

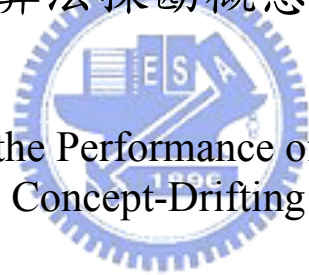
國立交通大學

資訊科學與工程研究所

博士論文

一個提昇分類演算法探勘概念漂移資料效能之研究

A Study to Improve the Performance of Classification for Mining  
Concept-Drifting Data



研究生：蔡政容  
指導教授：楊維邦 教授  
                  李建億 教授

中華民國九十七年一月

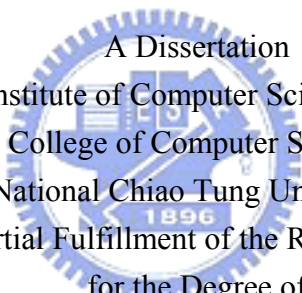
# 一個提昇分類演算法探勘概念漂移資料效能之研究

A Study to Improve the Performance of Classification for Mining  
Concept-Drifting Data

研究生： 蔡政容  
指導教授： 楊維邦 博士  
              李建億 博士

Student: Cheng-Jung Tsai  
Advisors: Dr. Wei-Pang Yang  
           Dr. Chien-I Lee

國立交通大學  
資訊科學與工程研究所  
博士論文



A Dissertation  
Submitted to Institute of Computer Science and Engineering  
College of Computer Science  
National Chiao Tung University  
in partial Fulfillment of the Requirements  
for the Degree of  
Doctor of Philosophy  
in

Computer Science

January 2008

Hsinchu, Taiwan, Republic of China

中華民國九十七年一月

# 一個提昇分類演算法探勘概念漂移資料效能之研究

學生： 蔡政容

指導教授： 楊維邦 博士  
李建億 博士

國立交通大學資訊科學與工程研究所 博士班

## 摘 要

隨著資料數位化技術的快速發展，近年來資料探勘技術已被廣泛地運用，以從龐雜的資料中萃取出有用的資訊。資料探勘可細分為數個研究領域如關聯法則、分類、群集…等，其中分類技術對於預測未知的資料扮演著十分重要的角色並已成功的運用到現實生活中。分類的研究領域包含了許多重要的研究議題，如可調適性、不平衡資料集、合議分類器、關聯資料庫探勘、隱私權…等。因為目前許多日常生活中的資料是以接連不斷的資料區塊之方式呈現，學者愈來愈重視資料串流的探勘。已提出之探勘資料串流的方法，大部份都假設資料區塊是呈現平穩分佈。然而此假設是不合理的，因為資料的概念可能會隨著時間而改變。此種資料概念隨著時間遞延而改變的情形，便稱之為概念漂移。概念漂移的發生，使得利用資料串流建構分類器的工作變得更複雜。

目前已提出用來探勘具有概念漂移之資料串流的方法，共同的缺點之一是當資料串流十分穩定時會耗損許多不必要的系統成本：包括重建分類器的計算成本或紀錄具有類似區塊的記錄成本。此外，這些方法對於含有概念漂移樣本之資料串流皆不具有高敏銳度：這些方法只有在產生漂移的樣本達到一臨界值時才會發現概念漂移的產生；此外，這些方法皆無法解決雙向漂移的問題。對於某些即時的應用如電腦病毒偵測，一個具有高敏銳度的演算法是很重要的。因此，本論文提出 SCRIPT 演算法來處理具有概念漂移

的資料串流。SCRIPT 演算法對於具有概念漂移的資料串流可以建立更準確且更具敏感度的分類器。

此外，目前已提出能處理概念漂移資料的演算法，全都只著重在正確地更新原有的分類器以維持預測的準確性。但對使用者而言，其可能對於引起概念漂移的規則更感興趣。探勘概念漂移規則是一個十分有趣且實用的議題。例如：醫生會想瞭解引起疾病變化的主因、學者會想要知道氣候轉變的規則、或是決策者會想找出顧客購物習慣改變的因素...等。然而此一問題在過去卻被忽略掉。為瞭解決這個問題，本論文提出 CDR-Tree 演算法來探勘出造成概念漂移的規則。CDR-Tree 演算法的另一個特點在於，當使用者需要檢視或運用分類器時，其能經由簡單的抽取程式快速且正確地產生資料的分類器而不須重新建構。

最後，為了減少 CDR-Tree 的建構時間並簡化其所產生的概念漂移規則，本論文針對離散化的問題進行探討。離散化是一個將數值屬性切割成多個有限區間的技術。離散化技術對於須處理大量資料和只能處理類別屬性的學習演算法扮演著十分重要的角色。過去的實驗結果顯示離散化技術不但能加速學習演算法，同時也能維持甚至改進學習演算法所建構出之分類器的準確度。然而，目前已知的離散化演算法皆只能處理單一屬性值和單一類別的資料，並無法離散 CDR-Tree 所使用的多重屬性值和多重類別的資料。因此，本論文提出了能處理多重屬性值和多重類別資料的離散化演算法 OMMD。

**關鍵字：**資料探勘，分類，決策樹，資料串流探勘，概念漂移，概念漂移規則，離散化，多重屬性值，多重類別。

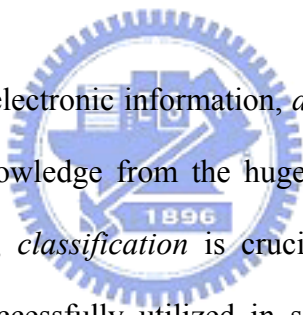
# A Study to Improve the Performance of Classification for Mining Concept-Drifting Data

Student: Cheng-Jung Tsai

Advisors: Dr. Wei-Pang Yang  
Dr. Chien-I Lee

Institute of Computer Science and Engineering  
National Chiao Tung University

## ABSTRACT

The logo of National Chiao Tung University is a circular emblem with a blue border. Inside the circle, there is a stylized representation of a building or a gear-like structure. At the bottom of the circle, the year '1896' is inscribed.

With the rapid growth of electronic information, *data mining* has been widely applied to the identification of useful knowledge from the huge bank of extant data. Among several functionalities of data mining, *classification* is crucially important for the predication of unseen data, and has been successfully utilized in several real-world applications. In the research domain of classification, there are several important issues to be addressed; including scalability, imbalanced datasets, ensemble classifiers, multi-relational databases mining, privacy-preservation, and so on. Since many real-world data nowadays come in the form of consecutive data blocks, researchers have focused increasing attention on *data stream mining*. Most proposed approaches to data stream mining assume that data blocks are to be obtained in stationary distributions. This assumption is unreasonable since the distribution underlying the data is likely to change over time. This problem, known as *concept drift*, complicates the task of learning a classification model from data streams.

Proposed solutions to mine concept-drifting data streams consume some unnecessary system resources, including computational costs to rebuild the decision tree or storage costs to

record similar data blocks. In addition, they generally do not sufficiently account for the problem of concept drift: a) the proposed solutions can detect the changes until the number of drifting instances reaches a threshold to cause obvious difference in accuracy or information gain or gini index; b) the proposed solutions would make a wrong estimation when there are *two-way drifts*. For some real-time applications such as computer virus detection, a sensitive approach to detecting drifting concepts is very important. In this dissertation, we propose a sensitive concept drift probing decision tree algorithm named SCRIPT to accurately and sensitively mine concept-drifting data streams.

Then, we address proposed solutions for mining concept-drifting data that focus only on accurately updating the classification model. They are deficient in that they are unable to provide users with a satisfactory solution to concept drift; the rules of which may be of considerable interest to some users. Mining *concept-drifting rules* is both of great academic interest and high practical applicability. Some examples include: doctors desiring to know the root causes behind variations in the causes and development of disease, scholars longing for the rules underlying weather transition, and sellers wanting to discern the reasons why consumers' shopping habits change. However, this issue was ignored in the past. We propose a concept drift rule mining tree algorithm named CDR-Tree to accurately elucidate the underlying rules governing concept drift. Another important characteristic of CDR-Tree is that it can efficiently and accurately generate classification models via a simple extraction procedure instead of building them from scratch should classification models be required.

Finally, in order to speed up the building procedure and simplify the rules produced by CDR-Tree, we focus our attention on discretization techniques. *Discretization* is a technique for reducing the number of values for a given continuous attribute by dividing the range of the attribute into a finite set of adjacent intervals. It is an important data preprocessing technique for data mining algorithms which are highly sensitive to the size of data or are only able to handle categorical attributes. Empirical evaluations show that the discretization technique has

the potential to speed up the learning process while retaining or even improving predictive accuracy of learning algorithms. Unfortunately, all of proposed discretization algorithms are designed to handle only single-valued and single-labeled data and are infeasible to discretize the *multi-valued* and *multi-labeled* data used in CDR-Tree. We propose a new discretization approach named OMMD (ordered multi-valued and multi-labeled discretization algorithm) as the solution to discretize the input data of CDR-Tree.

**Keywords:** Data mining, classification, decision tree, data stream mining, concept drift, concept-drifting rule, discretization, multi-valued, multi-labeled.



# Contents

<b>Abstract in Chinese .....</b>	<b>I</b>
<b>Abstract in English .....</b>	<b>III</b>
<b>Contents .....</b>	<b>VI</b>
<b>List of Tables .....</b>	<b>IX</b>
<b>List of Figures .....</b>	<b>X</b>
<b>1 Introduction .....</b>	<b>1</b>
1.1 Concept Drift in Data Stream Mining .....	1
1.2 The Rules of Concept Drift .....	4
1.3 Multi-valued and Multi-labeled Discretization .....	6
1.4 Synopsis of This Dissertation .....	9
<b>2 Background and Related Work .....</b>	<b>10</b>
2.1 Decision Tree .....	10
2.2 Data Stream Mining and Concept Drift .....	13
2.3 Discretization Techniques .....	16
2.3.1 Proposed Discretization Approaches .....	16
2.3.2 CAIM Discretization Algorithm .....	19
2.4 UCI Database and IBM Data Generator .....	21
<b>3 Sensitive Concept Drift Probing Decision Tree Algorithm .....</b>	<b>23</b>
3.1 One-way Drift and Two-way Drift .....	23



3.2	Sensitive Concept Drift Probing Decision Tree Algorithm .....	26
3.2.1	Class Distribution on Attribute Value .....	26
3.2.2	Correction Mechanism in SCRIPT .....	32
3.2.3	The Pseudocode and Computational Complexity of SCRIPT .....	36
3.3	Experiment and Analysis .....	40
3.3.1	Experimental Datasets .....	40
3.3.2	The Comparison of Accuracy .....	41
3.3.3	The Comparison of Execution Time .....	44
<b>4</b>	<b>Concept Drift Rule Mining Tree Algorithm .....</b>	<b>47</b>
4.1	The Rules of Concept Drift .....	47
4.2	Concept Drift Rule Mining Tree Algorithm .....	51
4.2.1	Building a CDR-Tree .....	51
4.2.2	Extracting Decision Trees from a CDR-Tree .....	56
4.3	Experiment and Analysis .....	62
4.3.1	The Analysis of CDR-Tree .....	63
4.3.2	The Comparison of Accuracy between E-CDR-Tree and C5.0 .....	65
4.3.3	The Comparison of Execution Time among CDR-Tree, E-CDR-Tree, and C5.0	66
<b>5</b>	<b>Ordered Multi-valued and Multi-labeled Discretization Algorithm.....</b>	<b>68</b>
5.1	Problem Formulation .....	68
5.1.1	Multi-valued and Multi-labeled Discretization .....	68
5.1.2	Ordered versus Non-ordered data .....	69
5.2	Ordered Multi-valued and Multi-labeled Discretization Algorithm.....	72
5.2.1	A New Discretization Metric .....	72
5.2.2	Discretizing Continuous Attributes and the Computational Complexity .....	75

5.2.3	Discretize Categorical Attributes and the Computational Complexity .....	81
5.3	Experiment and Analysis .....	84
5.3.1	The Comparison of Discretization Scheme .....	84
5.3.2	The Performance Evaluation of OMMD .....	90
<b>6</b>	<b>Conclusions and Future Work.....</b>	<b>92</b>
6.1	Conclusions .....	92
6.2	Future Work .....	94
	<b>Bibliography.....</b>	<b>95</b>



# List of Tables

Table 2.1 The quanta matrix for attribute $A$ and discretization scheme $D$ .....	19
Table 2.2 The summary of nine basic attributes in IBM data generator.....	22
Table 3.1 The class label distribution on an attribute $A_i$ .....	28
Table 3.2 Two data sets $D$ and $D'$ without the occurrence of concept drift .....	30
Table 3.3 Two data sets $D$ and $D'$ with the occurrence of concept drift.....	32
Table 3.4 The Comparisons of system cost among SCRIPT, DNW, and CVFDT .....	39
Table 3.5 The summary of three selected experimental UCI datasets.....	41
Table 4.1 The patients' diagnostic data.....	49
Table 4.2 The new coming diagnostic data from the same patients .....	49
Table 4.3 The integrated data of Table 4.1 and Table 4.2 .....	52
Table 5.1 A non-ordered multi-valued and multi-labeled dataset.....	71
Table 5.2 An ordered multi-valued and multi-labeled dataset.....	71
Table 5.3 Age dataset.....	73
Table 5.4 The discretization scheme of age dataset generated by CAIM.....	73
Table 5.5 Two datasets with equal $caim$ values but different data distribution.....	74
Table 5.6 Two datasets with equal $caim$ values but different $cair$ values .....	74
Table 5.7 The discretization scheme of Table 5.2 after iterative splits.....	75
Table 5.8 The discretization scheme of Table 5.2 after merging .....	77
Table 5.9 The summary of thirteen UCI real datasets .....	85
Table 5.10 The experimental results of CDR-Tree with/without NOMMD.....	91

