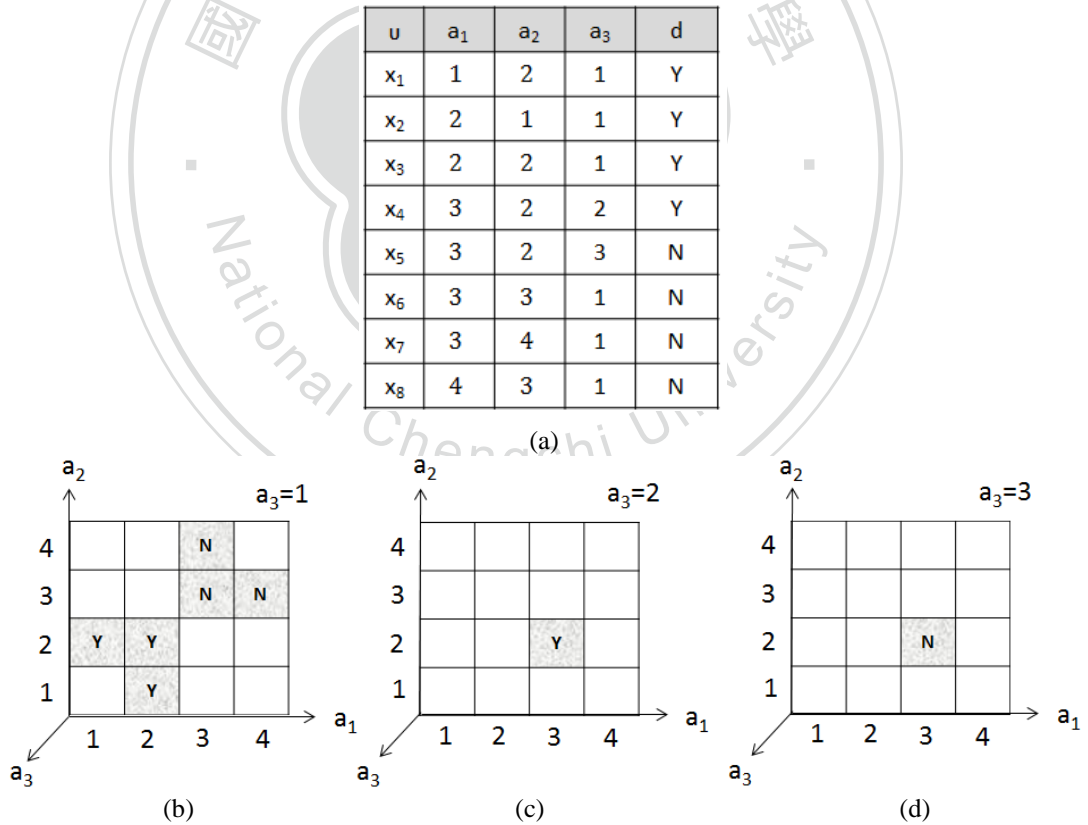


Figure 6. (a) The new decision table with a_3 . (b) Data space of $a_3=1$. (c) Data space of $a_3=2$. (d) Data space of $a_3=3$.

u	a ₁	a ₃	d
x ₁	1	1	Y
x ₂	2	1	Y
x ₃	2	1	Y
x ₄	3	2	Y
x ₅	3	3	N
x ₆	3	1	N
x ₇	3	1	N
x ₈	4	1	N

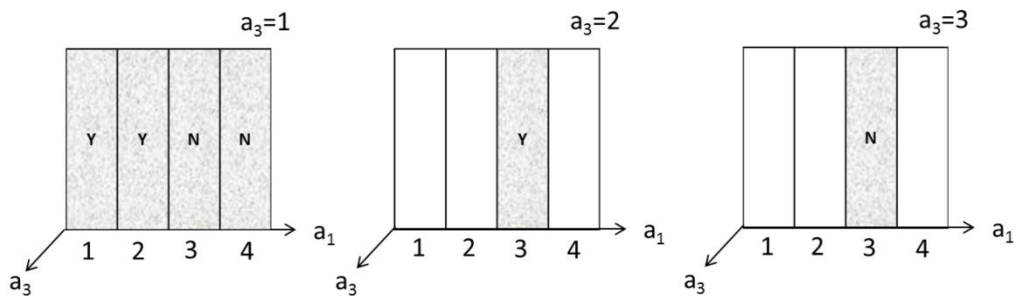


Figure 7. a_1, a_3 as the reduct.

u	a ₂	a ₃	d
x ₁	2	1	Y
x ₂	1	1	Y
x ₃	2	1	Y
x ₄	2	2	Y
x ₅	2	3	N
x ₆	3	1	N
x ₇	4	1	N
x ₈	3	1	N

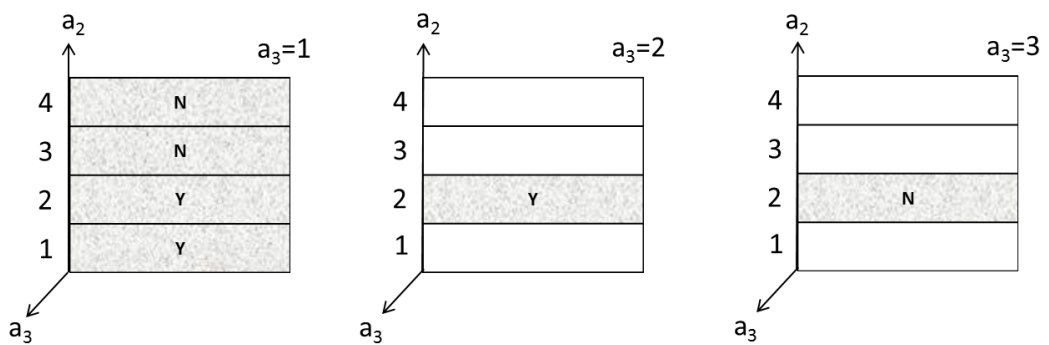


Figure 8. a_2, a_3 as the reduct.

2.2.5 Indiscernibility Matrix

Given a decision table $A = (U, C \cup D)$, a discernibility matrix $M_D(C)$ of A is a $n \times n$ matrix, and the entry of the matrix is defined as follows:

$$c_{ij} = \{a \in C : a(x_i) \neq a(x_j) \wedge d(x_i) \neq d(x_j)\} \text{ for } i, j = 1, 2, \dots, n.$$

where n is the number of elements in U and $x_i, x_j \in U$.

Discernibility function $f_D(A)$ is defined as follows:

$$f_D(C) = \bigwedge \{ \bigvee a : a \in c_{ij}, 1 \leq i < j \leq n, c_{ij} \neq \emptyset \}$$

A discernibility function $f_D(A)$ is a boolean function, all constituents in the disjunctive normal form of $f_D(C)$ are all D -reducts of C , and all prime implicants of the conjunctive normal form of $f_D(C)$ are also all D -reducts of C .

An indiscernibility matrix of decision table in Figure 6 (a) is given in Figure 9 below.

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
x_1								
x_2								
x_3								
x_4								
x_5	a_1, a_3	a_1, a_2, a_3	a_1, a_3	a_3				
x_6	a_1, a_2, a_3	a_1, a_2, a_3	a_1, a_2	a_2, a_3				
x_7	a_1, a_2	a_1, a_2	a_1, a_2	a_2, a_3				
x_8	a_1, a_2	a_1, a_2	a_1, a_2	a_1, a_2, a_3				

Figure 9. An example of indiscernibility matrix.

CHAPTER 3

DESIGN OF THE PROPOSED METHOD

3.1 Potential Boundary Region and Discernibility Power

One of the contributions of this thesis is presenting a new search heuristics named **discernibility power** based on the rough set theory. Before introducing discernibility power, we have to redefine the rough set for disambiguation and convenience.

Redefining a Rough Set

Guided by the original definition of rough set theory, we redefine a rough set. Given a decision table $A = (U, C \cup D)$, for each block X of partition U/D , the rough set of X is redefined below:

The positive region of X :

$$POS_A(X) = \{x \mid C(x) \subseteq X\}.$$

The negative region of X :

$$NEG_A(X) = \{x \mid C(x) \cap X = \emptyset\}.$$

The boundary region of X :

$$BOUND_A(X) = X - POS_A(X) - NEG_A(X).$$

Notice that the positive region here is the same as the definition of the lower-approximation of a rough set, but it differs from the one mentioned in 2.2.3, which is the positive region of D respect to C . As sketched in Figure 10, a rough set is redefined by 3 disjunctive traditional sets, positive region, negative region and boundary region. **The redefined rough set is also a**

partition of U . The purpose of the redefinition is to connect the rough set theory with the separate-and-conquer algorithm, which iteratively grows a rule by rejecting as many negative data records as possible and accepting as many positive data records as possible. Based on the redefinition of a rough set, we introduce two concepts: Potential boundary region (*PotBound*) and discernibility power (*DiscPow*).

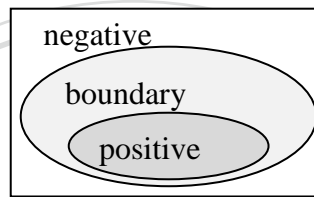


Figure 10. The redefined rough set.

Potential Boundary Region

Consider the rough set of X defined above, the meaning of the *potential boundary region* of attribute a_i is the set of elements which will become indiscernible without a_i . The definition of *PotBound* of X with respect to attribute a_i is given below:

$$PotBound_A(X, a_i) = Bound_{A'}(X) - Bound_A(X),$$

where $A = (U, C \cup D)$ and $A' = (U, C \cup D - \{a_i\})$.

Here is an example of *PotBound*. Consider *sdfsff*, the original decision table is $A = (U, C \cup D)$, if the attribute a_2 is removed, the new decision table becomes $A' = (U, C \cup D - \{a_i\})$. x_6, x_7 become indiscernible, and the boundary region of Y expands. The expanded part of the boundary region $\{x_6, x_7\}$ is the *PotBound* of a_2 .

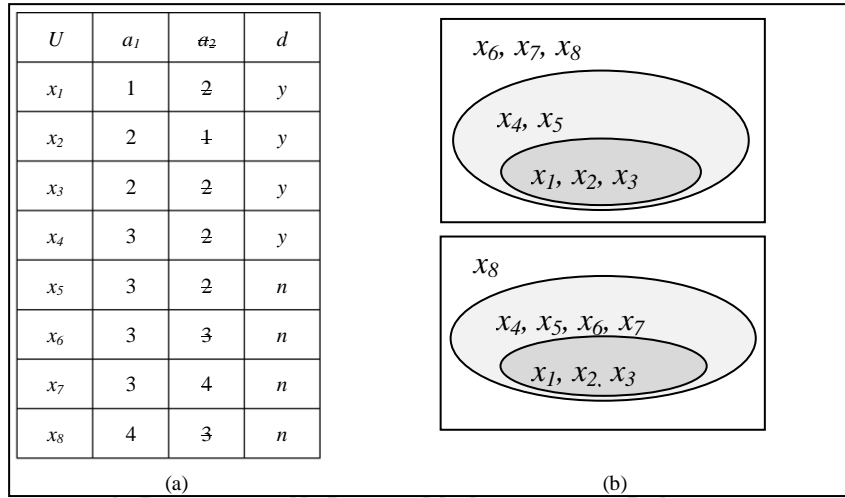


Figure 11. (a) Decision table $A' = (U, C \cup D - \{a_2\})$. (b) The new rough set of $d=y$.