# List of Figures

Figure 2.1 A typical decision tree.....	11
Figure 3.1 A data stream with the occurrence of concept drift. ....	24
Figure 3.2. Two data blocks with the occurrence of concept drift: (a) original data block and the corresponding sub-tree; (b) new data block and the corresponding sub-tree..	33
Figure 3.3 The illustrations of the correction mechanism in SCRIPT when concept drift occurs. ....	35
Figure 3.4 The comparison of accuracy on dataset ‘satimage’.....	42
Figure 3.5 The comparison of accuracy on dataset ‘thy’.....	43
Figure 3.6 The comparison of accuracy on dataset ‘spambase’.....	43
Figure 3.7 The comparison of execution time on dataset ‘satimage’.....	45
Figure 3.8 The comparison of execution time on dataset ‘thy’.....	45
Figure 3.9 The comparison of execution time on dataset ‘spambase’.....	46
Figure 4.1 The decision tree built using Table 4.1.....	50
Figure 4.2 The decision tree built using Table 4.2.....	50
Figure 4.3 The CDR-Tree built using Table 4.3.....	53
Figure 4.4 Illustrations of the extraction strategy in CDR-Tree algorithm.....	57
Figure 4.5 The extracted decision trees from Fig. 4.3: (a) the model of Table 4.1 without implementing Step 5; (b) the model of Table 4.1 with the implementation of Step 5; (c) the model of Table 4.2 without implementing Step 5; (d) the model of Table 4.2 with the implementation of Step 5. ....	61
Figure 4.6 The accuracy of CDR-Trees under five different drifting ratios.....	64

Figure 4.7 The accuracy of concept-drifting rules produced by CDR-Trees. .... 64

Figure 4.8 The comparison of accuracy between E-CDR-Tree and C5.0 using four datasets with 10% drifting ratio. .... 65

Figure 4.9 The comparison of execution time among CDR-Tree, E-CDR-Tree, and C5.0 by using datasets D(43). .... 67

Figure 5.1 The comparison of CACC against the other discretization methods with the Holm’s post-hoc tests ( $\alpha = 0.05$ ): (a) and (b) *cair* value; (c) number of intervals; (d) and (e) execution time. .... 88

Figure 5.2 The comparison of C5.0 performance on CACC against C5.0 performance on the other discretization methods with the Holm’s post-hoc test ( $\alpha = 0.05$ ): (a) and (b) accuracy; (c) and (d) number of rules; (e) and (f) execution time. .... 90



# Chapter 1

## Introduction

With the rapid development and large-scale distribution of electronic data, extracting useful information from many numerous and jumbled sources has become an important goal for many scholars. Data Mining [30][57], an important technique for extracting information from massive data repositories, has been proposed to solve this problem. Among the several functionalities of data mining, classification is crucially important and has been applied successfully to several areas [29][78]. In the research domain of classification, several important issues, including scalability [28], imbalanced datasets [34][47], ensemble classifiers [9][48], incremental learning [27][51][52][66][71][72], multi-relational databases mining [47], and so on, have been widely studied. Since current real-world data may come in the form of consecutive data blocks [16][17], researchers have focused ever increasing attention on *data stream mining*. Relevant applications include e-mail sorting [16], calendar scheduling [5], and computer intrusion detection [53].

### 1.1 Concept Drift in Data Stream Mining

Most proposed approaches to data stream mining have assumed that data blocks exist in *stationary distributions*. Such an assumption is unreasonable since the *concept* (also called

*class label*) of an instance might change over time. That is, an instance with concept “yes” in the current data block may be with concept “no” in the next one. Such a change of concept is known as *concept drift* [31][37][39][45][67][73], *changing concepts* [38], or *time-varying concepts* [42].

Window-based approaches [32][38][46][51][75] are the common solutions to concept drift in a data stream. They use a fixed or adaptive window [33] to select appropriate training data for different time points. Weighting-based methods [40][41] and ensemble classifiers [22][68] have also been introduced to handle the concept-drifting problem. However, while the concept is stationary, the methods mentioned above consume some unnecessary system resources, including computational costs to rebuild the decision tree or storage costs to record similar data blocks. Moreover, they generally do not sufficiently account for the problem of concept drift: a) the proposed solutions can detect the changes until the number of drifting instances reaches a threshold to cause obvious difference in accuracy or information gain or gini index; b) the proposed solutions would make a wrong estimation when there are *two-way drifts* (the formal definition of two-way drift will be given in Chapter 3). For some real-time applications such as fraudulent credit card transactions or computer virus detection, a sensitive approach to detecting drifting concepts is very important since it can reduce the possibility of serious damage. Finally, it is interesting to note that drifting instances must gather in specific areas of the dimensional space of attributes, otherwise they should be referred to as noise data. These foregoing observations motivate us to propose a more efficient and sensitive classification approach to mine drifting concepts in data streams.

Popular techniques that have been developed for classification include: *bayesian classification*, *neural networks*, *genetic algorithms*, and *decision trees* [28][54][62][65]. Among them, the decision tree is a popular tool for following reasons [64]: a) it is more easily interpreted by humans than neural networks or bayesian-based approaches,; b) it is more efficient for large quantities of training data than neural networks which require much

time on thousands of iterations; c) it does not require a domain knowledge or prior knowledge; and, d) it displays good classification accuracy as compared to other techniques. Due to the above yields, a decision tree-based approach named sensitive concept drift probing decision tree algorithm (SCRIPT) is proposed in this dissertation to classify concept-drifting data streams. The main benefits of SCRIPT are: a) it can avoid unnecessary system costs for stable data streams; b) it can efficiently rebuild classifiers while data streams are instable; c) it is more suitable for the applications in which sensitive detection of concept drift is required.





## 1.2 The Rules of Concept Drift

Although SCRIPT can sensitively and efficiently handle the concept-drifting problem in data streams, as with most proposed approaches regarding concept drift, it focuses on updating the classification model to accurately predict new incoming data. Relevant to users may be the concept-drifting rules. For example, doctors desiring to know the main causes of disease variation, scholars longing for the rules of weather transition, and sellers wanting to discern the reasons why consumer shopping habits change. Below is a simple concept-drifting rule, elucidated by analysis of customers.

If (marry = ‘no    yes’) and (baby = ‘no    yes’) and (salary = ‘30000    40000’)  
then (buys = ‘digital camera    video camera’).

This rule means that a customer tends to buy a video camera instead of a digital camera if he: gets married, has a newborn baby, and has an increase in salary

Mining concept-drifting rules is both of great academic interest and high practical applicability. However, this issue was ignored in the past. In order to accurately discover concept-drifting rules, we propose the concept drift rule mining tree algorithm, called CDR-Tree. The main principle behind CDR-Tree is that it integrates the data from two data blocks into one dataset and then uses this integrated dataset as training data to build a decision tree. This idea is simple but novel; to our knowledge, we are the first one to address the problem of mining concept-drifting rules. The main benefits of CDR-Tree are: a) CDR-Tree can accurately mine the rules of concept drift; b) if classification models are required, CDR-Tree can efficiently and accurately generate them via a simple extraction procedure rather than building them from scratch.

Note that, what we address is different from the *emerging pattern*; which is the itemset whose support increases significantly in association rule mining [21][74]. As claimed in [26],

classification and association rule discovery are fundamentally different mining tasks. The former can be considered a nondeterministic task, which is unavoidable given the fact that it involves prediction; while the later can be considered a deterministic task which does not involve prediction in the same sense as the classification task does. Most importantly, the goal of emerging patterns is to find a group of instances which have the same itemset but significant changes of class labels are occurred within these instances, while concept-drifting rules attempt to reveal the reasons why concept drifts.



### 1.3 Multi-valued and Multi-labeled Discretization

Nevertheless, a problem remains with CDR-Tree; that is, if there are mass drifting instances, CDR-Tree will require much more learning time and will generate a lot of rules. In order to speed up the building procedure and simplify the rules produced by CDR-Tree, we have focused our attention on discretization techniques. *Discretization* [19][23][24][61] is a technique for reducing the number of values for a given continuous attribute by dividing the range of the attribute into a finite set of adjacent intervals. It is an important data preprocessing technique for data mining algorithms which are highly sensitive to the size of data or are only able to handle categorical attributes [13][14][15][35][58]. Given a continuous attribute  $A$ , discretization algorithms discretize this attribute into  $n$  discrete intervals  $\{[d_0, d_1], (d_1, d_2], \dots, (d_{n-1}, d_n]\}$ , where  $d_0$  is the minimal value and  $d_n$  is the maximal value of attribute  $A$ . The discrete result  $\{[d_0, d_1], (d_1, d_2], \dots, (d_{n-1}, d_n]\}$  is called a *discretization scheme*  $D$  on attribute  $A$ . A good discretization scheme should maintain the high interdependency between the discrete attribute and the class labels so as to carefully avoid changing the distribution of the original data [56][69]. Another merit of discretization algorithms is that they can produce a concise summarization of continuous attributes to help experts and users understand the data more easily.

Over the years, many discretization algorithms have been proposed. Empirical evaluations show that the discretization technique has the potential to speed up the learning process while retaining or even improving predictive accuracy of learning algorithms [49]. In [49], the taxonomy of proposed discretization algorithms is proposed, with five axes: *supervised* versus *unsupervised*, *static* versus *dynamic*, *global* versus *local*, *merging* versus *splitting*, and *direct* versus *incremental*. However, all of these proposed discretization algorithms are designed to handle only single-valued and single-labeled data and are

infeasible to discretize *multi-valued* and *multi-labeled* one. In real-world applications, some available datasets including the data used in CDR-Tree are multi-valued and multi-labeled [8][10]. If data are multi-valued and multi-labeled, a record can have multiple values of one attribute, and can belong to multiple class labels. A simple example is that Tomatoes can be green or red (two-valued) and belongs to both fruit and vegetable (two-labels). Another common example is that users may have  $l$  credit cards ( $l$ -labels) and therefore  $v$  kinds of consumption ( $v$ -values). In recent years, researchers have focused their attention on developing classifiers to mine multi-valued and multi-labeled data. For example, MMC (multi-valued and multi-labeled classifier) [10] and MMDT (multi-valued and multi-labeled decision tree) [8] have been proposed to classify multi-valued and multi-labeled data. Unfortunately, there is no discretization algorithm devoted to handling such data.

While extending traditional single-valued and single-labeled discretization methods to handle multi-valued and multi-labeled data, there are some problems that must be addressed. First, traditional discretization approaches discretize only continuous attributes. However, for categorical attributes which contain  $c$  distinct values, the size of the value domain becomes  $v^c-1$  in a  $v$ -valued dataset, and therefore heavily burdens the learning process. In addition, multi-valued and multi-labeled datasets can be *ordered* or *non-ordered* and require different discretization strategies. In an ordered multi-valued and multi-labeled dataset, each attribute value corresponds to a class label. On the contrary, a non-ordered multi-valued and multi-labeled dataset contains records which may have different numbers of values and different numbers of labels. The formal definitions of ordered and non-ordered multi-valued and multi-labeled datasets can be found in Chapter 5. Since our main goal is to design a discretization algorithm to discretize the input data of CDR-Tree, we focus on ordered multi-valued and multi-labeled data in this dissertation and propose an ordered multi-valued and multi-labeled discretization algorithm (OMMD). OMMD uses a new discretization metric and the simulated annealing search approach to generate discretization schemes. The new

discretization metric is inspired by the statistical *contingency coefficient*. OMMD also integrates the idea of splitting and merging discretization techniques to discretize ordered multi-valued and multi-labeled data. Note that, the problem we address here is different from the issue of *multivariate discretization* [3][7][20][25], which considers the interdependent relationship between attributes to discretize continuous attributes.



## 1.4 Synopsis of This Dissertation

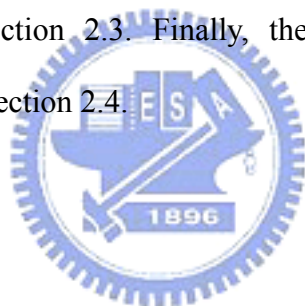
The rest of the dissertation is organized as follows. In Chapter 2, we briefly review related research, including: the introduction of the typical decision tree algorithm, the problem of concept drift in data stream mining, and previous discretization algorithms. In Chapter 3, the problem of mining concept-drifting data and some formal definitions are first introduced. We then elucidate SCRIPT and evaluate its performances. In Chapter 4, we first use an example to introduce the concept-drifting rules to enhance reader understanding of this problem more clearly. Then, CDR-Tree is detailed and evaluated. In Chapter 5, we give some formal definitions of the problem of discretizing multi-valued and multi-labeled data. The details of our new discretization metric and OMMD, the empirical evaluations of OMMD are then presented. Finally, the conclusions of this dissertation are made and some open problems are described in Chapter 6.



# Chapter 2

## Background and Related Work

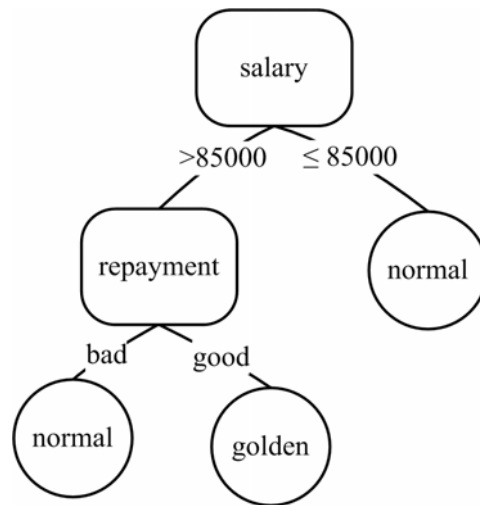
In this chapter, we provide a brief survey of the background and related work. First of all, the typical decision tree algorithm is introduced in Section 2.1. In Section 2.2, we review the methods which are proposed to mine a concept-drifting data stream. Previous discretization algorithms are studied in Section 2.3. Finally, the experimental datasets used in this dissertation are introduced in Section 2.4.