Discernibility Power

The meaning of *DiscPow* of attribute a_i is how many elements will become indiscernible without a_i . The definition of *DiscPow* of a_i with respect to the X is given below:

$$DiscPow_A(X, a_i) = Card(PotBound_A(X, a_i)).$$

Reuse the example above, the *DiscPow* of a_1 with respect to Y is 2, or $DiscPow_A(Y, a_1) = 2$.

DiscPow has the **monotonicity** property, which means that removing elements from a rough set, or a partition of U , will never increase the *DiscPow*. Below is the proof.

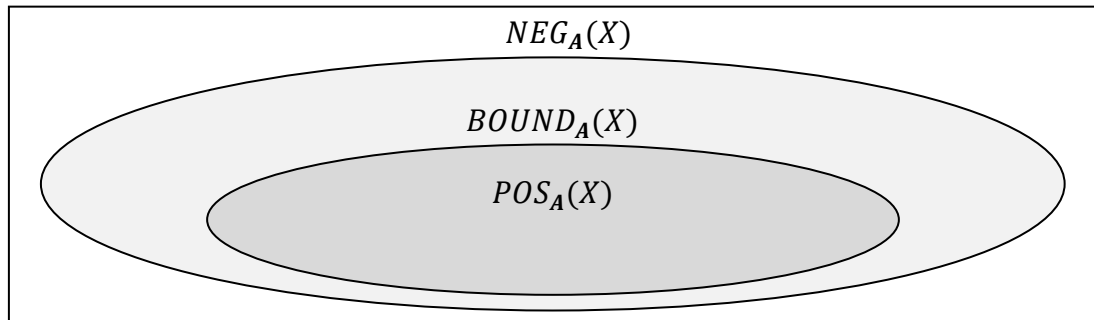


Figure 12. The rough set of $A = (U, C \cup D)$.

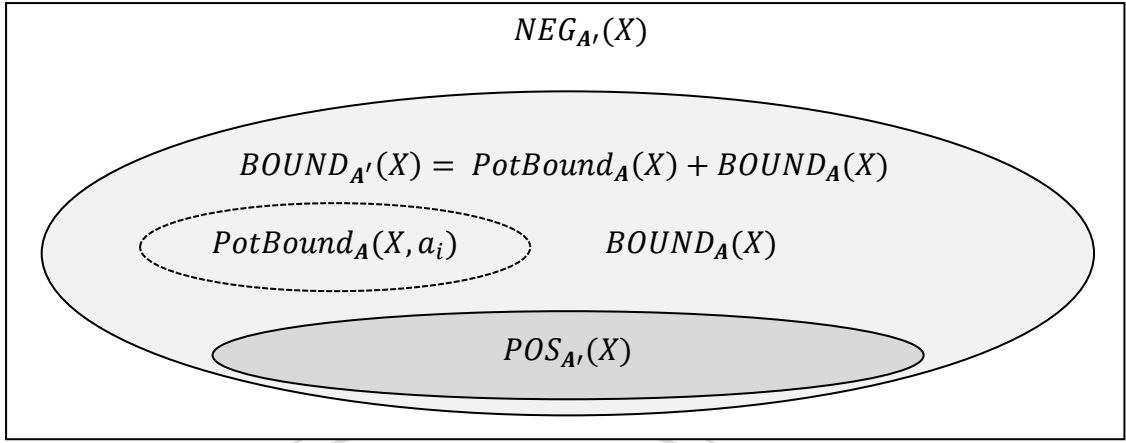


Figure 13. The rough set of $A' = (U, C \cup D - \{a_2\})$

Given a decision table $A = (U, C \cup D)$ as shown in Figure 12, the DiscPow of $a_i \in C$ with respect to the rough set of $X \in U/D$ is as below:

$$DiscPow_A(X, a_i) = Card(PotBound_A(X, a_i)),$$

and the $PotBound_A(X, a_i)$ is given below:

$$PotBound_A(X, a_i) = Bound_{A'}(X) - Bound_A(X),$$

where $A' = (U, C \cup D - \{a_i\})$, as shown in Figure 13.

Below are definitions for X with respect to A' :

The positive region of X :

$$POS_{A'}(X) = \{x \mid C(x) \subseteq X\}$$

The negative region of X :

$$NEG_{A'}(X) = \{x \mid C(x) \cap X = \emptyset\}$$

The boundary region of X :

$$BOUND_{A'}(X) = X - POS_{A'}(X) - NEG_{A'}(X)$$

By the $PotBound_A(X, a_i)$ given above, the boundary region of X has another definition:

$$BOUND_{A'}(X) = PotBound_A(X, a_i) + BOUND_A(X),$$

hence the rough set of X is equal to $\{POS_{A'}(X), NEG_{A'}(X), PotBound_A(X, a_i), BOUND_A(X)\}$, which is a partition of U . This indicates that any element e removed from U originally belongs to exactly one block of this partition. Since $DiscPow_A(X, a_i) = Card(PotBound_A(X, a_i))$, the only way to modify the value of $DiscPow_A(X, a_i)$ is inserting or removing element from $PotBound_A(X, a_i)$. It is obvious that removing an element from $PotBound_A(X, a_i)$ will cause the $DiscPow_A(X, a_i)$ to drop, and inserting an element into $PotBound_A(X, a_i)$ will cause the $DiscPow_A(X, a_i)$ to rise. Removing more than one element from U can be considered as iterally removing an element, and inserting more than one element from U can be considered as iterally inserting an element. By all above, removing elements from U will cause the $DiscPow_A(X, a_i)$ to either hold or drop, and inserting elements to U will cause the $DiscPow_A(X, a_i)$ to either hold or rise, and this is the monotonicity property of DiscPow.

Discernibility Power is one of the search heuristics of the proposed rule-based algorithm: ROUSER, which will be introduced in the next subsection.

3.2 ROUSER

ROUSER follows the separate-and-conquer algorithm as the framework. Our contribution here is connecting the proposed DiscPow as the search heuristic used by the GROW function in the separate-and-conquer algorithm. The GROW function of ROUSER is shown in Figure 14. ROUSER removes attributes whose values of $DiscPow$ are zero in each iteration, and it updates $DiscPow$ of every attribute until all values of $DiscPow$ of the remaining attributes are not zero. If multiple attributes need to be removed, the current version of ROUSER simply removes the one that is independent of the class entered as a parameter to the separate-and-conquer algorithm in Figure 2. We use Chi-Squared value to decide the

degree of independence. Chi-Squared value was first used in feature selection in [9]. Feature selection with Chi-Square test together with rough set theory was proposed in [19].

```

GROW(Rule, Covered):
  do:
    for every attribute  $a_i$ :
       $DiscPow_i = DISCPOW(a_i, Covered)$ 
       $ChiSquared_i = CHISQUARED(a_i, Covered)$ 
      Among attributes with  $DiscPow_i = 0$ , ignore  $a_i$  with
      minimum  $ChiSquared_i$ 
    while exist  $a_i$  with  $DiscPow_i = 0$ 
       $(a, v) = CHOOSE\_ATTR\&VALUE()$ 
      grow the rule with  $(a, v)$  as an antecedent
  
```

Figure 14. The **GROW** function.

Once an attribute is removed in an iteration when the GROW function is running, we no longer need to compute its DiscPow value anymore because of the monotonicity property of DiscPow. When elements are removed from the rough set covered by current rule, the DiscPow value of an attribute will be the same or a smaller value. Once the DiscPow value of the attribute is zero, it will no longer increase and hence the attribute can be removed. The DISCPOW function is shown in Figure 15.

```

DISCPOW( $a_i, Covered$ ):
  decision table  $A = (U, C \cup D)$ 
  let  $C'$  be  $C - \{a_i\}$ 
  for every elements  $x_i$  and  $x_j, i < j$ 
    if  $C'(x_i) = C'(x_j) \wedge D(x_i) \neq D(x_j)$ 
       $PotBound(a_i) = PotBound(a_i) \cup \{x_i, x_j\}$ 
  return the cardinality of  $PotBound(a_i)$ 
  
```

Figure 15. The **DISCPOW** function.

The CHOOSE_ATTR&VALUE function in GROW function searches for an attribute-value pair, i.e. (a_i, v_i) , that will be used to grow a rule. We use the idea of purity value [9][20] as the search heuristics. In our algorithm we provide 3 types of purities as options: PurityOverAll, PurityPotBound, and PurityHybrid. The first is the same as the original definition of purity, and the others are proposed by us. The definitions of these purities are given below:

$$\text{PurityOverAll} = |p_{\text{all}}| / (|p_{\text{all}}| + |n_{\text{all}}|),$$

where p_{all} is the positive records covered by the candidate attribute and value, and n_{all} is the negative records covered by the candidate attribute and value;

$$\text{PurityPotBound} = |p_{\text{pb}}| / (|p_{\text{pb}}| + |n_{\text{pb}}|),$$

where p_{pb} is the positive records in the potential boundary region of the candidate attribute, and p_{pb} is covered by the candidate attribute and value, and n_{pb} is the negative records in the potential boundary region of the candidate attribute, and n_{pb} is covered by the candidate attribute and value;

$$\text{PurityHybrid} = |p_{\text{pb}}| / (|p_{\text{pb}}| + |n_{\text{all}}|),$$

where p_{pb} is the positive records in the potential boundary region of the candidate attribute, and p_{pb} is covered by the candidate attribute and value; n_{all} is the negative records covered by the candidate attribute and value.

In addition to purity, we provide weighted Information Gain as an option for search heuristic, which is defined as:

$$\text{WInfoGain} = (p2_{\text{all}}/p1_{\text{all}}) * (\log(|p2_{\text{all}}| / (|p2_{\text{all}}| + |n2_{\text{all}}|)) - \log(|p1_{\text{all}}| / (|p1_{\text{all}}| + |n1_{\text{all}}|)))$$

where $p1_{\text{all}}$ and $n1_{\text{all}}$ is the positive and negative records respectively from the original set of data records, and $p2_{\text{all}}$ and $n2_{\text{all}}$ is the positive and negative records respectively from the chosen subset of data records. The “ $\log(|p1_{\text{all}}| / (|p1_{\text{all}}| + |n1_{\text{all}}|))$ ” is the information content of the

original set of data records, while $\log(|p2_{all}|/(|p2_{all}|+|n2_{all}|))$ is the information content of the chosen subset. $(p2_{all}/p1_{all})$ is the weight of the Information Gain.

We also provide 2 methods, and the first is called “Max”, which finds the maximum (i.e. purity) from all possible attribute-value pairs. The second is called “Frequent Max”, which finds the most frequent value in each attribute and then finds the maximum (i.e. purity) from them.

At last, our CHOOSE_ATTR&VALUE function can choose an attribute-value pair in 7 different ways:

1. PurityOverAll, Max
2. PurityPotBound, Max
3. PurityHybrid, Max
4. PurityOverAll, Frequent Max
5. PurityPotBound, Frequent Max
6. PurityHybrid, Frequent Max
7. WInfoGain, Max

ROUSER generates a set of rules for each class. As soon as a rule set is generated, it is concatenated to the bottom of the rule list. The BUILD_CLASSIFIER algorithm of ROUSER is shown in Figure 16. The class list is sorted by ascending frequency order as RIPPER does. For an unseen case, ROUSER searches down the rule list and uses the first rule that covers the case to classify it.

```

BUILD_CLASSIFIER( ):
    build a ClassList by ascending frequency order
    for each Class in ClassList:
        RuleSet=SEPERATE&CONQUER(Class,TrainData)
        concatenate the RuleSet to the bottom of the RuleList
    return RuleList

```

Figure 16. The BUILD_CLASSIFIER function.

ROUSER has to decide if two records are indiscernible to determine the boundary and potential boundary regions. Consider the examples in Figure 17, where there are two records j and k . If we want to know if record j and record k are indiscernible, we have to check every attribute's value. If each attribute has the same value in record j and k , we say that the two records are indiscernible.

	attributes ...	a_i	attributes ...
record j	...	v_j	...
record k	...	v_k	...

Figure 17. The example for checking if two records are indiscernible.

It is a simple task to decide if two records are indiscernible or not. However, missing values make the task complicated. We define four types of indiscernibility between two values, as shown in Table III, Table IV, Table V, and Table VI. These tables show how we treat a missing value for an attribute when we try to check if two records are indiscernible. From type 1 to type 4, the determination of indiscernibility becomes stricter. Currently, ROUSER uses type 3 to find boundary region, and it uses type 1 to find potential boundary region. Part of our study in the future is to consider other types of indiscernibility.

Table III. Type 1 indiscernibility.

type 1		v_j	
		missing	α
v_k	missing	same	same
	α	same	same
	β	same	diff

Table IV. Type 2 indiscernibility.

type 2		v_j	
		missing	α
v_k	missing	diff	same
	α	same	same
	β	same	diff

Table V. Type 3 indiscernibility.

type 3		v_j	
		missing	α
v_k	missing	same	diff
	α	diff	same
	β	diff	diff

Table VI. Type 4 indiscernibility.

type 4		v_j	
		missing	α
v_k	missing	diff	diff
	α	diff	same
	β	diff	diff

Records with same conditions and different decisions are considered as contradictions. Based on the four types of indiscernibility, there will be four types of contradictions. ROUSER simply ignores the contradictions (type 3) in the training data.

CHAPTER 4

IMPLEMENTATION OF THE PROPOSED METHOD

4.1 WEKA

Weka[8] is an open source data mining software, which provides free Java code for machine learning task. Weka is developed by and updated by the University of Waikato in New Zealand. We use Weka 3.6.5 as our developing environment.

4.1.1 Import Data

Weka accepts several data formats, including the simplest format named Comma-Separated Values (CSV), and Attribute Relationship File Format (ARFF). After data is imported, it is stored by the Weka-defined data structures. Each data record is stored by an **Instance** object, and the whole data set is stored by an **Instances** object, which contains multiple **Instance** objects. An **Attribute** object contains all the details about an attribute, like the data type is nominal or real number, and how many values are in the attribute. Multiple **Attribute** objects are also contained in one **Instances** object.

4.1.2 Classifier

To develop a classifier under Weka's environment, an abstract class `weka.classifiers.Classifier()` must be extended. After that, an abstract method `buildClassifier()` must be implemented, and this method is called every time when the

classifier is invoked. This method builds up the classification model by learning from the training data. After the model is built, one of the two methods is called for classifying testing data: **classifyInstance()** and **distributionForInstance()**, which utilize the model built by **buildClassifier()** to generate the classification result for every single data record. The difference between these two functions is that, the former one returns exact one class label for prediction, while the latter one returns an array of probabilities with respect to class labels.

4.1.3 Cross-Validation

Weka offers several evaluation methods, and they are easy to implement. Here we introduce how to realize a cross-validation method. First an evaluator must be built by invoking **weka.classifiers.Evaluation()**, and then we choose the provided method **crossValidateModel()**.

4.2 Data Structure

The data structure used in the implementation of ROUSER is partially learned from the JRip provided by Weka.

4.2.1 Rough Set

In order to implement rough set intuitively, a data structure for rough set is built, as in Figure 18. A data set is split as a partition of 3 blocks, namely positive, boundary and negative, with respect to the definition of a rough set in Section 3.4, and each block is actually an Instances object as mentioned in Section 4.1.1. For the convenience, the blocks filled by black color is empty, while white is not empty. Some necessary information is stored in the structure, such as DiscPow, Chi-square value, Purity For several further use, such as choosing the best attribute and value to build a rule. This data structure never appears in Weka.

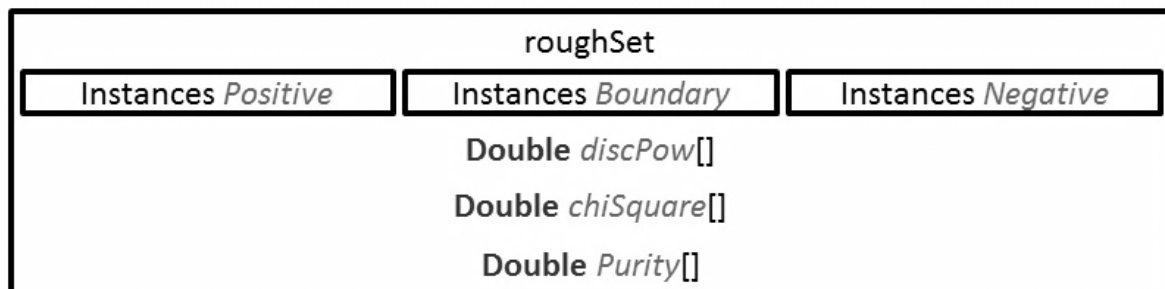


Figure 18. The data structure of a rough set.

4.2.2 Decision Rule

As mentioned in Section 2.1.1, a decision rule is a logic statement with the following form:

$$condition1 \wedge condition2 \wedge \dots \rightarrow class,$$

hence there can be multiple antecedents. We define a data structure named **RAntd** to store each condition, and some necessary information is contained in the structure,, such as the DiscPow, the number of instances covered by the rule so far (from the 1st condition to this condition), and the number of instances explained by the rule so far, as shown in Figure 19. This data structure is learned from JRip provided by Weka, however some of the information stored in it are different.

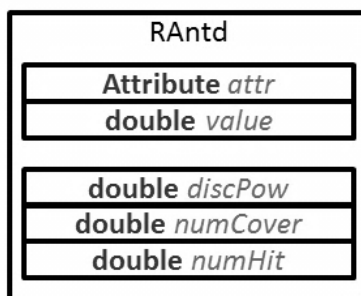


Figure 19. The data structure of the antecedent of a rule: RAntd.

Another structure learned from JRip provided by Weka is **RouserRule**, which stores a rule, as in Figure 20, and it contains two parts: The queue of the antecedents and a class label.

When a rule is grown, **RAntd** is generated one after another, and they are stored in a queue in order.

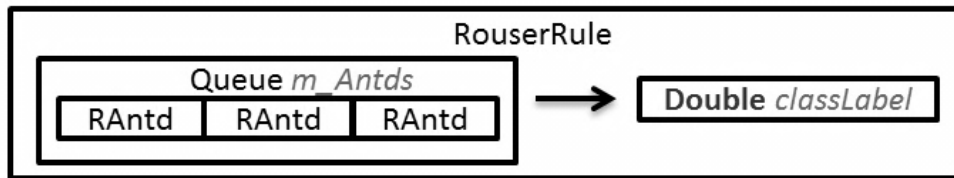


Figure 20. The data structure of a rule: RouserRule.

After a rule is generated, it is stored in the rule set in the growing order, as shown in Figure 21. The rule set is a queue. This is also learned from JRip provided by Weka.

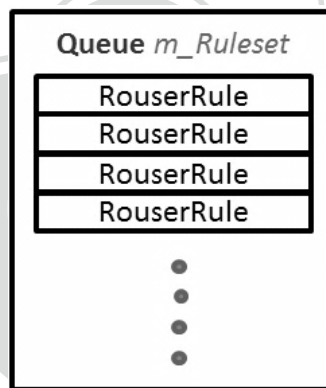


Figure 21. A data structure of the rule set: m_Ruleset.

The whole data structure of the rule model built by ROUSER is shown in Figure 22:

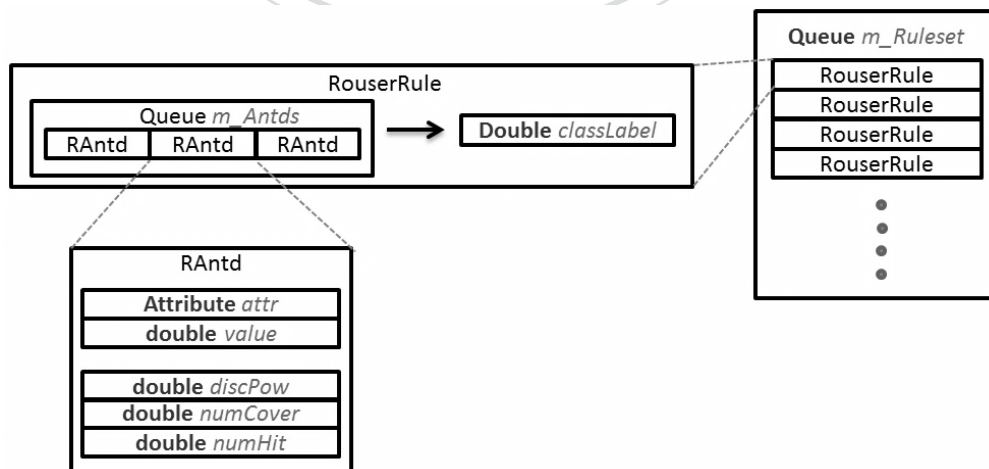


Figure 22. The data structure of ROUSER's rule model.