### 2.1 Decision Tree

A typical decision tree [62][63] is a flow-chart-like tree structure, which is constructed by a recursive divide-and-conquer algorithm. In a decision tree, each *internal node* denotes a test on an attributes, each branch represents an outcome of the test, and each *leaf node* has an associated *target class (class labels)*. The top-most node in a tree is called *root* and each path from the root to a leaf node represent a *rule*. A typical decision *tree* is shown in Figure 2.1. To classify an unseen example, beginning with the root node, successive internal nodes are visited until this example reaches a leaf node. The class of this leaf node is then assigned to this example as a prediction. For instance, the decision in Figure 2.1 will approve a golden credit card application if the applicant has a salary higher than 85000 and his/her repayment

record is good.



**Figure 2.1** A typical decision tree.

A number of decision tree algorithms, such as ID3 [71], C4.5 [62], CART [6], CHAID [6], SLIQ [54], SPRINT [65], RainForest [28], and PUBLIC [64] have been proposed over the years. Most of them are composed of two phases: the *building phase* and the *pruning phase*. Besides, before inducing the decision tree, the original dataset is usually to be divided into *training data* and *testing data*. In the building phase, the training data is recursively partitioned by a *splitting function* until all the examples in a node is pure (i.e. all examples in this node have the same class labels) or can not be further partitioned (i.e. all examples in this node contain the same attribute value but different target class). Several famous splitting functions, such as *information gain* and *gain ratio* [30], had widely been used in past. After a decision tree is built, many of the branches will reflect anomalies in the training data due to *noise data* or *outlier*. To prevent such an *overfitting* problem [30], decision tree would prune its model to remove the least reliable branches, and generally resulting in faster classification and an improvement in the ability of the tree to correctly classify unseen data. There are two common approaches to prune tree: *pre-pruning* [64] and *post-pruning* [55]. Pre-pruning



approach halt its tree building early by deciding no further partitioning the subset of training data at a given node, while post-pruning removes branches from a fully grown tree by use of testing data.



## 2.2 Data Stream Mining and Concept Drift

Most proposed algorithms of data stream mining [27][51][66][71][72] assumed data blocks come under stationary distributions, but in reality, the concept of an instance might vary. While the concept of data starts drifting, the classification model constructed by using old data is unsuitable for the new one. Thus, it is imperative to revise the old classification model or re-build a new one. VFDT (Very Fast Decision Tree Learner) [18] has been proposed to solve the scalable problem when learning from very large data stream. It starts with a single leaf and starts collecting training examples from a data stream. When VFDT gets enough data to know, with high confidence that it knows which attribute is the best to partition the data with, it turns the leaf into an internal node and goes on splitting it. However, as most incremental learning methods, it assumes that the data is a random sample drawn from a stationary distribution and is inappropriate for the mining of concept drifting data such as credit card approval and fraud detection.

Window-based approaches [32][39][46][51][75] are the common solutions for the problem of concept drift on data stream. They use a fixed or sliding window [33] to select appropriate training data for different time points. CVFDT [32] (concept-adapting very fast decision tree learner), which is formerly VFDT, is a representative window-based approach for mining concept drift on data stream. It solves the concept-drifting problem by maintaining only fixed amount of data within the window. CVFDT keeps its learned tree up-to-date with this window by monitoring the quality of its old decisions as data move into and out of the window. In particular, whenever a new instance is read it is added to the statistics at all the nodes in the tree that it passes through, the last example in the window is forgotten from every node where it had previously had an effect, and the validity of all statistical tests are checked. If CVFDT detects a change, it starts growing an alternate tree in parallel which is rooted at the

newly-invalidated node. When the alternate is more accurate on new data than the original, the original will be replaced by the alternate tree.

WAH (window addition huristic) [75] and DNW (determine new window size) [38] [39] are also representative window-based algorithms, however, they use sliding window. WAH take the actual condition of decision tree into account to dynamically adjust the window size. After new data stream join, the doubt for concept drift will reduce the size of windows by 20%. Contrarily, when data are stable, a unit of window is deleted to avoid maintaining too many unused data. When the concept seems to be stable, the original window size is maintained. If none of the conditions mentioned above are valid, it means that more information will be needed to build classifiers. As a result, old data will not be left out of the window and new data will also be added in it. Although WAH can solve the problem of concept drift according to actual conditions, but it is suitable only for small databases. DNW deals with the learning of training data by way of data block, which is suitable for data stream environment. DNW has a similar way of learning to WAH; however, they are different in condition and way of assessment. DNW builds a classifier for each block, and compares the three parameters: accuracy, recall, and precision for classifiers on the current blocks with the ones for the previous classifiers. Weighting-based [40][41] and ensemble classifier [22][68] were also introduced to handle the concept-drifting problem on data stream. Weighting-based approach provides each example with a weight according to their age and utility for the classification task. Ensemble classifier built separate sub-classifiers and then combines the prediction of each sub-classifier to classify the unseen data. The main disadvantage of an ensemble classifier is the huge system cost caused by the building and maintenance of all sub-classifiers.

While the concept is stable, the methods mentioned above would spend unnecessary system cost, including computational cost to build or rebuild a decision tree or storage cost to record similar data streams. Moreover, when concept is drifting, they generally are not

sensitive enough to the concept drift problem. That is, if the proportion of drifting instances to all instances in a data block is small, the proposed solutions can detect the changes until the number of drifting instances reaches a threshold to cause obvious difference in accuracy or information gain. For some applications such as fraudulent credit card transactions, the sensitivity to detect drifting concepts would be very important. In an ensemble classifier, the fraudulent transactions might be ignored due to the predictions of old sub-classifiers. For weighting-based approaches, even giving a high weight to the transactions in the new data block, they might also make a wrong prediction since the influence of old transactions. Fixed window-based approaches have a similar problem to that in weighting-based approaches, and sliding window-based approaches would also disregard such changes since these drifting transactions would not cause obvious variance of accuracy or information gain.



## 2.3 Discretization Techniques

In this subsection, we review some proposed discretization algorithms and the detail of the-state-of-art CAIM discretization algorithm.

### 2.3.1 Proposed Discretization Approaches

Attributes can be divided into *continuous attribute* and *categorical attribute*. Among the proposed classification algorithms, some of them such as AQ [58], CLIP [14][13], and CN2 [15] can handle only categorical attributes, and some of them can handle continuous attributes but would perform better on categorical attributes [76]. To solve this problem, a lot of discretization algorithms have been proposed. Given a continuous attribute  $A$ , discretization algorithms would discretize this attribute into  $n$  discrete intervals  $\{[d_0, d_1], (d_1, d_2], \dots, (d_{n-1}, d_n]\}$ , where  $d_0$  is the minimal value and  $d_n$  is the maximal value of attribute  $A$ . The discrete result  $\{[d_0, d_1], (d_1, d_2], \dots, (d_{n-1}, d_n]\}$  is called a *discretization scheme*  $D$  on attribute  $A$ . Discretization is usually performed prior to the learning process and can be broken into two steps. The first step is to decide the number of discrete intervals, and most discretization algorithms require the user to specify the number of intervals [11]. The second step is to find the width (also called the boundary) of each interval. A good discretization scheme should keep the high interdependency between the discrete attribute and the class labels to carefully avoid changing the distribution of the original data [56][69].

The literature on discretization is vast. Liu, Hussain and Dash [49] stated that discretization approaches have been proposed along five lines: supervised versus unsupervised, static versus dynamic, global versus local, splitting versus merging, and direct

versus incremental. Below we give a brief introduction of the five lines.

1. Supervised methods discretize attributes with the consideration of class information, while unsupervised methods do not.
2. Dynamic methods [4][77] consider the interdependence among the features attributes and discretize continuous attributes when a classifier is being built. On the contrary, the static methods consider attributes in an isolated way and the discretization is completed prior to the learning task.
3. Global methods, which use total instances to generate the discretization scheme, are usually associated with static methods. On the contrary, local methods are usually associated with dynamic approaches in which only parts of instances are used for discretization.
4. Merging methods start with the complete list of all continuous values of the attribute as cut-points and remove some of them by merging intervals in each step. Splitting methods start with an empty list of cut-points and add new ones in each step. The computational complexity of merging methods is usually larger than splitting ones.
5. Direct methods, such as Equal-Width and Equal-Frequency [12], require users to decide on the number of intervals  $k$  and then discretize the continuous attributes into  $k$  intervals simultaneously. On the other hand, incremental methods begin with a simple discretization scheme and pass through a refinement process although some of them may require a stopping criterion to terminate the discretization.

Take the two simplest discretization algorithms Equal Width and Equal Frequency [12] as examples; both of them are unsupervised, static, global, splitting and direct method. In the follows, we review some typical discretization algorithms by following the line of splitting versus merging.

Famous splitting methods include Equal Width and Equal Frequency [12], Paterson-Niblett [60], maximum entropy [76], CADD (Class-Attribute Dependent Discretizer

algorithm) [11], IEM (Information Entropy Maximization) [23], CAIM (Class-attribute Interdependence Maximization) [44], and FCAIM (Fast Class-attribute Interdependence Maximization) [43]. Experiments showed that FCAIM and CAIM are superior to the other splitting discretization algorithms since its discretization schemes can generally maintain the highest interdependence between class labels and discretized attributes, result to the least number of generated rules, and attain the highest classification accuracy [43] [44]. FCAIM is the extension of CAIM. The main framework, including the discretization criterion and the stopping criterion, as well as the time complexity between CAIM and F-CAIM are all the same. The only difference is the initialization of the boundary point in two algorithms. Compared to CAIM, F-CAIM was faster and had a similar C5.0 accuracy.

A common characteristic of the merging methods is in the use of the significant test to check if two adjacent intervals should be merged. ChiMerge [36] is the most typical merging algorithm. In addition to the problem of high computational complexity, the other main drawback of ChiMerge is that users have to provide several parameters during the application of this algorithm that include the significance level as well as the maximal and minimal intervals. Hence, Chi2 [50] was proposed based on the ChiMerge. Chi2 improved ChiMerge by automatically calculating the value of the significance level. However, Chi2 still requires the users to provide an inconsistency rate to stop the merging procedure and does not consider the freedom which would have an important impact on discretization schemes. Thereafter, Modified Chi2 [70] takes the freedom into account and replaces the inconsistency checking in Chi2 by the quality of approximation after each step of discretization. Such a mechanism makes Modified Chi2 a completely automated method to attain a better predictive accuracy than Chi2. After Modified Chi2, Extended Chi2 [69] considers that the class labels of instances often overlap in the real world. Extended Chi2 determines the predefined misclassification rate from the data itself and considers the effect of variance in two adjacent intervals. With these modifications, Extended Chi2 can handle an uncertain dataset.

Experiments on these merging approaches by using C5.0 show that the Extended Chi2 outperformed the other bottom-up discretization algorithms since its discretization scheme, on the average, can reach the highest accuracy [69].

### 2.3.2 CAIM Discretization Algorithm

CAIM is the newest splitting discretization algorithm. In comparison with other splitting discretization algorithms, experiments showed that on the average, CAIM can generate a better discretization scheme. These experiments also showed that a classification algorithm, which uses CAIM as a preprocessor to discretize the training data, can on the average, produce the least number of rules and reach the highest classification accuracy [44]. Given the two-dimensional *quanta matrix* (also called a *contingency table*) in Table 2.1, where  $n_{ir}$  denotes number of records belonging to the  $i$ th class label that are within interval  $(v_{r-1}, v_r]$ ,  $L$  is total number of class labels,  $N_{i+}$  is number of records belonging to the  $i$ th class,  $N_{+r}$  is number of records that are within interval  $(v_{r-1}, v_r]$  and  $I$  is number of intervals, CAIM defines the interdependency between the class labels and the discretization scheme of a continuous attribute  $A$  as

$$caim = (\sum(max_r^2/N_{+r})) / I, \quad (2.1)$$

where  $max_r$  is the maximum value among all  $v_{ir}$  values. The larger the value of  $caim$  is, the better the generated discretization scheme  $D$  will be.