4.3 ROUSER

Following the separate-and-conquer algorithm, ROUSER is implemented under the Weka environment. As mentioned in Section 4.1.1, the **BuildClassifier()** function must be implemented

4.3.1 BuildClassifier()

In the **BuildClassifier()** function shown in Figure 23, the **oneClassRule()** is an implement of the separate-and-conquer algorithm, which build rules for one chosen class. The **oneClassRule()** function is called for each class by ascending class order, since we adapt the **ascending ordered rules** strategy here, which is also adapted by RIPPER.

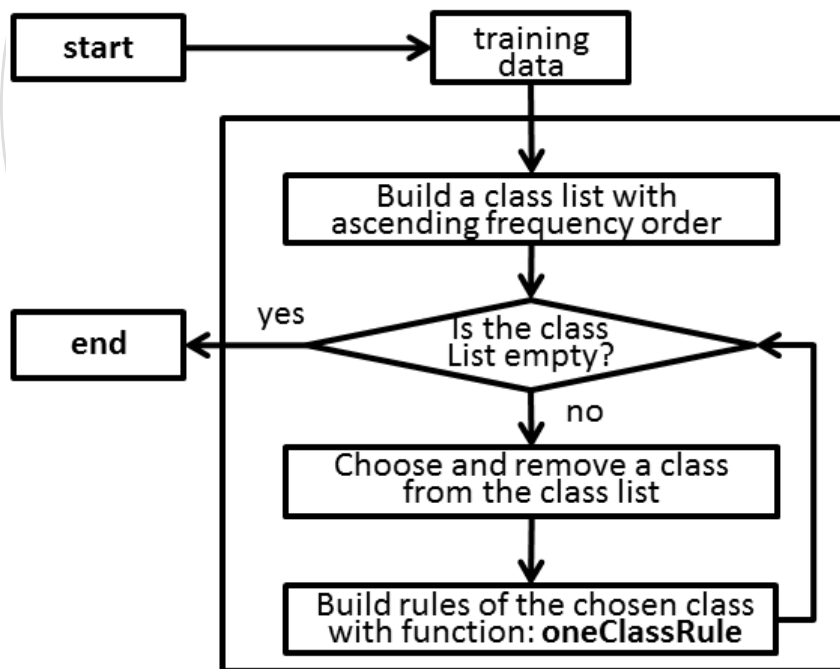


Figure 23. The flow chart of **BuildClassifier()**.

4.3.2 OneClassRule()

The function **OneClassRule()** shown in Figure 24 is an implement of the separate-and-conquer algorithm. The training data is first transformed into a rough set of the

chosen class, which split the original data into three parts, and we make the boundary region empty to accelerate further processes. If there are contradicted instances in the data set, they will be in the boundary region, and there are many methods to handle the contradictions, such as assigning the most frequent class label to the contradicted instances. We choose a simple method: Deleting the instances in the boundary region. After that we build a rule from the rough set by the **grow()** function. The rule is concatenated at the end of the rule set right after it is built. After a rule is built, the positive instances explained by the rule are removed from the positive region. The remaining instances in the rough set will then be used to build another rule iteratively until all instances are explained.

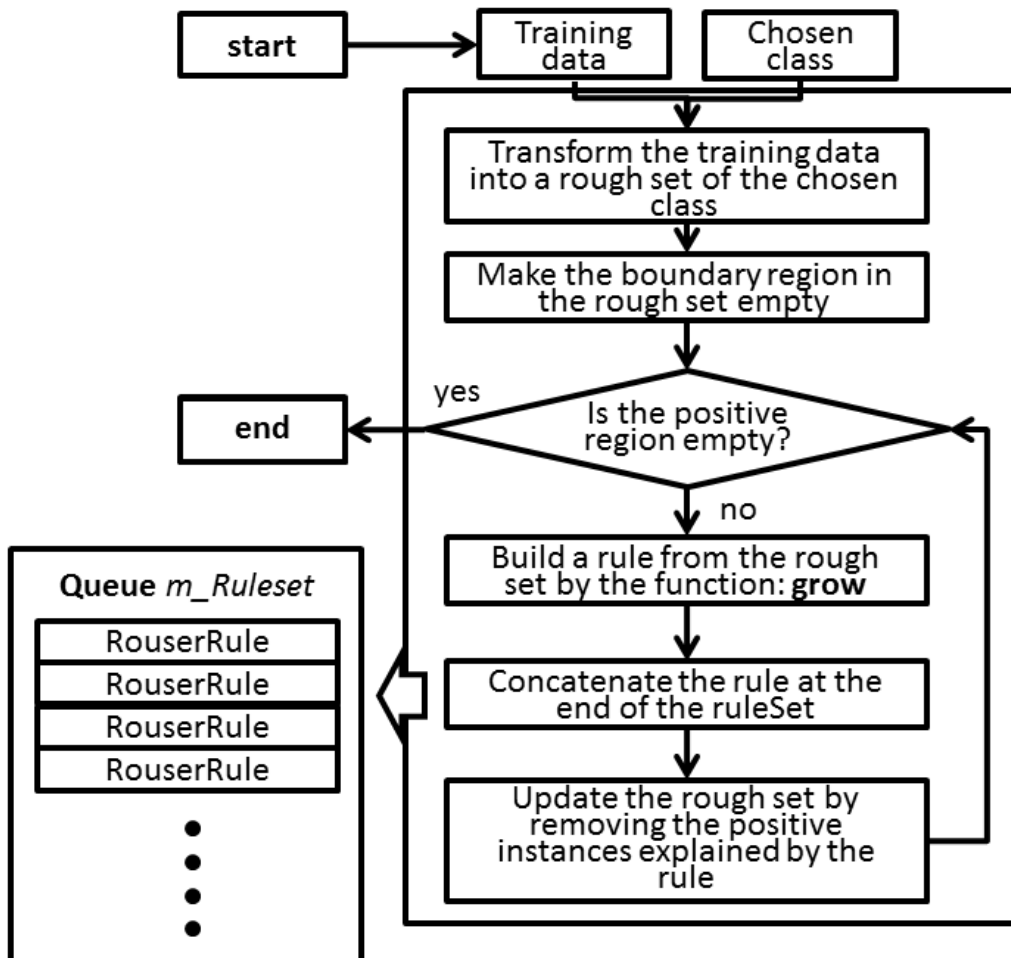


Figure 24. The flow chart of **OneClassRule()**.

4.3.3 grow()

The **grow()** function shown in Figure 25 builds a rule that explains some of the positive instances and none of the negative instances in the rough set. At the beginning an empty rule is built. DiscPow and Chi-Squared value of each attribute are calculated, and the rule is enriched by the antecedents built by the **bestAntd()** function. The longer the rule grows, the fewer the negative instances are covered. The rule is finally done when none of the negative instances are covered by the rule.

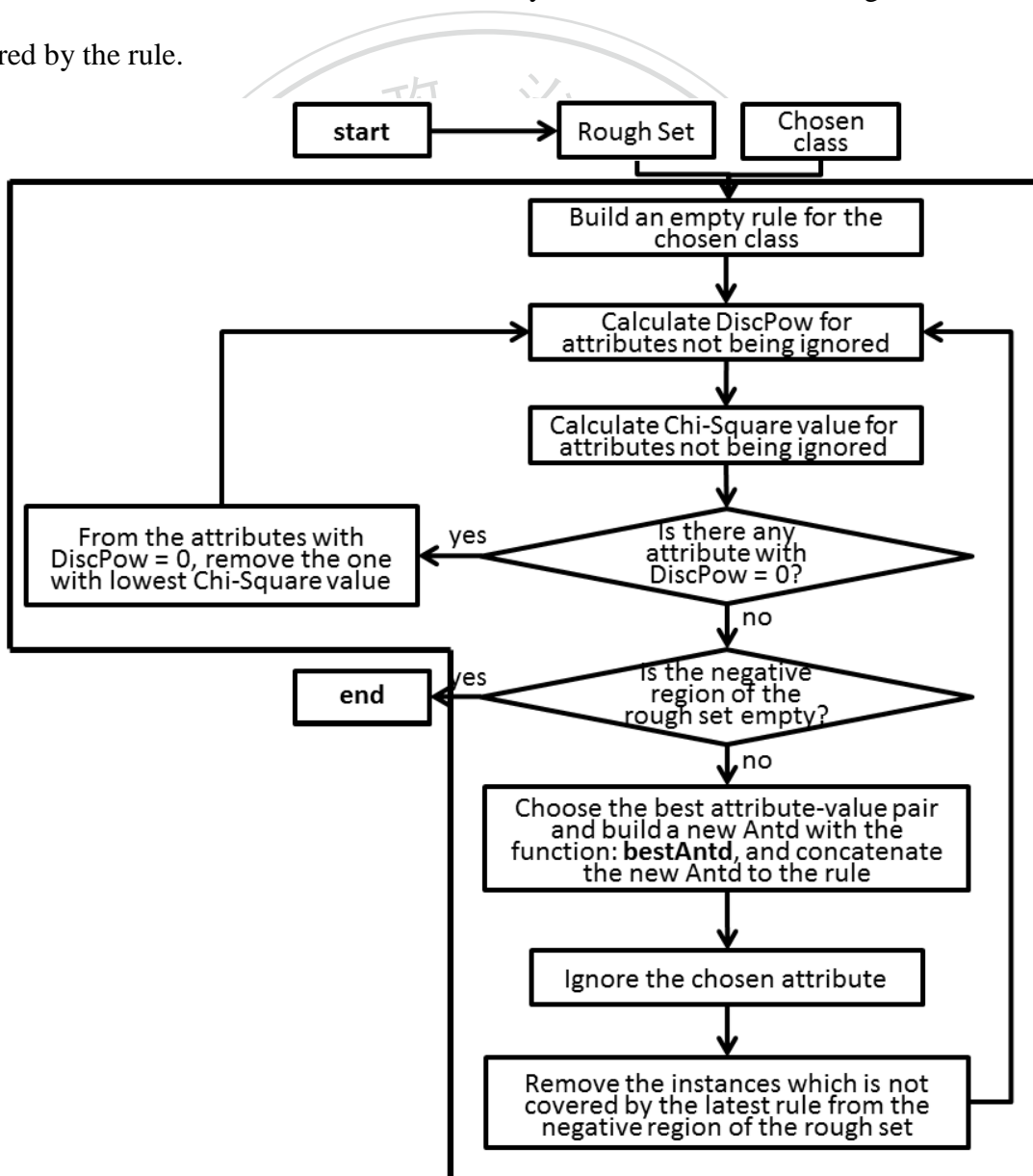


Figure 25. The flow chart of **grow()**.

4.3.4 BestAntd()

BestAntd() chooses the best pair of attribute and value to grow the rule, and is the same as the **CHOOSE_ATTR&VALUE()** function in the pseudo code in **grow()** function in Figure 14.