**Table 2.1** The quanta matrix for attribute  $A$  and discretization scheme  $D$

label \ interval	$[v_0, v_1]$	...	$(v_{r-1}, v_r]$	...	$(v_{I-1}, v_I]$	summation
$l_1$	$n_{11}$	...	$n_{1r}$	...	$n_{1I}$	$N_{1+}$
$\vdots$	$\vdots$		$\vdots$		$\vdots$	$\vdots$
$l_i$	$n_{i1}$	...	$n_{ir}$	...	$n_{iI}$	$N_{i+}$



⋮	⋮	⋮	⋮	⋮		
$l_L$	$n_{L1}$	...	$n_{Lr}$	...	$n_{Ln}$	$N_{L+}$
<i>summation</i>	$N_{+1}$	...	$N_{+r}$	...	$N_{+l}$	$N$

CAIM is a progressing discretization algorithm and it does not require users to provide any parameter. For a continuous attribute, CAIM will test all possible cutting points and then generate one in each loop; the loop is stopped until a specific condition is met. For each possible cutting point in each loop, the corresponding *caim* value is computed according to the Formula 2.1, and the one with the highest *caim* value is chosen. Since finding the discretization scheme with the globally optimal *caim* value would require a lot computation cost, CAIM algorithm only finds the local maximum *caim* to generate a sub-optimal discretization scheme. In the experiment, CAIM adopts *cair* criterion [76] as shown in Formula 2.2 to evaluate the quality of a generated scheme. *Cair* criterion is used in CADD algorithm [11]. CADD has several disadvantages, such as it needs a user-specified number of intervals and requires training for selection of a confidence interval. Some experimental results also show that *cair* is not a good discretization formula since it can suffer from the overfitting problem [44]. However, *cair* can effectively represent the interdependency between target class and discretized attributes, and therefore is used to measure a discretization scheme.

$$cair = \frac{\sum_{i=1}^L \sum_{r=1}^I p_{ir} \log_2 \frac{p_{ir}}{p_{i+} p_{+r}}}{\sum_{i=1}^L \sum_{r=1}^I p_{ir} \log_2 \frac{1}{p_{ir}}} , \quad (2.2)$$

where  $p_{ir} = \frac{n_{ir}}{N}$ ,  $p_{i+} = \frac{N_{i+}}{N}$ , and  $p_{+r} = \frac{N_{+r}}{N}$  in Table 2.1.

## 2.4 UCI Database and IBM Data Generator

In this dissertation, we use both real and synthetic datasets to carry out a series of experimental evaluations. The real experimental datasets are selected from UCI database [59] which is a repository of several kinds of datasets. UCI database is widely used by the machine learning community for the empirical analysis of machine learning algorithms.

For artificial experimental datasets, we use IBM data generator [1][2], which was designed by IBM Almaden Research Center and is an open source written by C++ programming language. IBM data generator is a popular tool for researchers to generate artificial data to evaluate the performance of proposed algorithms. One advantage of IBM data generator is that it contains a lot of built-in functions to generate several kinds of datasets, and therefore enable researchers to carry out a series of experimental comparisons. There are nine basic attributes (salary, commission, loan, age, zipcode, h-years, h-value, e-level, and car) and a target attribute in IBM data generator. Among the nine attributes, zipcode, e-level, and car are categorical attributes; and all the others are continuous ones. The number of class labels can be decided by users and is set to 2 as default. The summary of these nine basic attributes are illustrated in Table 2.2. In this dissertation, we modify IBM data generator to generate datasets containing concept-drifting records.

**Table 2.2** The summary of nine basic attributes in IBM data generator

Attribute	Type	Value domain
salary	continuous	20,000 to 150,000
commission	continuous	if Salary $\geq$ 75000, Commission = 0 else uniformly distributed from 10000 to 75000
loan	continuous	0 to 500000
h-year	continuous	1 to 30
h-value	continuous	$0.5k*100000$ to $1.5k*100000$ , where $k \in \{1 \dots 9\}$ depends on zipcode
age	continuous	20 to 80
car	categorical	1 to 20
e-level	categorical	0 to 4
zipcode	categorical	1 to 9



# Chapter 3

## Sensitive Concept Drift Probing Decision Tree Algorithm

In this chapter, we first give some formal discussions of the concept-drifting problem in Section 3.1. In Section 3.2, we introduce our sensitive concept drift probing decision tree algorithm (SCRIPT). The empirical analyses of SCRIPT are presented in Section 3.3.



### 3.1 One-way Drift and Two-way Drift

To make readers easily understand the problem we will address later, in this dissertation we divide the concept drift into *concept stable*, *concept drift* and *concept shift*. We refer to the examples in [73] and modify the figures to illustrate the problem in Figure 3.1. Figure 3.1 represents a two-dimensional data stream and is divided into six successive data blocks according to the arriving time of data. Instances arriving between  $t_i$  and  $t_{i+1}$  form block  $B_i$ , and the separating line in each block stands for the optimum classification boundary in this block. During time  $t_0$  to  $t_1$ , data blocks  $B_0$  and  $B_1$  have similar data distribution. That is, data stream during this period is stable. Thereafter in  $B_2$ , some instances shows concept drift and the optimum boundary changes. This is defined as concept drift. Finally, data blocks  $B_4$  and  $B_5$

have opposite sample distribution and this is defined as concept shift. Obviously, since the sample distributions of the first two blocks  $B_0$  and  $B_1$  are quite close, we can use decision tree  $DT_0$  built by  $B_0$  as the classifier for  $B_1$  to save the computational and recording cost. Meanwhile,  $B_2$  shows slight differences when compared with the sample distribution of  $B_1$  and an efficient approach should make correction according to the original decision tree in stead of rebuilding it.

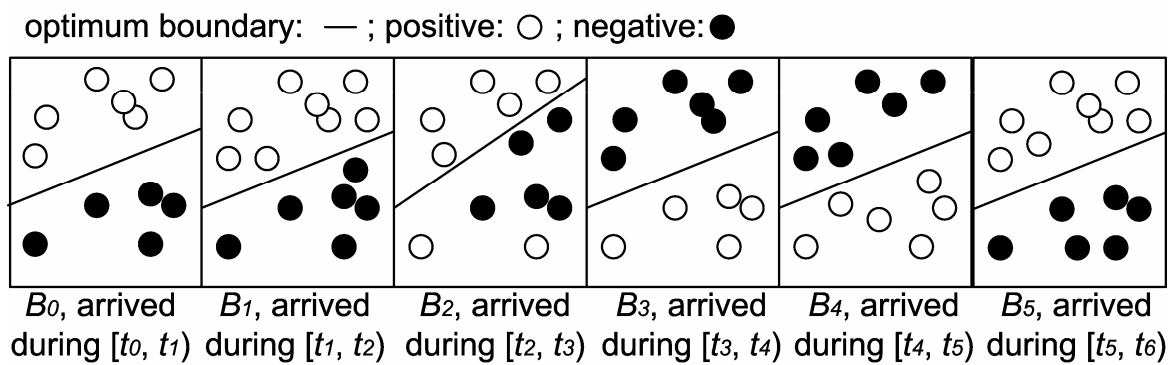


Figure 3.1 A data stream with the occurrence of concept drift.

In Section 2.2, we have showed that past proposed solutions are not sensible enough to the drifting concepts. That is, the proposed solutions can detect the changes until the number of drifting instances reaches a threshold to cause obvious difference in accuracy or information gain or gini index. Here we describe another concept drift problem which would enforce some proposed solutions such as CVFDT and DNW make a wrong prediction. In order to introduce this problem, we subdivide concept drift into *one-way drift* and *two-way drift*. Take Figure 3.1 as the example again, we can find that some negative data in  $B_2$  drift to be positive data in  $B_3$ , known as one-way drift. However, the positive data in  $B_4$  drift to be negative in  $B_5$ , and vice versa, known as two-way drift. We can regard two-way drift as a kind of “local” concept shift if it occurs in the internal or leaf node of a decision tree. If the variation of information gain or gini index is used as the criterion to judge the occurrence of

concept drift, e.g. the difference of information gain adopted in CVFDT, we can detect only one-way drift since the information gain obtained from  $B_4$  would be the same as  $B_5$ . It is worth to note that for the real data, two-way drift might happen. For example, a hacker in turn uses two computers with IP address  $x$  and  $y$  to send attack packages. When an internal node, which is learned from the first data block, splits the packages from  $x$  as safe and that from  $y$  as attack, there might be a contrary result learned from another data block. A similar condition might be found in trash mail protection, image comparison and so on.



## 3.2 Sensitive Concept Drift Probing Decision Tree Algorithm

### 3.2.1 Class Distribution on Attribute Value

Since the proposed solutions to mine concept-drifting data stream check the occurrence of concept drift on the level of instance or attribute, they generally are not sensitive enough. Besides, they are also unable to detect the two-way drift illustrated in Figure 3.1. To solve these problems, SCRIPT probe the changes at a more detailed level, which is called *Class Distribution on Attribute Values* (CDAV) and defined as follows.

**Definition 3.1:** Assuming that a data block contains  $m$  target classes  $c_k$  ( $k = 1, \dots, m$ ),  $n$  attributes  $A_i$  ( $i = 1, \dots, n$ ), and each attribute  $a_i$  having  $v$  attribute values  $a_{ij}$  ( $j = 1, \dots, v$ ), then the distribution of target class  $c_k$  on the attribute value  $a_{ij}$  is defined as a  $CDAV_{ij}$  (*Class Distribution on Attribute Value*).

With Definition 3.1, we can use the chi-square ( $X^2$ ) test to check if there are concept drifts between two data blocks.  $X^2$  test is a statistical measure used to test the hypothesis that two discrete attributes are statistically independent. Applied to the concept-drifting problem, it tests the hypothesis that the class distribution on an attribute value of two data blocks is identical. The formula to computing the  $X^2$  value is

$$X^2 = \frac{(f'_{ijk} - f_{ijk})^2}{f_{ijk}} \quad (3.1)$$

, where  $f_{ijk}$  represents the number of instances having attribute value  $a_{ij}$  and class  $c_k$  in  $D$  and  $f'_{ijk}$  is that in  $D'$ . With Formula 3.1, we can then define the variance of a  $CDAV_{ij}$  in the two data blocks as follows.

**Definition 3.2:** For a given significant level  $\alpha$ , the variance  $CDAV_{D \rightarrow D'}(i, j)$  of the a  $CDAV_{ij}$  between two data blocks  $D$  and  $D'$  in a data stream is defined as

$$CDAV_{D \rightarrow D'}(i, j) = \sum_{k=1}^m \frac{(f'_{ijk} - f_{ijk})^2}{f_{ijk}}. \quad (3.2)$$

**Proposition 3.1:** For the two data blocks  $D$  and  $D'$ , if all  $CDAV_{D \rightarrow D'}(i, j) < \varepsilon$ , then the concept distribution on all attribute value  $a_{ij}$  in the two data blocks show no significant difference, and neither do the accuracy of decision tree built according to  $D$  and  $D'$ , respectively.

**Proof:**

Since  $CDAV_{D \rightarrow D'}(i, j) < \varepsilon$ ,

we can obtain  $f_{ijk} \cong f'_{ijk}$  for target classes  $c_k$  ( $k = 1, \dots, m$ ), attributes  $A_i$  ( $i = 1, \dots, n$ ) and attribute value  $a_{ij}$ .

For attribute  $A_i$ , the Entropy before the splitting is

$$I(A_i) = - \sum_{k=1}^m P_{ik} \log P_{ik},$$

where  $P_{ik} = f_{i+k} / N$ ,  $f_{i+k}$  denotes the total number of instances belonging to class  $c_k$  as shown in Table 3.1, and  $N$  denotes the total number of instances in the data block.

Since  $f_{ijk} \cong f'_{ijk}$  and  $N = N'$  we can obtain

$$P_{ik} = f_{i+k} / N = \sum_{j=1}^v f_{ijk} / N \cong - \sum_{j=1}^v f'_{ijk} / N' = f'_{i+k} / N' = P'_{ik}.$$

Continually, we can obtain that

$$- \sum_{k=1}^m P_{ik} \log P_{ik} \cong - \sum_{k=1}^m P'_{ik} \log P'_{ik} \text{ and}$$

$$I(A_i) \cong I(A'_i). \quad (3.3)$$

That is, the Entropy of attribute  $a_i$  before splitting in data blocks  $D$  and  $D'$  is similar.

Suppose we splitting all instances  $N$  into  $v$  subset by attribute  $A_i$ , the Entropy of attribute  $A_i$



after splitting is

$$E(A_i) = \sum_{j=1}^v \frac{f_{ij+}}{N} \times \left( - \sum_{k=1}^m P_{ijk} \log P_{ijk} \right), \quad (3.4)$$

where  $P_{ijk} = f_{ijk} / f_{ij+}$ ,  $f_{ij+}$  denotes the total number of instances having attribute value  $a_{ij}$  as shown in Table 3.1.

Since  $f_{ijk} \cong f'_{ijk}$  we can infer that for the attribute value  $a_{ij}$

$$f_{ij+} \cong f'_{ij+}. \quad (3.5)$$

As a result, we can also obtain that

$$P_{ijk} \cong P'_{ijk} \text{ and } - \sum_{k=1}^m P_{ijk} \log P_{ijk} \cong - \sum_{k=1}^m P'_{ijk} \log P'_{ijk} \quad (3.6)$$

From Formulas (3.4), (3.5) and (3.6), we can get that

$$E(A_i) \cong E(A'_i). \quad (3.7)$$

From Formulas (3.3) and (3.7), we can get that

$$Gain(A_i) = I(A_i) - E(A_i) \cong I(A'_i) - E(A'_i) = Gain(A'_i).$$

That is, the Information gain of attribute  $A_i$  in data blocks  $D$  and  $D'$  is similar.

As a result, the two decision trees which are respectively built by using blocks  $D$  and  $D'$  will be similar. ■

**Table 3.1** The class label distribution on an attribute  $A_i$

Class label \ value	$a_{i1}$	$a_{i2}$	...	$a_{iv}$	Summation
$c_1$	$f_{i11}$	$f_{i21}$		$f_{iv1}$	$f_{i+1}$
$c_2$	$f_{i12}$	$f_{i22}$		$f_{iv2}$	$f_{i+2}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$c_m$	$f_{i1m}$	$f_{i2m}$		$f_{ivm}$	$f_{i+m}$
Summation	$f_{i1+}$	$f_{i2+}$		$f_{iv+}$	$N$

By Proposition 3.1 and Formula 3.2, we can detect any kind of concept drift between two data blocks and then build an accurate decision tree. The significance level can be set to be

smaller or larger according to the needs of applications. With a given significance level, we can obtain the  $\varepsilon$  by checking the  $X^2$  table in a statistical book. The degree of freedom will be 1 less than the number of classes. Suppose that we set the level of significance  $\alpha = 5\%$  and there are three classes, if all  $CDAV_{D \rightarrow D'}(i, j)$  are less than  $\varepsilon = 5.991$ , that means the class distribution on all attributes shows no significant difference between  $D$  and  $D'$  with 95% confidence. As a result, the information gain obtained from any attribute will show no significant difference and the decision tree need not to be rebuilt. Note that the purpose of Proposition 3.1 is to claim that a rebuild tree will have very similar accuracy to that of original one, rather than to guarantee the rebuild tree will be a copy of the original one.

**Example 3.1:** For clearly understand our idea, a case with two datasets  $D$  and  $D'$  is presented in Table 3.2. Each of the two sets has two attributes  $A_1$  and  $A_2$ , and each attribute has three attribute values ( $a_{11}, a_{12}, a_{13}; a_{21}, a_{22}, a_{23}$ ). There are total 500 instances and two classes are  $c_1$  and  $c_2$  in each dataset. Assuming that the level of significance  $\alpha = 5\%$  (*degree of freedom* = 1 and  $\varepsilon = 3.841$ ), we can infer the following by Formula 3.2:

$$CDAV_{D \rightarrow D'}(1,1) = 0.6723 < \varepsilon ;$$

$$CDAV_{D \rightarrow D'}(1,2) = 0.5948 < \varepsilon ;$$

$$CDAV_{D \rightarrow D'}(1,3) = 0.5326 < \varepsilon ;$$

$$CDAV_{D \rightarrow D'}(2,1) = 2.7763 < \varepsilon ;$$

$$CDAV_{D \rightarrow D'}(2,2) = 1.7223 < \varepsilon ;$$

$$CDAV_{D \rightarrow D'}(2,3) = 1.5948 < \varepsilon .$$

Since all  $CDAV$ s have no significant difference, by Proposition 3.1 mentioned above, the decision trees built respectively with  $D$  and  $D'$  would be very similar. To verify this, we build the two decision trees and show the corresponding rules. The rules obtained from data set  $D$  are

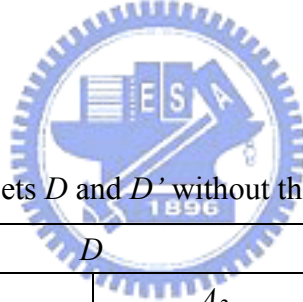
$$(1) A_1 = "a_{12}" \rightarrow c_2;$$

- (2)  $A_1 = "a_{13}" \rightarrow c_2$ ;  
(3)  $A_1 = "a_{11}" \cap A_2 = "a_{21}" \rightarrow c_2$ ;  
(4)  $A_1 = "a_{11}" \cap A_2 = "a_{22}" \rightarrow c_1$ ;  
(5)  $A_1 = "a_{11}" \cap A_2 = "a_{23}" \rightarrow c_2$ .

And the rules obtained from data set  $D'$  are

- (1)  $A_1 = "a_{12}" \rightarrow c_2$ ;  
(2)  $A_1 = "a_{13}" \rightarrow c_2$ ;  
(3)  $A_1 = "a_{11}" \cap A_2 = "a_{21}" \rightarrow c_2$ ;  
(4)  $A_1 = "a_{11}" \cap A_2 = "a_{22}" \rightarrow c_1$ ;  
(5)  $A_1 = "a_{11}" \cap A_2 = "a_{23}" \rightarrow c_2$ .

We can find that the two decision tree have identical rules. This result corresponds to Proposition 3.1.



**Table 3.2** Two data sets  $D$  and  $D'$  without the occurrence of concept drift

Dataset		$D$						$D'$					
attribute		$A_1$			$A_2$			$A_1$			$A_2$		
Attribute value		$a_{11}$	$a_{12}$	$a_{13}$	$a_{21}$	$a_{22}$	$a_{23}$	$a_{11}$	$a_{12}$	$a_{13}$	$a_{21}$	$a_{22}$	$a_{23}$
Class label	$c_1$	192	41	13	18	216	12	198	42	12	25	211	15
	$c_2$	33	142	79	74	122	58	37	133	73	76	108	65

**Corollary 3.1:** By Proposition 3.1, we can infer that if the variance of  $CDAV$  for the two data blocks  $D$  and  $D'$  is greater than or equivalent to a threshold  $\varepsilon$ , (i.e.  $CDAV_{D \rightarrow D'}(i,j) \geq \varepsilon$ ), then concept drift may occur between  $D$  and  $D'$ . As a result, the original decision tree needs to be corrected.

**Example 3.2:** Here, we use the two datasets in Table 3.3, which is modified from Table 3.2, to illustrate this Corollary. Again assuming that the level of significant  $\alpha = 5\%$  (degree of

$freedom = 1$  and  $\varepsilon = 3.841$ ), we can infer the following by Formula 2:

$$CDAV_{D \rightarrow D'}(1,1) = 3.0848 < \varepsilon;$$

$$CDAV_{D \rightarrow D'}(1,2) = 1.5402 < \varepsilon;$$

$$\mathbf{CDAV_{D \rightarrow D'}(1,3) = 5.9085 > \varepsilon;}$$

$$CDAV_{D \rightarrow D'}(2,1) = 1.8754 < \varepsilon;$$

$$CDAV_{D \rightarrow D'}(2,2) = 0.4274 < \varepsilon;$$

$$CDAV_{D \rightarrow D'}(2,3) = 2.7299 < \varepsilon$$

Since  $CDAV_{13}$  achieves significant difference, by Corollary 3.1, we can claim that concept drift occurs and the decision trees built respectively with  $D$  and  $D'$  would be different. To verify this, we again show the corresponding rules for two trees as follows. The rules obtained from data set  $D$  are

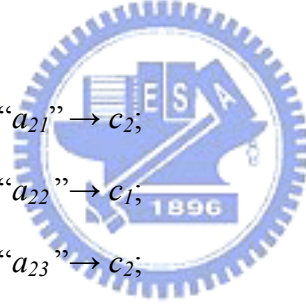
$$(1) A_1 = "a_{12}" \rightarrow c_2;$$

$$(2) A_1 = "a_{11}" \cap A_2 = "a_{21}" \rightarrow c_2;$$

$$(3) A_1 = "a_{11}" \cap A_2 = "a_{22}" \rightarrow c_1;$$

$$(4) A_1 = "a_{11}" \cap A_2 = "a_{23}" \rightarrow c_2;$$

$$(5) A_1 = "a_{13}" \rightarrow c_2.$$



And the rules obtained from data set  $D'$  are

$$(1) A_1 = "a_{12}" \rightarrow c_2;$$

$$(2) A_1 = "a_{11}" \cap A_2 = "a_{21}" \rightarrow c_2;$$

$$(3) A_1 = "a_{11}" \cap A_2 = "a_{22}" \rightarrow c_1;$$

$$(4) A_1 = "a_{11}" \cap A_2 = "a_{23}" \rightarrow c_2;$$

$$(5) A_1 = "a_{13}" \cap A_2 = "a_{21}" \rightarrow c_2;$$

$$(6) A_1 = "a_{13}" \cap A_2 = "a_{22}" \rightarrow c_1;$$

$$(7) A_1 = "a_{13}" \cap A_2 = "a_{23}" \rightarrow c_2.$$

By comparison, we can find that the rule  $A_1 = a_{13} \rightarrow c_2$  in dataset  $D$  have some changes in data set  $D'$ ; the results correspond to our Corollary.

**Table 3.3** Two data sets  $D$  and  $D'$  with the occurrence of concept drift

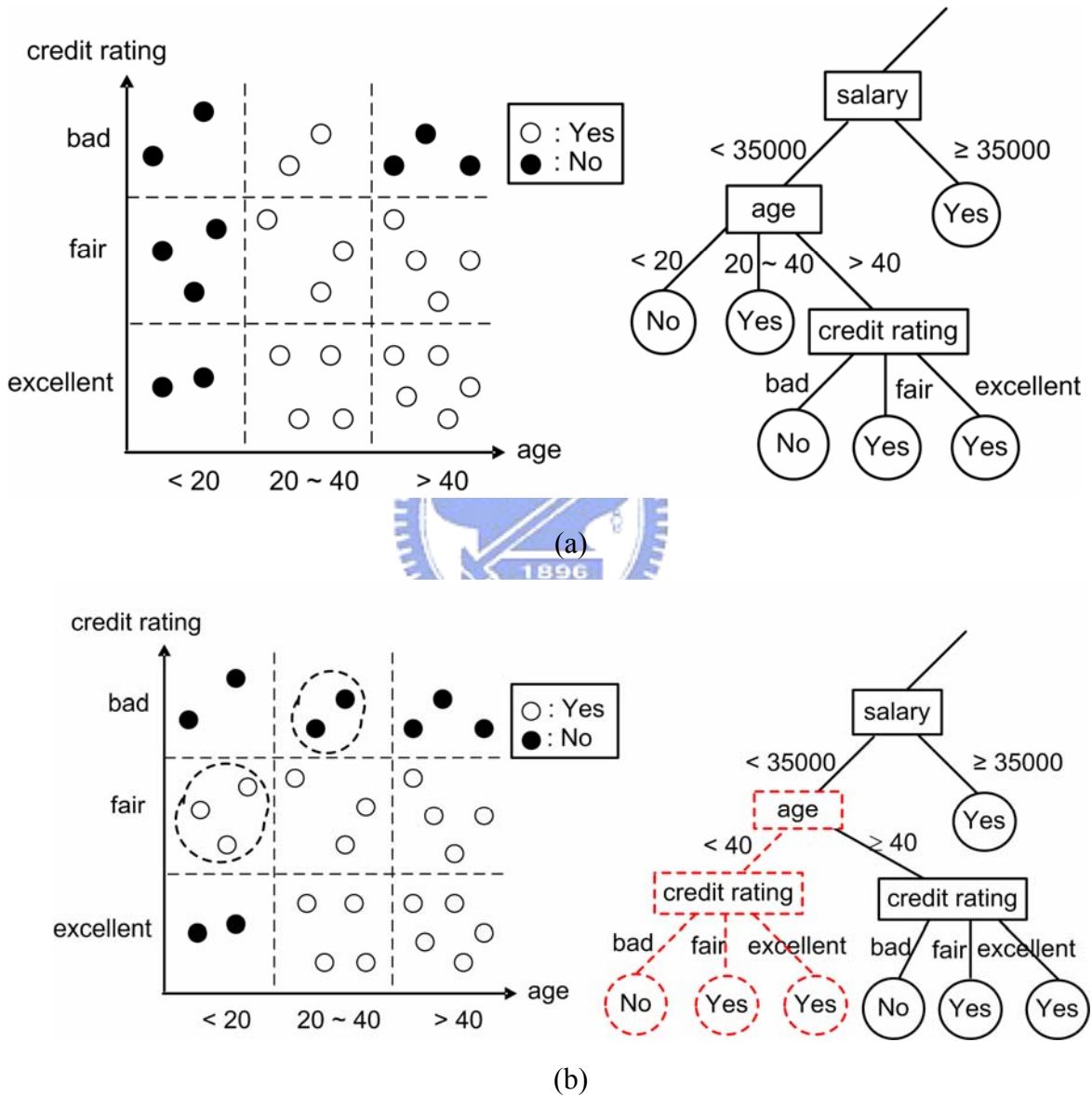
Dataset		$D$						$D'$					
attribute		$A_1$			$A_2$			$A_1$			$A_2$		
Attribute value		$a_{11}$	$a_{12}$	$a_{13}$	$a_{21}$	$a_{22}$	$a_{23}$	$a_{11}$	$a_{12}$	$a_{13}$	$a_{21}$	$a_{22}$	$a_{23}$
Class label	$c_1$	192	41	13	18	216	12	203	34	20	23	208	16
	$c_2$	33	142	79	74	122	58	42	135	66	68	118	67

### 3.2.2 Correction Mechanism in SCRIPT

Before we introduce the correction mechanism in SCRIPT, it is worth to note that drifting instances should gather in some specific areas in the dimensional space of attributes, otherwise they can be regarded as noise instances. Accordingly, another advantage of CDAV is that it can reveal which attribute values cause concept drift before building the decision tree by aggregating the drifting CDAVs. This enables SCRIPT to efficiently and immediately amend the original decision tree. For example, we can recognize the concept drift is caused by attribute value  $a_{13}$  in Example 3.2. Therefore, we can only correct the subtree rooted at  $a_{13}$  to efficiently correct the classification model.

**Example 3.3:** We use Figure 3.2 to further illustrate the idea of correction mechanism in SCRIPT. Figure 3.2 is a decision tree trained from old customer's data to predict if a customer will apply for credit cards. For better understanding, only the subtree rooted at attribute "salary" is shown. A similar decision tree, except that it is trained from new customer's data stream, is shown in Figure 3.2 (b). By comparison with the CDAVs in Figure 3.2 (a) and Figure 3.2 (b), we can find that some concepts in new data block are significantly different from that in old one. More importantly, we can find that these changes gather up in the branch

of “age < 20 and 20 ≤ age < 40”. Accordingly, the aggregated drifting CDAVs is  $0 \leq \text{age} < 40$  and it means that a people younger than 40 have changed his concepts in this example. To efficiently provide a decision tree suitable for new customers’ data block, we can only correct the subtree rooted at  $0 \leq \text{age} < 40$  as in Figure 3.2 (b).



**Figure 3.2.** Two data blocks with the occurrence of concept drift: (a) original data block and the corresponding sub-tree; (b) new data block and the corresponding sub-tree.