CHAPTER 5

EXPERIMENT AND RESULTS

5.1 Environmental Setting

The experiment is executed on a computer with Windows7 32bit operating system. The memory is 4GB DDR3 SDRAM 1333Mhz, and the chipset is Intel Q67 Express, the CPU is Intel Core i7 -2600, 3.4GHz. The Weka's version is 3.6.5.

5.2 Data Sets

The data sets used for experiments are all available from UCI Machine Learning Repository [23], and the data sets which are originally nominal data are shown in Table VII, and the discretized data sets which originally contain some real number data are shown in Table VIII. They are collected from different application domains, such as biology, gaming, politics, and marketing; the number of their attributes ranges from 5 to 69; the number of their classes ranges from 2 to 24; since the class numbers are different in each data set, we use bar charts to visualize the class distributions, for some of them, the class distributions are imbalanced; and some data sets are with missing values on some attributes.

The data set names with the “_dis” concatenated behind are not pure nominal data originally. We perform discretization on these data sets, and the details about what attributes are discretized and how they are discretized are shown in Table IX.

Table VII. Original nominal data sets.










<i>Data name</i>	<i>#instances</i>	<i>#attributes including class</i>	<i>Class distribution</i>	<i>missing value</i>
<i>Agaricus-lepiota</i>	8124	23		yes
<i>Audiology.standardized</i>	226	69		yes
<i>Car</i>	1728	6		no
<i>House-votes-84</i>	435	16		yes
<i>Kr-vs-kp</i>	3196	36		no
<i>Nursery</i>	12960	8		no
<i>Promoters</i>	106	58		no
<i>Splice</i>	3190	61		no
<i>Tic-tac-toe</i>	958	9		no

Table VIII. Discretized data sets.


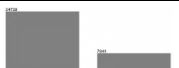




<i>Data name</i>	<i>#instances</i>	<i>#attributes including class</i>	<i>Class distribution</i>	<i>missing value</i>
<i>Abalone_dis</i>	4177	9		no
<i>Adult_dis</i>	32561	15		no
<i>Australian_dis</i>	690	15		no
<i>Balance-scale_dis</i>	625	5		no
<i>German_dis</i>	1000	21		no
<i>Hearts_dis</i>	270	14		no

Table IX. Details of discretization.

<i>Data name</i>	<i>Supervised discretization</i>	<i>Equal bean discretization (number of bean)</i>	<i>Numerical to nominal</i>
<i>Abalone</i>	2,3,4,5,6,7,8	9(5)	
<i>Adults</i>	1,5,11,12,13	3(10)	
<i>Australian</i>	2,3,7,10,13,14		1,4,5,6,8,9,11,12,15
<i>Balance-scale</i>			1,2,3,4
<i>German</i>	2	5(10),13(10)	8,11,16,18,21
<i>Heart</i>	8,10	1(5),4(5),5(5)	2,3,6,7,9,11,12,13,14

We defined four types of contradictions in section 3.2, and the number of contradictions in each data set is shown in Table X and Table XI.

Table X. Number of contradictions in original nominal data sets

<i>Data sets</i>	<i>Number of instances</i>	<i>Number of contradictions</i>			
		<i>type 1</i>	<i>type2</i>	<i>type3</i>	<i>type4</i>
<i>Agaricus-lepiota</i>	8124	0	0	0	0
<i>Audiology.standardized</i>	200	0	0	0	0
<i>Car</i>	1728	0	0	0	0
<i>House-votes-84</i>	435	293	149	0	0
<i>Kr-vs-kp</i>	3196	0	0	0	0
<i>Nursery</i>	12960	0	0	0	0
<i>Promoters</i>	106	0	0	0	0
<i>Splice</i>	3190	2	2	2	2
<i>Tic-tac-toe</i>	958	0	0	0	0

Table XI. Number of contradictions in discretized data sets.

<i>Data sets</i>	<i>number of instances</i>	<i>Number of contradictions</i>			
		<i>type 1</i>	<i>type2</i>	<i>type3</i>	<i>type4</i>
<i>Abalone_dis</i>	4177	3190	3190	3190	3190
<i>Adult_dis</i>	32561	4431	4431	4431	4431
<i>Australian_dis</i>	690	29	29	29	29
<i>Balance-scale_dis</i>	625	0	0	0	0
<i>German_dis</i>	1000	0	0	0	0
<i>Hearts_dis</i>	270	2	2	2	2

5.3 Results

We design several experiments to examine ROUSER's performance in different situations. We use 10-fold cross-validation to evaluate the classification performance.

The results for the data sets which are original nominal are summarized in Table XII. The numbers reported in Table XII are accuracy rates in percentage, and the maximum values are in bold, and the minimum values are underlined. As we mentioned in section 3.2 that ROUSER has seven choices to search for the attribute-value pair to grow a rule, and the results of all the seven choices are shown in Table XII. Four out of nine accuracy results of ROUSER_6 are better than or the same as both J48 and JRip. On two data sets ROUSERs are outperformed by JRip and J48. ROUSER_1 and ROUSER_6 are the most stable versions among these seven versions, and their accuracy rates are comparable to J48 and JRip. However, ROUSER does not perform well on the data sets *car* and *splice*. We think that there are no optimization stage and pruning methods in ROUSER (but there are in RIPPER) and overfitting occurs. The *car* data set is a data set with hierarchy structure which is easily captured by a tree structure, and we think that this is the reason that J48 outperforms JRip and ROUSER. The embedded feature selection method of ROUSER performs well on the *splice* data set (as shown in experiment results later), but ROUSER itself does not perform well on this data set. We think that this might be the overfitting problem. A deeper investigation of this will be part of the future work.

We design an experiment to examine ROUSER's capability to handle missing values. We choose three data sets: *Kr-vs-kp*, *Nursery*, and *Tic-tac-toe*, to produce artificial data sets with missing values. The missing values are distributed randomly in each attribute with the same percentage (10%, 20%, 30%), while the distributions of missing values are different between attributes. The class attribute has no missing values, and besides the class attribute, no

instances have all missing values. Missing values cause contradictions, and the number of contradictions in each data set is shown in Table XIII.

Table XII. Results for original nominal data sets.

<i>Data sets</i>	<i>Accuracy (%)</i>								
	<i>ROUSER</i>							<i>JRip</i>	<i>J48</i>
	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>	<i>6</i>	<i>7</i>		
<i>Agaricus-lepiota</i>	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
<i>Audiology.standardized</i>	78.0	75.5	77.0	77.5	76.5	77.0	76.5	<u>71.5</u>	77.5
<i>Car</i>	84.5	85.2	83.0	<u>82.2</u>	84.8	83.5	85.3	88.3	92.7
<i>House-votes-84</i>	93.3	94.5	<u>93.1</u>	94.5	94.5	94.7	93.8	95.6	96.3
<i>Kr-vs-kp</i>	99.2	99.3	99.5	<u>91.9</u>	99.3	99.6	99.4	99.2	99.5
<i>Nursery</i>	98.3	97.8	98.3	<u>76.9</u>	97.1	98.3	98.0	96.8	97.2
<i>Promoters</i>	80.2	83.0	<u>74.5</u>	75.5	84.0	84.0	79.3	82.1	81.1
<i>Splice</i>	83.0	82.6	<u>79.2</u>	82.0	83.2	80.3	84.4	93.8	94.2
<i>Tic-tac-toe</i>	96.9	91.8	<u>96.1</u>	91.7	94.2	97.2	96.8	97.7	<u>85.8</u>

Table XIII. Number of contradictions in artificial missing values in data sets.

<i>Data sets</i>	<i>total</i>	<i>number of contradictions</i>			
		<i>type1</i>	<i>type2</i>	<i>type3</i>	<i>type4</i>
<i>Kr-vs-kp 10% average</i>	3196	604.8	451.9	0	0
<i>Kr-vs-kp 20% average</i>	3196	1865	830.4	0	0
<i>Kr-vs-kp 30% average</i>	3196	2892.5	574.1	0	0
<i>Nursery 10% average</i>	12960	11260.5	11198.8	155.6	0
<i>Nursery 20% average</i>	12960	12958.6	12958.4	289.5	0
<i>Nursery 30% average</i>	12960	12960	12960	636.8	0
<i>Tic-tac-toe 10% average</i>	958	214.2	196.2	0	0
<i>Tic-tac-toe 20% average</i>	958	693.9	620.5	0.4	0
<i>Tic-tac-toe 30% average</i>	958	928.4	876.5	1.2	0

The results are shown in Table XIV. The numbers reported in Table XIV are accuracy rates in percentage, and except those of the original (0%) data sets, each accuracy rate is the average accuracy rate of 10 different artificial data sets with the same rate of missing value.

The results indicate that the performances of ROUSER are similar to JRip in Kr-vs-kp

and Tic-tac-toe data sets, better than J48 in Tic-tac-toe data set, and worse than J48 in Kr-vs-kp data set. The accuracy rate of ROUSER drops faster than JRip and J48 do in Nursery data set when missing value percentage rises. We speculate that the Nursery data set with missing values have too many type 3 contradictions, which will be ignored by ROUSER as we mentioned in section 3.2, or there are too many type 1 contradictions and this makes ROUSER miscalculate the potential boundary region. To address the problem, we may adapt probability theory and assign contradicted instances to the class with higher probability.

Table XIV. Results for artificial missing values in data sets.

<i>data sets</i>	<i>Accuracy (%)</i>			
	<i>ROUSER_1</i>	<i>ROUSER_6</i>	<i>JRip</i>	<i>J48</i>
Kr-vs-kp 0% (original)	99.2	99.6	99.2	99.5
Kr-vs-kp 10% average	91.2	90.6	91.2	94.0
Kr-vs-kp 20% average	85.5	84.8	84.3	88.6
Kr-vs-kp 30% average	78.1	78.0	78.8	84.1
Nursery 0% (original)	98.3	98.3	96.8	97.2
Nursery 10% average	56.5	57.1	83.8	88.3
Nursery 20% average	41.7	44.2	74.3	80.5
Nursery 30% average	39.2	40.0	66.4	73.7
Tic-tac-toe 0% (original)	96.9	97.2	97.7	85.8
Tic-tac-toe 10% average	86.4	86.3	89.2	79.8
Tic-tac-toe 20% average	80.1	79.6	80.9	73.1
tic-tac-toe 30% average	74.2	74.9	74.0	70.0

We design an experiment to examine the “ordered rules” strategy. ROUSER with ascending order rules are compared with ROUSER with descending order rules. The experimental results are given in Table XV. Each result is presented in two numbers, and the upper number is the original accuracy with ascending order rules, and the lower number is the difference after we switch to descending ordered rules. Ascending order is apparently better than descending order only in the *Audiology.standardized* data set, which has 24 class labels

with imbalanced distribution. However descending order is better in the *Car* data set and the accuracy becomes comparable with the accuracy of JRip and J48. Descending order is better than ascending order in the *Splice* data set. However the accuracy is still not comparable with the accuracy of JRip and J48. We observe that imbalanced multi-class data sets are sensitive to the ordered rule strategy. A deeper investigation of this will be part of the future work.