Now, we detail the correction mechanism in SCRIPT. In the processing of data stream,

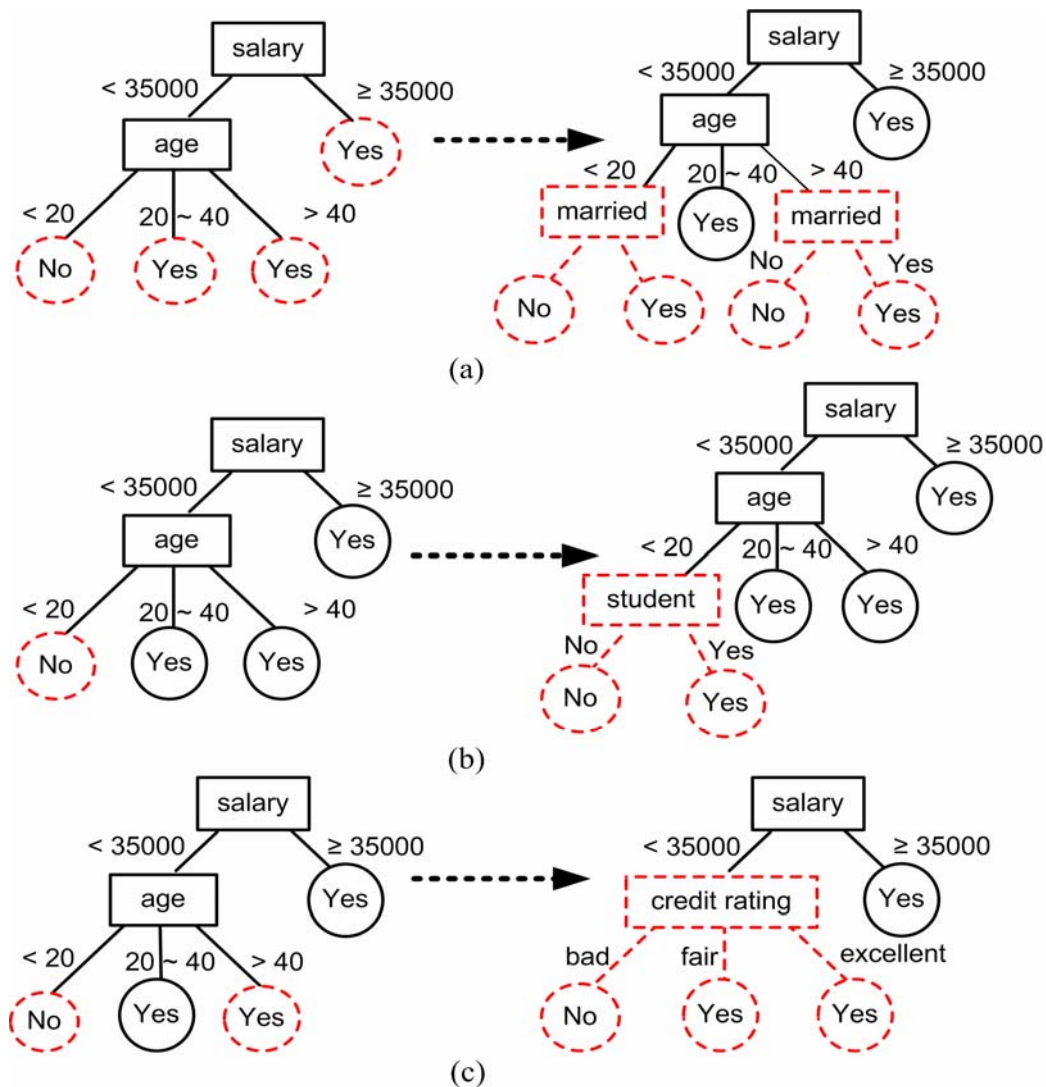
when the difference of CDAV between new data block  $B_t$  and original data block  $B_{t-i}$  ( $t \geq i \geq 1$ ) is greater than the given threshold (the level of significance is set 0.05 as the default), the correction methods in SCRIPT can be divided into the following cases. The corresponding illustration of each case is shown in Figure 3.3. In each case of Figure 3.3, the dotted node in the left tree (original tree) denoted the occurring of concept drift and the dotted subtree in the right tree (new tree) is an alternate tree built by SCRIPT.

For each aggregated drifting CDAV in attribute  $A_i$  with value(s)  $a_{ij}$ ,

- a. If attribute  $A_i$  is not a splitting attribute of a node in the original decision tree, SCRIPT will use this attribute to split all leaf nodes by using data block  $B_t$ . Such a variation of CDAV indicates that an attribute with originally little information changes into an optimal splitting attribute due to concept drift. We illustrate this condition in Figure 3.3(a).
- b. If attribute  $A_i$  is a splitting attribute of a node in the original decision tree and all CDAVs group in an interval  $a_{ij}$ , SCRIPT will remove the subtree rooted at the attribute value  $a_{ij}$  from the original tree and use data block  $B_t$  to build the alternative tree. Such a variation means concept drift is caused by a fixed range  $a_{ij}$  of the attribute  $A_i$ . Take Figure 3.3(b) for example, for the attribute age, those under 20 were originally inclined not to apply for credit cards; however, with the growing consuming ability of students, more and more are applying.
- c. If attribute  $A_i$  is a split attribute of a node in the original decision tree but all CDAVs are scattered in several interval, SCRIPT will remove the subtree rooted at this attribute  $A_i$  from the original tree and use data block  $B_t$  to build the alternative tree. Such a variation represents concept drift is caused by the attribute  $A_i$  but within multiple ranges of the attribute. For instance, for the attribute of age, people younger than 20 and older than 40 were originally both inclined not to apply for credit cards; however, with the change of payment types, more and more are applying. In this case,

attribute 'age', no longer the optimal split attribute, is replaced by attribute 'credit rating', according to a test result. This case is illustrated in Figure 3.3(c).

Note that all aggregated drifting CDAVs might distribute among several attributes and SCRIPT will check if they are in the same path in the original tree before the correctness. If two aggregated CDAVs are in the same path, the one locates in the highest level will be reserved and the other will be ignored.



**Figure 3.3** The illustrations of the correction mechanism in SCRIPT when concept drift occurs.



### 3.2.3 The Pseudocode and Computational Complexity of SCRIPT

Here we present the pseudo-code of SCRIPT and analyze its computational complexity. Below is the pseudo code of SCRIPT. Giving the size of data block  $N$  and the significance level  $\alpha$ , SCRIPT calculates the CDAVs in data block  $B_0$  in Line 4 as the initial reference. Note that,  $N$  can be set larger in a high speed environment or smaller for the real time application; however,  $f_{ijk}$  must be larger than 5 which is a basic requirement in  $X^2$  statistics test. Similarly,  $\alpha$  can be set smaller if the detection of concept drift is very important and larger otherwise. The default significant level  $\alpha$  in SCRIPT is set as 0.05 since this value is widely used as the default in statistics. It is not hard to imagine that SCRIPT will be more sensitive to the concept drift but may require more computational cost if we use a larger significant level  $\alpha$ ; on the contrary, SCRIPT will be more tolerant to the noise data with a smaller  $\alpha$ . The CDAVs of new coming block  $B_{t+1}$  are calculated in Line 7. The CDAVs of two data blocks  $B_t$  and  $B_{t+1}$  are compared in Lines 8 to 11. All drifting CDAVs are then aggregated in Line 13 for the purpose of efficiently correcting the decision model in Lines 19 to 28. The recorded information is updated in Line 29. Finally, the decision tree is output in Line 31.

## SCRIPT Algorithm

Inputs:  $N$ : the size of the block (1000 as the default);

$B_t$ : the data block in time step  $t$ ;

$\alpha$ : the level of significance (0.05 as the default);

SCRIPT ( $N, \alpha, B_t$ )

/\* Initialization \*/

1.  $t = 0$ ;
2. Build the original decision tree  $DT_0$  by  $B_0$ ;
3. Record all splitting attributes of  $DT_0$  in  $Splitatt[]$ ;
4. Count the  $CDAVs$  in  $B_0$  and record them in  $RCDAV[]$ ;
5.  $t = t + 1$
6. For the new coming data block  $B_t$  in time step  $t$
7. Count  $CDAV_{ijk}$  in  $NCDAV[]$ ;
8. For each  $CDAV_{ij}$  in the new data block
9. If  $|CDAV_{t-1 \rightarrow t}(i, j)| > \varepsilon$
10. Record this  $CDAV$  in  $DCDAV[]$ ;
11. End if
12. If  $DCDAV[]$  is not empty
13. Aggregate the recorded  $CDAVs$ ;
14. For all aggregated  $CDAVs$
15. If two aggregated  $CDAVs$  are in the same path in the original decision tree
16. Reserve the one locates in the highest level in  $ACDAV[]$  ;
17. End if
18. Update ;
19. For all aggregated  $CDAVs$  belonged to attribute  $A_i$  in  $ACDAV[]$
20. If attribute  $A_i$  is not in  $Splitatt[]$
21. Build an alternative tree rooted at this node by using  $B_i$ ;
22. Elseif attribute  $A_i$  is in  $Splitatt[]$  and all aggregated  $CDAVs$  in  $a_i$  group in an interval  $a_{ij}$
23. Remove the subtree rooted at this attribute value  $a_{ij}$  from  $DT_{t-1}$ ;
24. Build the alternative tree rooted at  $a_{ij}$  by using  $B_i$ ;
25. Else
26. Remove the subtree rooted at this attribute  $A_i$  from  $DT_{t-1}$ ;
27. Build the alternative tree rooted at  $A_j$  by using  $B_i$ ;
28. End if
29. Update  $Splitatt[]$ ,  $RCDAV[]$ , and the recorded decision tree;
30. End If
31. Output the decision tree.

Below, we compare the system cost of SCRIPT to that of two state-of-the-art window-based approaches: DNW and CVFDT. Assumed a data block has  $i$  attributes,  $k$  class labels, each attribute has  $j$  attribute values, since SCRIPT records the referred CDAVs, it has a memory cost  $O(ijk)$ . SCRIPT also needs to record a decision tree and the splitting attributes in this tree, however, the memory cost is  $O(n)$  and can be ignored, where  $n$  is the number of nodes of the recorded decision tree. For CVFDT, since it has to record the counting in each node of the recorded decision tree, the memory cost is  $O(nijk)$ . DNW, which is a sliding window approach, might have a worst record cost since it record instances instead counting and new data blocks might have to be mixed into old ones. If the maintained data is  $w$  times as much as that of a new data block, then the memory cost of DNW is  $O(wNijk)$ , where  $N$  is the number of instances in the block.

In the aspect of computational cost, while the concept is stable, the required computational cost of SCRIPT is  $O(ijk)$ . For CVFDT, it has to check the information gain for each attribute in each node of decision tree when a new data block is given. If the tree has  $n$  nodes, the computational cost needed would be  $O(nijk)$  [38]. For DNW, the computational cost would be  $O(dwNijk)$ , where  $d$  is the depth of the tree, since the tree is rebuilt from scratch. When there is concept drift, DNW and CVFDT have a similar computational cost to that in stable stream. Since SCRIPT directly corrects some sub-trees by checking the drifting CDAVs, the computational cost of the rebuilding is  $O(ij)+O(n'ijk)$ , where  $n' \leq n$  and  $O(ij)$  is responsible for the comparison of CDAVs and  $O(n'ijk)$  is the computational cost for the rebuilding of sub-trees. Comparisons of system cost among SCRIPT, DNW, and CVFDT in stable and drifting data stream are summarized in Table 3.4. In summary, SCRIPT has the smallest memory requirement and computational cost when concept is stable. When concept drifts, SCRIPT still requires the smallest memory cost and a better or comparable computational cost.

**Table 3.4** The Comparisons of system cost among SCRIPT, DNW, and CVFDT

Algorithm	Concept Stable		Concept Drift	
	Memory Cost	Computational Cost	Memory Cost	Computational Cost
DNW	$O(wNijk)$	$O(dwNijk)$	$O(wNijk)$	$O(dwNijk)$
CVFDT	$O(nijk)$	$O(nijk)$	$O(nijk)$	$O(nijk)$
SCRIPT	$O(ijk)$	$O(ijk)$	$O(ijk)$	$O(n'ijk)$



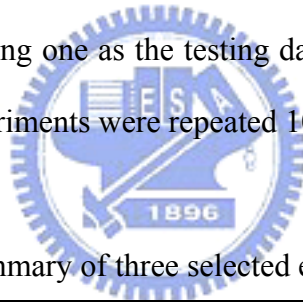
### 3.3 Experiment and Analysis

In this section, two the-state-of-art data stream mining algorithms, DNW and CVFDT, are implemented to compare with our SCRIPT. We run all experiments on a PC equipped with Windows XP professional operating system, Pentium III 1GHz CPU and 512mb SDRAM memory. For the preset of parameters in DNW, we refer to [38] and set  $\alpha = 5.0$ ,  $\beta = 0.25$ , and  $\gamma = 0.50$ .