Table XV. Results for ordered rule strategy.

<i>Data sets</i>	<i>Accuracy (%)</i>						
	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>	<i>6</i>	<i>7</i>
<i>Agaricus-lepiota</i>	100.0 0.0	100.0 0.0	100.0 0.0	100.0 0.0	100.0 0.0	100.0 0.0	100.0 0.0
<i>Audiology.standardized</i>	78.0 -7.0	75.5 -3.5	77.0 -5.5	77.5 -2.5	76.5 -3.5	77.0 -5.5	76.5 -2.5
<i>Car</i>	84.5 +6.0	85.2 +4.4	83.0 +7.2	<u>82.2</u> +4.3	84.8 +4.9	83.5 +6.3	85.3 +5.3
<i>House-votes-84</i>	93.3 +1.6	94.5 -0.5	<u>93.1</u> +0.2	94.5 -0.0	94.5 -0.5	94.7 -0.7	93.8 -0.5
<i>Kr-vs-kp</i>	99.2 0.0	99.3 +0.1	99.5 -0.3	<u>91.9</u> 0.0	99.3 +0.1	99.6 -0.4	99.4 -0.1
<i>Nursery</i>	98.3 +0.3	97.8 -0.4	98.3 +0.4	<u>76.9</u> 0.1	97.1 0.0	98.3 +0.2	98.0 +0.3
<i>Promoters</i>	80.2 +2.8	83.0 +1.9	<u>74.5</u> +10.4	75.5 +0.9	84.0 +0.9	84.0 +0.0	79.3 -1.0
<i>Splice</i>	83.0 +4.3	82.6 +4.0	<u>79.2</u> +8.0	82.0 +0.6	83.2 +3.5	80.3 +5.3	84.4 +3.4
<i>Tic-tac-toe</i>	96.9 +0.2	91.8 +1.7	96.1 +2.5	91.7 +0.7	94.2 +0.4	97.2 +1.5	96.8 +1.4

We design an experiment to prove that Chi-Square value is useful in the rule growing phase of ROUSER. In our original design, we adapt the Chi-Square value to reduce the attributes iteratively. To make a contrast, we replace the Chi-Square value with Information Gain, which is provided by Weka, and the results of the experiments on such a replacement are

given in Table XVI. The numbers reported in Table XVI are accuracy rates in percentage. Each result is presented in two numbers, and the upper number is the original accuracy before we replace Chi-Square value with Information Gain, and the lower number is the difference after we do such a replacement. We discover that the performance of Chi-Square version is obviously better than Information Gain version on the *Audiology.standardized* data set. The performances on the *Promoters* data set which are originally bad become better after we replace Chi-Square value with the Information Gain. However, the performances which are originally good become worse. Our conclusion is that ROUSER_1 and ROUSER_6 with Chi-Square feature selection are still more stable than all the other combinations.

Table XVI. Results of replacing Chi-Square value with Information Gain in ROUSER.

<i>Data sets</i>	<i>Accuracy (%) with Information Gain feature selection</i>						
	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>	<i>6</i>	<i>7</i>
<i>Agaricus-lepiota</i>	100.0 0.0	100.0 0.0	100.0 0.0	100.0 0.0	100.0 0.0	100.0 0.0	100.0 0.0
<i>Audiology.standardized</i>	78.0 -5.0	75.5 -3.0	77.0 -3.5	77.5 -5.5	76.5 -4.0	77.0 -3.5	76.5 -3.5
<i>Car</i>	84.5 +0.3	85.2 +0.2	83.0 +0.2	<u>82.2</u> 0.0	84.8 0.0	83.5 +0.1	85.3 -0.1
<i>House-votes-84</i>	93.3 +1.6	94.5 +1.1	<u>93.1</u> +0.9	94.5 -0.9	94.5 +1.1	94.7 -0.2	93.8 -0.5
<i>Kr-vs-kp</i>	99.2 +0.2	99.3 -0.1	99.5 0.0	<u>91.9</u> 0.0	99.3 -0.1	99.6 -0.2	99.4 0.0
<i>Nursery</i>	98.3 +0.2	97.8 -0.1	98.3 +0.2	<u>76.9</u> +1.4	97.1 +0.1	98.3 +0.1	98.0 0.0
<i>Promoters</i>	80.2 0.0	83.0 -4.7	<u>74.5</u> +11.3	75.5 +8.5	84.0 -3.8	84.0 -5.7	79.3 +6.5
<i>Splice</i>	83.0 -0.3	82.6 -1.0	<u>79.2</u> -1.1	82.0 -3.2	83.2 -1.6	80.3 -1.6	84.4 +0.5
<i>Tic-tac-toe</i>	96.9 +0.3	91.8 +0.6	96.1 0.0	91.7 -0.3	94.2 -0.6	97.2 0.0	96.8 0.0

The experimental results of the discretized data set are summarized in Table XVII. The numbers reported in Table XVII are accuracy rates in percentage. The performances of ROUSER on discretized data set are not as well as the performances in the original nominal data sets, and we speculate the reason is that discretization may assign the same value to different real numbers, and this may make instances indiscernible and be considered as contradictions by ROUSER. As we mentioned before, ROUSER simply discards the contradictions, and hence it shows poor performance on these discretized data sets.

Similarly, we can adapt probability theory and assign contradicted instances to the class with higher probability. We can also design an embedded discretization method for ROUSER, like what is done in JRip or J48, to handle real number data directly. From Table XI we discover that there are many contradictions in *Abalone_dis*, *Adult_dis*, and *Australian_dis* data sets, and hence ROUSER performs not as well as JRip and J48 on these data sets.

Table XVII. Results for discretized data sets.

<i>Data sets</i>	<i>Accuracy (%)</i>								
	<i>ROUSER</i>							<i>JRip</i>	<i>J48</i>
	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>	<i>6</i>	<i>7</i>		
<i>Abalone_dis</i>	74.2	71.6	73.5	71.4	69.8	72.4	74.9	77.2	77.6
<i>Adult_dis</i>	82.6	83.3	82.2	77.6	82.7	81.4	83.2	84.0	86.8
<i>Australian_dis</i>	82.0	78.3	81.2	79.7	79.1	77.1	80.7	85.8	86.2
<i>Balance-scale_dis</i>	74.1	73.9	73.8	56.2	72.8	72.6	73.0	73.8	63.2
<i>German_dis</i>	68.5	66.7	64.2	71.1	68.3	65.7	69.0	71.0	71.1
<i>Hearts_dis</i>	73.7	73.0	74.4	70.0	74.8	75.9	74.8	77.8	75.2

The execution time in millisecond of ROUSER, JRip and J48 are shown in Table XVIII. We measure the training time of entire data set, instead of 10-fold. It is clear that tree-based strategy overwhelms the separate-and-conquer strategy in execution time, and the reason is simple: Unlike the tree-based strategy which ignores the data divided away, although the

separate-and-conquer strategy “separates” positive data in each iteration of building a rule, it needs all the negative data to stay in memory to complete this mission, and hence the same negative data will be executed for several times.

There are two more reasons for ROUSER’s high execution time. First, DiscPow itself is not so “greedy”. To explain this, we make a comparison with Information Gain, which is adapted by C4.5 and RIPPER. When calculating the Information Gain for choosing an attribute, only the attribute itself and the class attribute are involved in the calculation. However, when calculating the DiscPow of an attribute, the whole decision table is involved, since we need to compare all the values between each pair of records. Second, ROUSER has no pruning methods and may build precise rules to explain only a few data records, and hence too many rules are built and time is wasted.

Table XVIII. Training time.

<i>Data sets</i>	<i>Training time (ms)</i>								
	<i>ROUSER</i>							<i>JRip</i>	<i>J48</i>
	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>	<i>6</i>	<i>7</i>		
<i>Agaricus-lepiota</i>	68398	67689	72931	72715	68608	70509	69899	690	102
<i>Audiology.standardized</i>	4242	4048	4559	4187	4220	4411	3953	36	9
<i>Car</i>	7025	10240	7594	7524	10511	7369	8847	483	5
<i>House-votes-84</i>	212	236	268	226	233	270	275	4	2
<i>Kr-vs-kp</i>	57657	23548	58968	16565	23893	53409	34280	280	22
<i>Nursery</i>	598793	773254	636622	415997	826664	633169	840923	30428	26
<i>Promoters</i>	184	219	162	191	218	167	235	4	2
<i>Splice</i>	1003074	1364238	1441220	676261	1352105	1247409	701302	297	38
<i>Tic-tac-toe</i>	440	1442	1126	495	2166	573	528	31	3

If we consider only the calculation complexity, ROUSER_{1~3} should be faster than ROUSER_{4~6}, since searching for the MAX purity takes more time than searching for the FREQUENT MAX purity. However, the results tell us a different story. The reason can be discovered by examining the rule set size, as shown in Table XIX, where each rule set is built

from entire data set instead of data sets generated by the 10-fold cross-validation method. Since the calculation of DiscPow is more complex than purity, and DiscPow will be calculated several times when generating a rule, the rule set size dominates the execution time. We also discover that JRip’s rule set size is usually smaller than ROUSER, and that is because JRip adapts some pruning methods and optimization methods, and hence it makes the rule set size smaller. Some rule set size are extremely high, while their accuracy is low, and this could be considered as an overfitting problem, and we think this might be the reason why ROUSER performs not well on *Car* and *Splice* data sets.

Table XIX. Rule set size.

<i>Data sets</i>	<i>Rule set size</i>								
	<i>ROUSER</i>							<i>JRip</i>	<i>J48</i>
	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>	<i>6</i>	<i>7</i>		
<i>Agaricus-lepiota</i>	12	11	12	12	11	11	11	8	24
<i>Audiology.standardized</i>	55	53	54	52	54	54	52	27	31
<i>Car</i>	230	265	229	230	270	229	230	97	131
<i>House-votes-84</i>	12	13	15	13	13	16	16	10	6
<i>Kr-vs-kp</i>	29	25	31	6	25	27	25	18	31
<i>Nursery</i>	588	671	569	389	829	569	585	317	359
<i>Promoters</i>	8	9	7	8	9	7	9	9	19
<i>Splice</i>	295	346	362	125	329	324	242	63	184
<i>Tic-tac-toe</i>	19	46	37	19	79	21	17	12	95

We design an experiment to prove the feature selection method embedded in ROUSER is useful. The feature selection method is in the grow function of ROUSER, which iteratively ignores an attribute with DiscPow=0 and the lowest Chi-Square value. This method selects attributes for one class, and hence we perform the feature selection method on all classes and union each result as the final result. We name it **DiscPow_Chi** method for convenience. **DiscPow_Chi** is a deterministic feature selection method which returns a fixed number of

selected attributes and needs no additional threshold settings, while the Information Gain method simply returns the rank of all attributes and an appropriate threshold is needed. Thus we first choose **CfsSubsetEval** method together with **BestFirst** search method provided by Weka, which is also a deterministic feature selection, as the comparison. We compare the accuracy of JRip and J48 between the original data sets and the feature selected data sets. The results of the experiment are shown in Table XX. In data sets *car* and *nursery*, **CfsSubsetEval** method chooses only 1 attribute, which is obviously not able to represent the original data sets. In data set *splice*, **DiscPow_Chi** selects half amount of attributes than **CfsSubsetEval**, while the accuracy rates of both JRip and J48 are merely the same. In the data sets *house-votes-84* **CfsSubsetEval** outperforms **DiscPow_Chi** by choosing fewer attributes while keeping the high accuracy rate, and in the data set *promoters* **CfsSubsetEval** outperforms **DiscPow_Chi** by higher accuracy rate. Both **DiscPow_Chi** and **CfsSubsetEval** failed in the *tic-tac-toe* data set, but the problem of **CfsSubsetEval** is far more serious. To sum up, it is more possible for **DiscPow_Chi** than for **CfsSubsetEval** to avoid accuracy loss.

We also make a comparison to Information Gain feature selection provided by Weka, which ranks each attribute from high to low. We choose attributes with higher rank, and the amount is the same with what **DiscPow_Chi** chose. The results are shown in Table XXI. On data sets *Agaricus-lepiota*, *Audiology.standardized* and *Kr-vs-kp*, **DiscPow_Chi** performs better than Information Gain feature selection, while Information Gain feature selection performs better on the *Promoters* data set. The other results are similar. The accuracy results show that **DiscPow_Chi** is no worse than Information Gain feature selection, but even better, since **DiscPow_Chi** is deterministic, and save the work of determining the number of selected attributes. The idea is very different between **DiscPow_Chi** and Information Gain feature selection. **DiscPow_Chi** iteratively removes the attributes that we do not need, while the idea

of Information Gain feature selection is to select what we want.

Table XX. The comparison of DiscPow_Chi and CfsSubsetEval.

<i>Data name</i>	<i>Feature selection method</i>	<i>Number of attributes selected</i>	<i>Accuracy</i>	
			<i>JRip</i>	<i>J48</i>
<i>Agaricus-lepiota</i>	none	22	100.0	100.0
	DiscPow_Chi	5	100.0	100.0
	CfsSubsetEval	4	99.0	99.0
<i>Audiology.standardized</i>	none	69	71.5	77.5
	DiscPow_Chi	13	69.0	76.0
	CfsSubsetEval	14	71.0	77.0
<i>Car</i>	none	6	88.3	92.7
	DiscPow_Chi	6	88.3	92.7
	CfsSubsetEval	1	70.0	70.0
<i>House-votes-84</i>	none	16	95.6	96.3
	DiscPow_Chi	8	95.4	95.9
	CfsSubsetEval	4	95.6	96.0
<i>Kr-vs-kp</i>	none	36	99.2	99.5
	DiscPow_Chi	29	99.0	99.1
	CfsSubsetEval	7	94.1	94.0
<i>Nursery</i>	none	8	96.8	97.2
	DiscPow_Chi	8	96.8	97.2
	CfsSubsetEval	1	71.0	71.0
<i>Promoters</i>	none	57	82.1	68.9
	DiscPow_Chi	4	82.1	76.4
	CfsSubsetEval	6	86.8	79.3
<i>Splice</i>	none	60	93.8	94.2
	DiscPow_Chi	11	93.7	94.3
	CfsSubsetEval	22	94.4	94.4
<i>Tic-tac-toe</i>	none	9	97.7	85.8
	DiscPow_Chi	8	90.0	85.2
	CfsSubsetEval	5	76.3	78.2

Table XXI. The comparison of DiscPow_Chi and Information Gain feature selection.

<i>Data sets</i>	<i>Number of selected attributes</i>	<i>Accuracy (%)</i>			
		<i>JRip</i>		<i>J48</i>	
		<i>DiscPow_Chi</i>	<i>InfoGain</i>	<i>DiscPow_Chi</i>	<i>InfoGain</i>
<i>Agaricus-lepiota</i>	5/22	100.0	99.9	100.0	99.9
<i>Audiology.standardized</i>	13/69	69.0	66.5	76.0	70.5
<i>Car</i>	6/6	88.3	88.3	92.7	92.7
<i>House-votes-84</i>	8/16	95.4	95.6	95.9	95.2
<i>Kr-vs-kp</i>	29/36	99.0	96.7	99.1	97.2
<i>Nursery</i>	8/8	96.8	96.8	97.2	97.2
<i>Promoters</i>	4/57	82.1	84.0	76.4	84.0
<i>Splice</i>	11/60	93.7	95.2	94.3	94.5
<i>Tic-tac-toe</i>	8/9	90.0	91.3	85.2	85.3

5.4 Summary

The performance of ROUSER in accuracy is usually no worse and sometimes better than that of JRip or J48. However, the time cost is high. ROUSER is sensitive to contradictions which are originally in the data, since ROUSER simply ignores contradictions. The embedded feature selection method is deterministic and more possible to avoid accuracy loss. ROUSER has some good properties, and how to keep these good properties while avoiding the shortcomings would be the focus of the feature work.

CHAPTER 6

CASE STUDY

6.1 Introduction

The objective of this case study is to find out the cause of machine fault of a roughing mill in a hot strip mill of the largest steel making company in Taiwan. First we will introduce how to implement data mining techniques in a cooperative data analysis. Second we will describe the background knowledge of the case study and make a brief explanation to the data. Then we will show how we build up models and rules to analyze the cause of machine fault. After that, we will look into the data and inspect the rules we built. Finally we will give a conclusion about this case study.

6.1.1 Back Ground Knowledge

We first introduce the background knowledge about the manufacturing process. The function of a hot strip mill is to turn a slab into a coil for the convenience of further process. There are two hot strip mills, and the structures inside them are different. Our attention is on one of the two hot strip mills. After a slab enters the hot strip mill, it must be heated up at the furnace to become soft. A prepared slab will then enter the roughing mill. A roughing stand contains two parts, the edge mill and the rough mill; the former adjusts the slab in a good width, while the later thins the slab. After this, a slab becomes a transfer bar. The transfer bar will be sliced into pieces by the crop shear, and finally it enters the finish mill and becomes a coil after cooling.

6.1.2 Problem

The problem occurs at the rough mill. The top and bottom working rolls of the rough mill directly contact the slab when rolling, and the torque comes from the engines connected by the spindle. In these recent years the spindles broke frequently, and the experts suspect that the cause is that a slip happens during the rolling process. The rough mill rolls the slab back and forth for 5 passes, and each rolling pass makes the slab thinner. A slip may occur in each rolling pass, and the spindles may suffer unexpected impact and a slip may lead to metal fatigue.

6.1.3 Data

The data collected from the mill can be roughly categorized into three types, namely the materials, the mill, and the rolls.

Material Data

Material data contains the features about the material, such as the steel family, and the steels in the same family have similar properties. The material data also contains the thickness drafts of each pass performed by the rough mill.

Mill Data

Mill data comes from the mill itself. Some attributes like the moving speed of a slab are not easily to measure directly, and hence the experts measure the rolling speed from the mill to represent the slab moving speed; the speed draft of a slab is also a parameter setting of the mill. The slab has a threading speed, which is the initial speed of threading. The running speed is the speed right before the slab enters the mill. Only the default settings of these two speeds are recorded. The roll torque of working rolls is generated by the motors of mill. The roll force pressed on the slab is also generated by the motors of mill.

Roll Data

There are two working rolls, top and bottom, and there are also plenty of measurement results about the working rolls. Here we introduce the rolling torque only. The difference between the roll torque and the rolling torque is that the roll torque is measured from the motors of the mill while the rolling torque is measured from the working rolls themselves.

6.2 Model Building

6.2.1 Data Preprocessing

Data Cleaning

There are 21,907 data records and 187 attributes (excluding the class attributes) in the original data set, which is collected from the hot strip mill for 2 months. After data cleaning, including removing error data records, redundant attributes, duplicate attributes, serial numbers, and time stamps, 21,891 data records and 172 attributes (excluding the class attributes) remain.

Data Reformatting

Each data record is bound to a particular piece of material, which is originally a slab and finally a coil, and hence data collected from 5 passes sticks together in one record. The static information such as material data we introduced before is also included. Torque ratio of each pass is also in one record. It is obvious that data in this format is not suitable for any classification algorithm to analyze, so we reformat the original data set into 5 data sets based on the pass number.

As we mentioned before, materials in the same family share similar physical properties, and hence we divide the original data for each family. There are 27 families in the original data set, and 5 passes for each record, and hence the original data set is reformatted into

27*5=135 data sets.

Data Discretization

Since the proposed ROUSER processes nominal data only, discretized data sets are made from the 135 data sets. The discretized method is provided by Weka, and it is an implementation of Fayyad & Irani's MDL method [6].

Data Integration: Torque Ratio

The torque ratio is the class attribute, and it is fused from the rolling torque we just mentioned. The value of torque ratio can be used to determine a slip is happening or not. The value range of torque ratio is [0, 1]. The safe range of pass 1 is (0.45, 0.55), and so is that of pass 2; the safe range is (0.4, 0.6) for pass 3; and it is (0.35, 0.75) for passes 4 and 5. The others are slip range.

Feature Selection

Some attributes are removed, and the reasons vary. Some of them are removed since they are already known to be dependent on the class attribute, and this type of attributes will dominate the results. However they are not helpful to explain the problem. Another reason of why the attributes are removed is that the timing they are measured is after the slip happens. Absolute time stamps and serial numbers are also removed.

Slip data records are rare in the final data sets.

6.2.2 Classification Algorithms

We use ROUSER to analyze the discretized data sets, and we use JRip and J48 provided by Weka to analyze data sets with real numbers. We use the default setting of JRip (-F 3 -N 2.0 -O 2 -S 1) together with two more settings (-F 3 -N 1.0 -O 0 -S 1 -E, -F 3 -N 1.0 -O 0 -S 1 -E -P). We also use the default setting of J48 (-C 0.25 -M 2) together with two more setting (-S -C 0.5 -M 1, -S -C 0.25 -M 2). Since the slip records are rare in the data set, the default

settings of JRip and J48 may consider them as noise and ignore them to pursue the overall accuracy; as a result, the models are too simple and explain nothing. So we try several different settings to remove the mechanisms which are designed to prevent models from growing too luxuriant and becoming over fitting.