#### 3.3.1 Experimental Datasets

Due to the lack of a benchmark containing concept-drifting datasets, we modified three selected UCI datasets in which the number of instances is larger than 4000 as our experimental datasets. The summary of the three UCI datasets is shown in Table 3.5. To simulate a data stream, for each UCI dataset we first divide it into three data blocks  $B_0$ ,  $B_1$ , and  $B_2$  to simulate a stable data stream. Then, we code a program to generate consecutively data blocks contain drifting concepts by modifying  $B_2$ . As described in Section 3.1, one purpose of SCRIPT is to detect the drifting concepts more sensitively to make it suitable for the applications in which a small part of drifting concepts would cause large damage. To evaluate the sensitiveness about the detection of concept drift, we set the drifting ratio as 1%; that is, there are  $N \times (t-2)\%$  drifting instances in the data block  $B_t$  in time step  $t$  ( $t \geq 2$ ), where  $N$  is the number of instances in  $B_0$ . This program works as follows. First, it randomly picks up one instance  $S$  in data block  $B_t$  and randomly selects attributes  $a_m$  ( $1 \leq m \leq 10$ ) for reference. Instances, which have the same values in all attributes  $a_m$  to that of  $S$ , are picked out. The class label and values  $\in a_m$  of these picked out instances are then replaced by a random value in the

corresponding value-domain. The main principle of our program is that concept drifts are caused by the variances of some attributes. We limit the number of referable attributes less than 10 since drifting concepts should be caused by some but not a lot attribute values. For example, age and salary may influence the application of credit cards but weight and height will not; IP address and the number of sending packages may be the main basis to find a PC which sends virus package; fraudulent credit card transactions can be detected by the payment amount and location. If the number of drifting instances is less than the requirement, the program goes on next loop to get more drifting instances. On the contrary, if there are more instances satisfy the requirement,  $N$  % instances are randomly picked up as drifting ones. Consequently, for each UCI dataset, we generate 13 data blocks ( $B_0 \sim B_{12}$ ) and each data block is divided into 10 parts of which nine parts are used as training dataset to calculate the execution time and the remaining one as the testing dataset to count the accuracy. Finally, to get an objective result, all experiments were repeated 10 times to obtain the average.



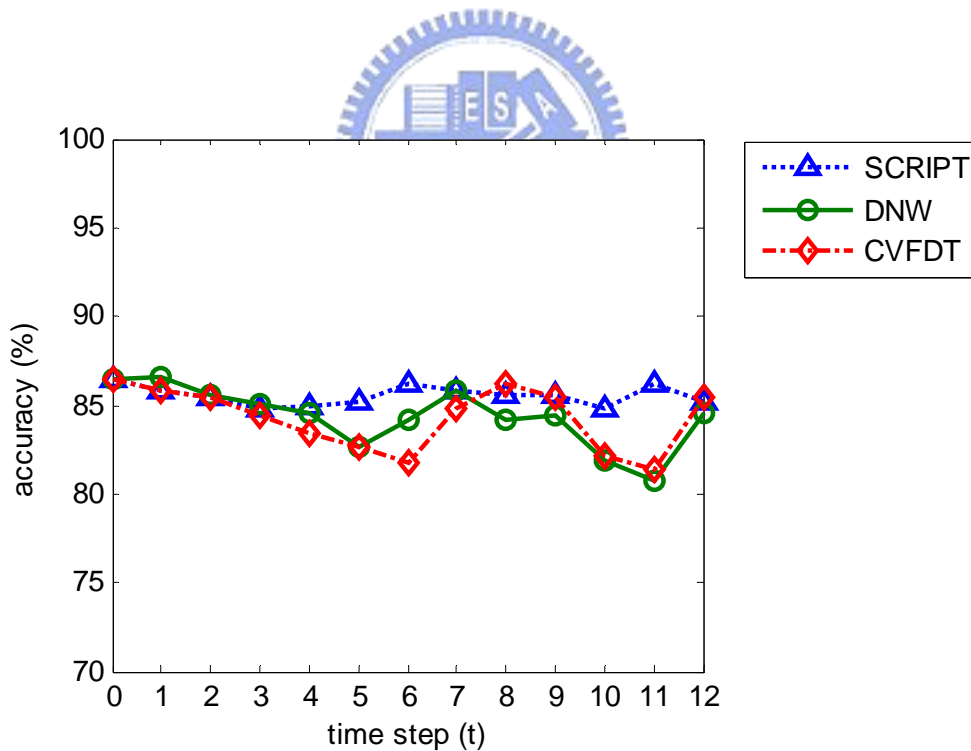
**Table 3.5** The summary of three selected experimental UCI datasets

Dataset	Number of instances	Number of attributes	Number of class labels
satimage	6435	36	6
thy	7200	21	3
spambase	4601	57	2

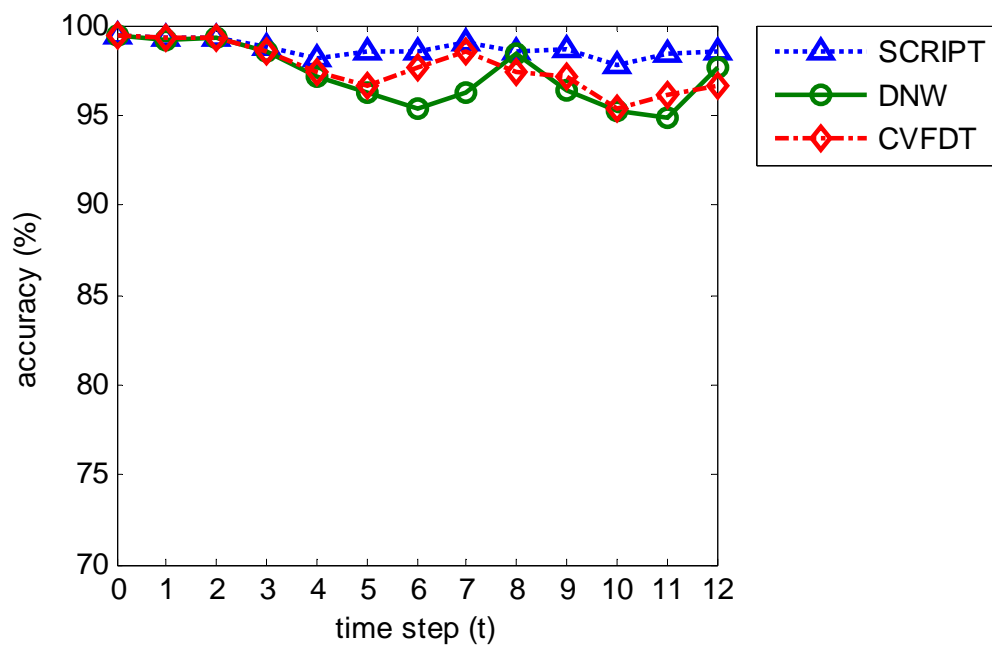
### 3.3.2 The Comparison of Accuracy

Figures 3.4 to 3.6 illustrate the comparison of accuracy among DNW, CVFDT, and SCRIPT on different time step  $t$ . In Figures 3.4 to 3.6, we can find that SCRIPT has a similar accuracy to that of DNW and CVFDT in stable data blocks  $B_0$  to  $B_2$ . It follows Proposition 3.1 that when concept is stable, the accuracy of the SCRIPT is similar to that of classifier which is

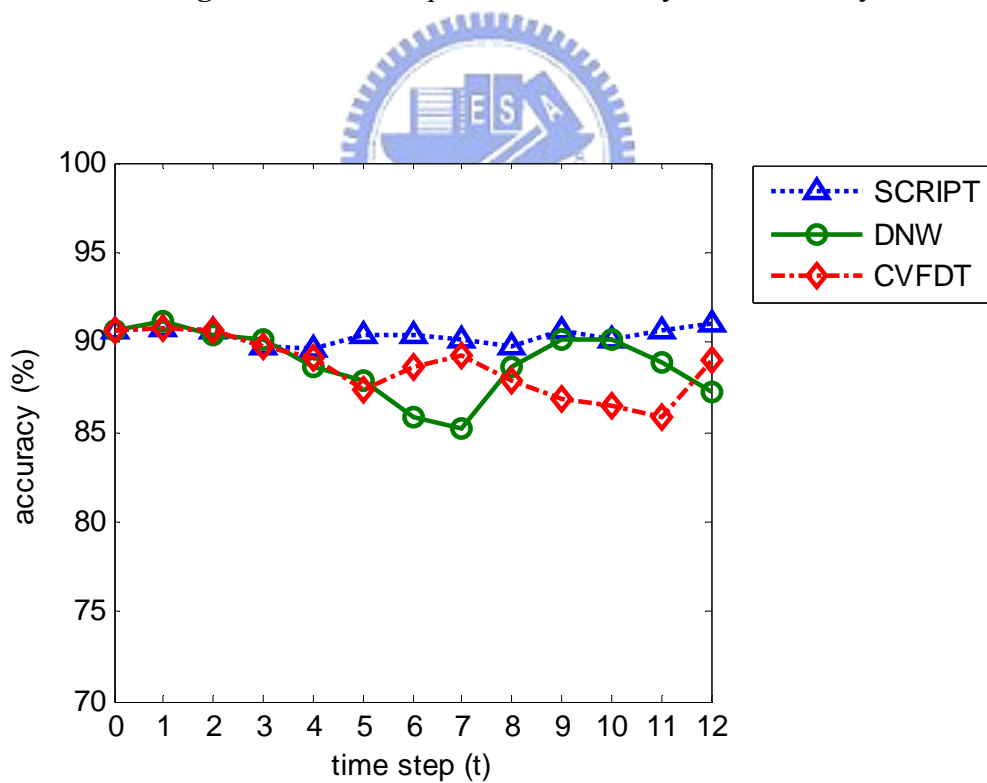
rebuilt. When there are drifting concepts from data blocks  $B_3$  to  $B_{12}$ , we can find SCRIPT is much more sensitive than DNW and CVFDT. That is, SCRIPT always keeps a high accuracy but DNW and CVFDT can detect the changes until the number of drifting instances reaches a threshold to cause obvious difference in accuracy or information gain. Take the satimage dataset in Figure 3.4 for example, DNW averagely recognizes the drifting concepts in blocks  $B_6$ ,  $B_7$  and  $B_{12}$  and therefore reduces the window size to build an up-to-date decision tree; CVFDT averagely uses the alternate subtree to correct the original classification model to raise the accuracy in time step 7, 8 and 12. Similar conditions occur in thy and spambase dataset: in thy, DNW usually discovers the changes in time step 7, 8, and 12, and in time step 8 and 9 in spambase; CVFDT averagely recognizes the drifting concepts in time step 6, 7, and 11 in thy and spambase dataset.