6.3 Inspect the Result Rules

We generate many rule sets from 135 data sets with 3 algorithms and different settings, and we give some of the rules. Following are the rules generated by J48 on the data set of family 27 at pass 5:

(R2 roll torque_pass5 [kNm]\[1] >= 3078.667) => Torque ratio p5=[0.65,1] (27.0/0.0).

(R2 roll torque_pass5 [kNm]\[1] >= 2585.333) and

(R2 total roll force_pass5 [kN]\[1] <= 23046.67) => Torque ratio p5=[0.65,1] (3.0/0.0).

The first rule indicates that if the torque value measured from the motors of mill for pass 5 is bigger than or equal to 3078.667, then the torque ratio of pass 5 will be in the range [0.65,1], which is a slip range. The second rule indicates that if the torque value measured from the motors of mill for pass 5 is bigger than or equal to 2585.333, and the force value measured from the motors of mill for pass 5 is smaller than or equal to 23046.67, then the torque ratio of pass 5 will be in the slip range [0.65,1]. From these two rules we may conclude that when torque measured from the motors of mill is too high, and sometimes when the force measured from the motors of mill is too low, a slip may occurs.

Our job is to summarize the results and let the experts to inspect the results. We need feedbacks from the experts to improve the experiments.

Consider the rules provided above, the torque value measured from the motors of mill for pass 5 attribute appears twice, and the force value measured from the motors of mill for pass 5 appears once. The more frequent an attribute appears in the rules, the more important the attribute is, especially when we built plenty of rules.

From all rule sets we discover one same phenomenon that the torque measured from the motors of mill when biting in a slab is the most frequent attribute for passes 3, 4, and 5. The rolling speed measured from the motors of mill is the second most frequent attribute for passes 3, 4, and 5.

We also discover that some attributes never appear in any rule. This discovery may help the experts to reduce dimensions when building a predictor.

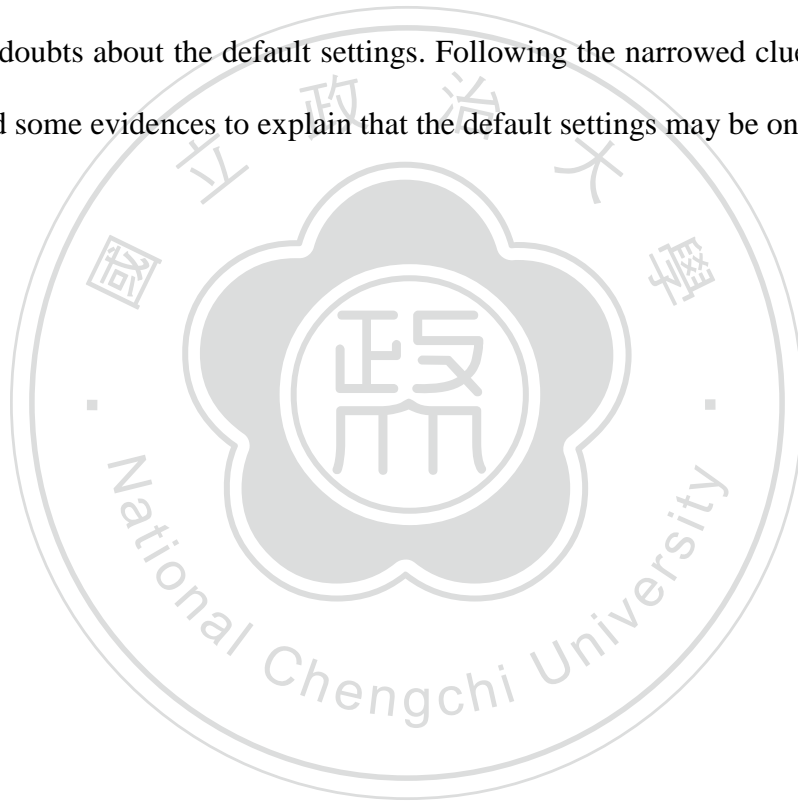
The third most frequent attribute for passes 3 and 4 is the rolling speed of the top working roll, which is preferred by JRip and J48, and the third most frequent attribute for passes 5 is the bottom working roll number, which is preferred by ROUSER and never chosen by JRip and J48. Both of these results are considered reasonable to experts. We discovered that JRip and J48 prefer real number attributes and they may overlook some important nominal attributes.

From the results we find that the default settings of running speed, threading speed, force, and torque, are involved, while the thickness draft of each pass are not involved. We look into the data to seek out the evidence of this discovery. First, we find that thickness draft of each pass differs only a little in each data record, which may be the reason of why the thickness is not involved. Second, we find that different records with exactly the same slab properties (such as family) and the same size of finished products may have different settings on the mill, and some setting combinations are rare with regard to the other records with the same slab properties, and these rare settings are usually accompanied with slip. We considered

this phenomenon as one of the causes of slip.

6.4 Summary

Through data mining techniques we narrow the exploring range of the problem happened in a rough mill. The attributes chosen by our experiments are considered reasonable, and we find that JRip and J48 are good at capturing important real number attributes, while ROUSER is good at capturing the important nominal attributes. The results also respond to the experts' doubts about the default settings. Following the narrowed clues, we look into the data and find some evidences to explain that the default settings may be one cause of slip.



CHAPTER 7

CONCLUSIONS AND FUTURE WORK

A rule-based classification algorithm named ROUSER is proposed. It is designed to process nominal data and generate human understandable decision rules. ROUSER uses a rough set approach as its search heuristic, and the rule generation method of ROUSER is based on the separate-and-conquer strategy.

As a prototype without the optimization stage or the pruning stage to reduce errors, ROUSER still provides classification performance comparable to or even better than that given by the rule-based or tree-based classification algorithms considered in experiments. Since the search heuristics of ROUSER is totally different from the search heuristics (Entropy and Information Gain) used by the other three algorithms, the results imply that the proposed *PotBound* and *DiscPow* are useful. This also shows the potential of ROUSER and gives an example of future work.

For future work, we plan to conduct more experiments, develop better strategies to select attributes and handle contradictions, and apply ROUSER to data sets obtained from a real-world case study.

REFERENCE

- [1] J. G. Bazan, H. S. Nguyen, S. H. Nguyen, P. Synak, J. Wróblewski, and blewski, "Rough set algorithms in classification problem," in Rough set methods and applications, ed: Physica-Verlag GmbH, 2000, pp. 49-88.
- [2] D. P. Bertsekas and J. N. Tsitsiklis, Neuro-Dynamic Programming: Athena Scientific, 1996.
- [3] W.W. Cohen, "Fast Effective Rule Induction," Proc. 12th Int'l Conf. Machine Learning (ICML), pp. 115-123, 1995.
- [4] C. Cortes and V. Vapnik, "Support-vector networks," Machine Learning, vol. 20, pp. 273-297, 1995.
- [5] J. Dai, Q. Xu, and W. Wang, "A comparative study on strategies of rule induction for incomplete data based on rough set approach," International Journal of Advancements in Computing Technology, vol. 3, p. 176–183, 2011.
- [6] U. M. Fayyad, K. B. Irani, "Multi-interval discretization of continuous-valued attributes for classification learning", presented at the Proceedings of 13th international joint conference on Artificial intelligence, 1022-1027, 1993.
- [7] J. Fürnkranz, "Separate-and-Conquer Rule Learning," Artif. Intell. Rev., vol. 13, pp. 3-54, 1999.
- [8] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: an update," SIGKDD Explor. Newsl., vol. 11, pp. 10-18, 2009.

- [9] L. Huan and R. Setiono, "Chi2: feature selection and discretization of numeric attributes," in *Tools with Artificial Intelligence*, 1995. Proceedings of IEEE Seventh International Conference on, 1995, pp. 388-391.
- [10] J. C. Huhn and E. Hullermeier, "FR3: A Fuzzy Rule Learner for Inducing Reliable Classifiers," *Fuzzy Systems, IEEE Transactions on*, vol. 17, pp. 138-149, 2009.
- [11] W. Jiabing, Z. Pei, W. Guihua, and W. Jia, "Classifying Categorical Data by Rule-Based Neighbors," in *Data Mining (ICDM)*, 2011 IEEE 11th International Conference on, 2011, pp. 1248-1253.
- [12] X. Jin, A. Xu, R. Bie, and P. Guo, "Machine Learning Techniques and Chi-Square Feature Selection for Cancer Classification Using SAGE Gene Expression Profiles Data Mining for Biomedical Applications." vol. 3916, J. Li, Q. Yang, and A.-H. Tan, Eds., ed: Springer Berlin / Heidelberg, 2006, pp. 106-115.
- [13] M. T. Mitchell, *Machine Learning*, 1997 :McGraw-Hill
- [14] G. Pagallo and D. Haussler, "Boolean Feature Discovery in Empirical Learning," *Machine Learning*, vol. 5, pp. 71-99, 1990.
- [15] Z. Pawlak, "Some Issues on Rough Sets," *Transactions on Rough Sets I*, vol. 3100, J. Peters, A. Skowron, J. Grzymala-Busse, B. Kostek, R. Swiniarski, and M. Szczuka, Eds., ed: Springer Berlin / Heidelberg, 2004, pp. 1-58.
- [16] Z. Pawlak, A. Skowron, "Rudiments of rough sets", *Information Sciences*, vol.177, no.1, pp.3-27, 2007.
- [17] J. R. Quinlan, "Induction of Decision Trees," *Machine Learning*, vol. 1, pp. 81-106, 1986.

- [18] J. R. Quinlan, C4.5: programs for machine learning: Morgan Kaufmann Publishers Inc., 1993.
- [19] J. Stefanowski and K. Slowiński, "Rough Set Theory and Rule Induction Techniques For Discovery of Attribute Dependencies in Medical Information Systems," Principles of Data Mining and Knowledge Discovery. vol. 1263, J. Komorowski and J. Zytkow, Eds., ed: Springer Berlin / Heidelberg, 1997, pp. 36-46.
- [20] S. M. Weiss and N. Indurkha, "Reduced complexity rule induction," presented at the Proceedings of the 12th international joint conference on Artificial intelligence - Volume 2, Sydney, New South Wales, Australia, 1991.
- [21] X. Wu, V. Kumar, J. Ross Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. McLachlan, A. Ng, B. Liu, P. Yu, Z.-H. Zhou, M. Steinbach, D. Hand, and D. Steinberg, "Top 10 algorithms in data mining," Knowledge and Information Systems, vol. 14, pp. 1-37, 2008.
- [22] L. A. Zadeh, "Fuzzy Sets," Information and Control, vol. 8, pp. 338–353, 1965.
- [23] Frank, A. & Asuncion, A. (2010). UCI Machine Learning Repository [<http://archive.ics.uci.edu/ml>]. Irvine, CA: University of California, School of Information and Computer Science.
- [24] "Data Mining Curriculum". ACM SIGKDD. 2006-04-30. Retrieved 2011-10-28.