**Figure 3.4** The comparison of accuracy on dataset ‘satimage’.



**Figure 3.5** The comparison of accuracy on dataset 'thy'.

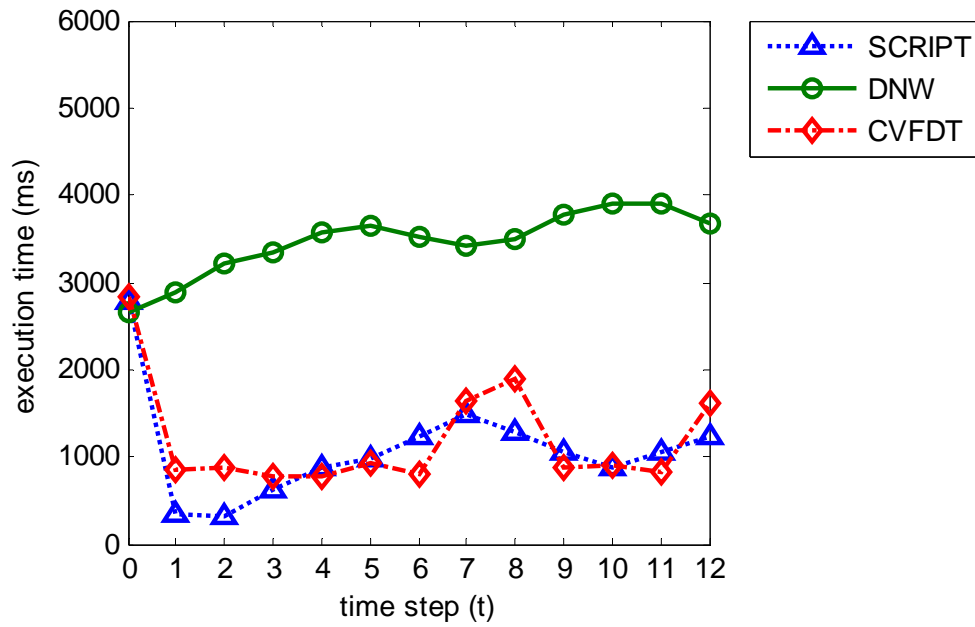


**Figure 3.6** The comparison of accuracy on dataset 'spambase'.

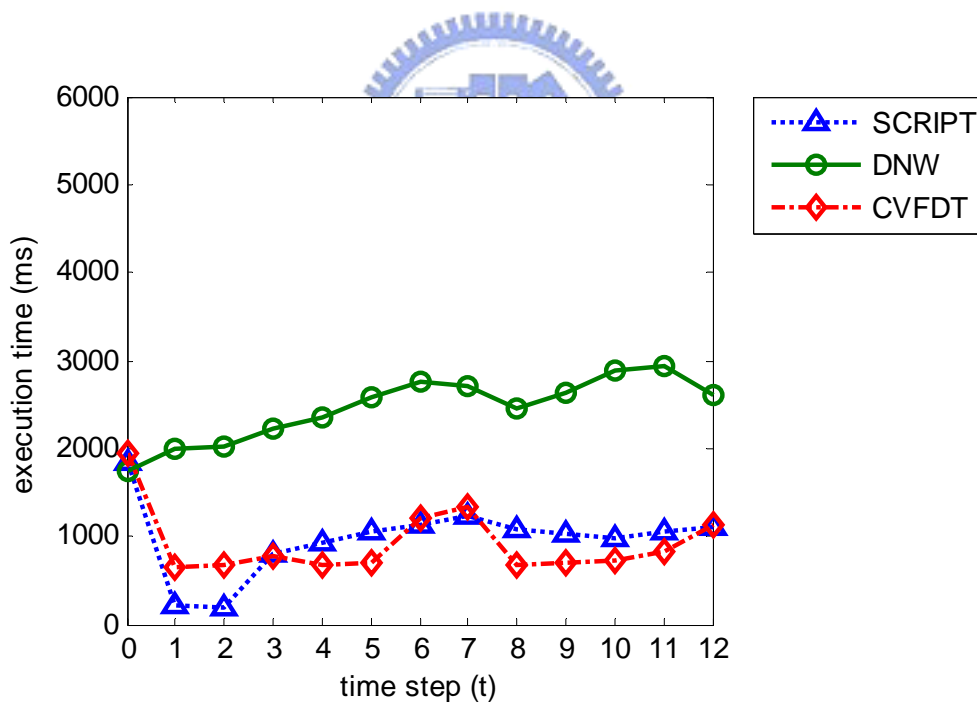


### 3.3.3 The Comparison of Execution Time

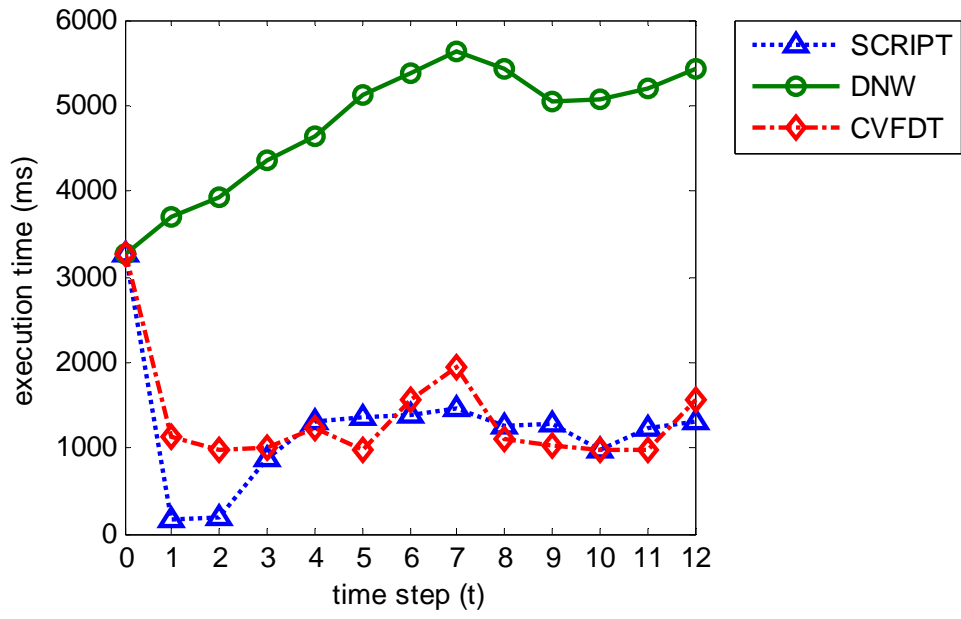
In the aspect of comparison of execution time in Figures 3.7 to 3.9, SCRIPT, DNW, and CVFDT have a similar execution time for the building of the first decision tree in data block  $B_0$ . However, SCRIPT and CVFDT requires a little more execution time in the initial step since SCRIPT needs to calculate the CDAVs and CVFDT must record the counts in each node. After the beginning step, SCRIPT is much more efficient than DNW and CVFDT when concept is stable in time step 1 and 2. When concept drifts, the execution time of SCRIPT is worse than that of CVFDT in most cases. However, this is caused by the fact that SCRIPT recognizes the drifting concepts and therefore needs more execution time to correct the original decision tree in these time steps. It is worth to note that when both SCRIPT and CVFDT detect the drifting instances and correct the decision tree, e.g. in time step 7, 8 and 12 in Fig. 8, SCRIPT is more efficient than CVFDT. The reason is SCRIPT can immediately know which sub-trees should be amended by checking the drifting CDAVs but CVFDT have to check the variation of information gain node by node from the root. Similar condition can be found in time step 6 and 7 in thy and spambase dataset. DNW, which builds a new classifier in each time step, always has the worst computational cost. The computational is much worse as time goes by since DNW does not recognize the concept drift and therefore mixes the new data block into the old one.



**Figure 3.7** The comparison of execution time on dataset 'satimage'.



**Figure 3.8** The comparison of execution time on dataset 'thy'.



**Figure 3.9** The comparison of execution time on dataset ‘spambase’.



## Chapter 4

# Concept Drift Rule Mining Tree Algorithm

In previous chapter, we propose SCRIPT as a solution to sensitively and efficiently handle the concept-drifting problem on data stream. However, as most proposed approaches about concept drift, SCRIPT focuses on updating the classification model to accurately predict new coming data. As for the users, they might be more interested in the rules of concept drift. For example, doctors desire to know the main causes of disease variation, scholars long for the rules of weather transition, and sellers would like to find out the reasons why the consumers' shopping habits change. In this chapter, we concentrate our focus on this problem. We first use an example to illustrate the problem of concept-drifting rules in Section 4.1. Then, the details of concept drift rule mining tree algorithm (CDR-Tree) are elucidated in Section 4.2. Experimental analyses and performance evaluations are shown in Section 4.3.

### 4.1 The Rules of Concept Drift

Here, we use a simple example to formally introduce the concept-drifting rules.

**Example 4.1:** Take the patients' diagnostic data in Table 4.1 as the example and assume that

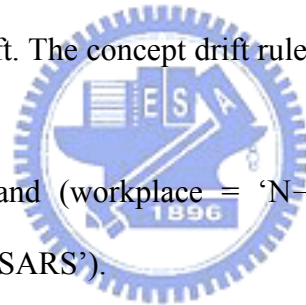
Table 4.2 is the new diagnostic dataset. In Table 4.2, the drifting values are marked with both underline and boldface. An instance with the same ID in both tables means the diagnostic data belongs to the same patient. Figures 4.1 and 4.2 are the decision trees constructed by using Tables 4.1 and 4.2, respectively. Comparing Figure 4.1 with Figure 4.2, we find that patients ID<sub>9</sub> and ID<sub>10</sub> are located in leaf node A in the old decision tree and in node B in the new one. The corresponding decision rules are:

If (fever = 'no') and (cough = 'no') then (diagnosis = 'healthy') and

If (fever = 'yes') and (workplace = 'S') and (cough = 'yes') then (diagnosis = 'SARS').

Comparing the rules of those two patients, we can see that someone might be infected with SARS if he displays fever, his working location is transferred to city S, and had a bad cough. Simply stated, the variations of attributes 'fever', 'workplace' and 'cough', are the primary factors influencing concept drift. The concept drift rules detected from the two patients can be written in the form:

If (fever = 'no→yes') and (workplace = 'N→S') and (cough = 'no→yes') then (diagnosis = 'healthy → SARS').



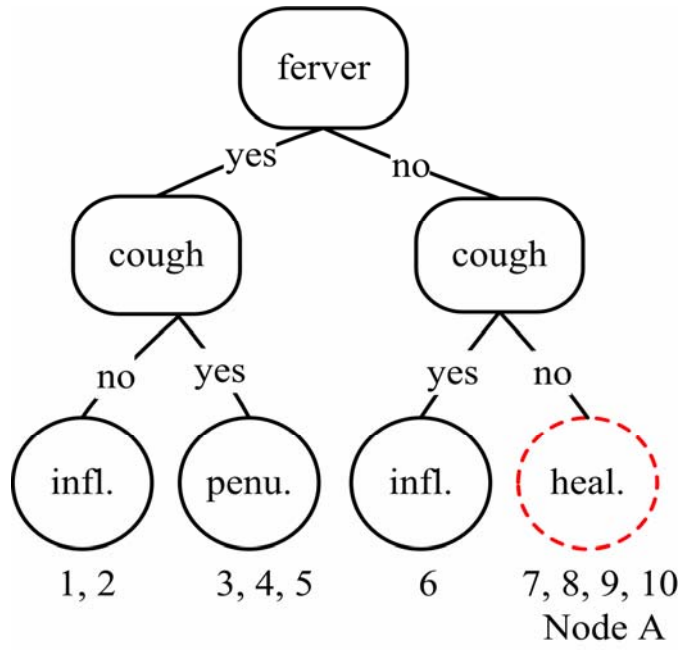
In this example, owing to the few instances and very simple rules, users can clearly and quickly find the concept-drifting rules between the two datasets. However in a real application, it is a very difficult task for users to figure out such rules since the number of produced rules is usually very large.

**Table 4.1** The patients' diagnostic data

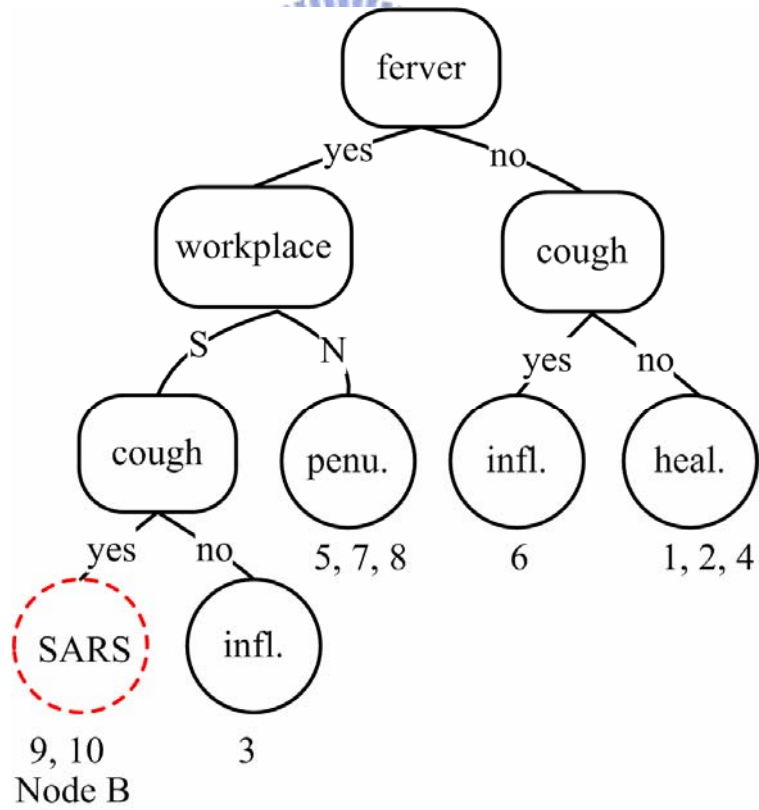
ID	sex	workplace	fever	cough	diagnosis
1	male	N	yes	no	influenza
2	female	C	yes	no	influenza
3	female	N	yes	yes	pneumonia
4	male	N	yes	yes	pneumonia
5	male	C	yes	yes	pneumonia
6	male	N	no	yes	influenza
7	female	N	no	no	healthy
8	male	N	no	no	healthy
9	female	C	no	no	healthy
10	female	C	no	no	healthy

**Table 4.2** The new coming diagnostic data from the same patients

ID	sex	workplace	fever	cough	diagnosis
1	male	<u>C</u>	<u>no</u>	no	<u>healthy</u>
2	female	C	<u>no</u>	no	<u>healthy</u>
3	female	<u>S</u>	yes	<u>no</u>	<u>influenza</u>
4	male	N	<u>no</u>	<u>no</u>	<u>healthy</u>
5	male	<u>N</u>	yes	<u>no</u>	pneumonia
6	male	N	no	yes	influenza
7	female	N	<u>yes</u>	no	<u>pneumonia</u>
8	male	N	<u>yes</u>	<u>yes</u>	<u>pneumonia</u>
9	female	<u>S</u>	<u>yes</u>	<u>yes</u>	<u>SARS</u>
10	female	<u>S</u>	<u>yes</u>	<u>yes</u>	<u>SARS</u>



**Figure 4.1** The decision tree built using Table 4.1.




**Figure 4.2** The decision tree built using Table 4.2.

## 4.2 Concept Drift Rule Mining Tree Algorithm

In order to mine the concept-drifting rules mentioned in Section 4.1, here we propose our CDR-Tree algorithm. Section 4.2.1 is the building step of CDR-Tree. Without loss of generality, here we consider only the case that there are two data blocks:  $T^p$  and  $T^q$  in a data stream. Note that, users might also require the classification model of each data block after they check these rules of concept drift. CDR-Tree can do that via a quick and simple extraction step as is presented in Section 4.2.2.

### 4.2.1 Building a CDR-Tree



To mine concept-drifting rules, CDR-Tree algorithm initially integrates new and old instances from different times into pairs; then, following the manner of traditional decision trees a CDR-Tree is built. During the building step, information gain is used as the criterion to select the best splitting attribute in each node. In other words, CDR-Tree regards the pair made by integration of new and old data as a single attribute value and mines the rules of concept drift through the construction of a traditional decision tree. In addition, since a traditional decision trees stop building while a node is pure, the generated concept drifting rules would miss some important information.

**Example 4.2:** Taking Tables 4.1 and 4.2 as our example again, the integrated data of the two tables are shown in Table 4.3, and Figure 4.3 is the corresponding CDR-Tree. As described in Example 4.1, for the patients  $ID_9$  and  $ID_{10}$ , there is a drifting rule:

If (fever = ‘no    yes’) and (workplace = ‘C    S’) and (cough = ‘no    yes’) then



(diagnosis = 'healthy SARS'),

However, a traditional decision tree will stop splitting at the node C in Figure 4.3 and then produce a rule:

If (fever = 'no yes') and (workplace = 'C S') then (diagnosis = 'healthy SARS').

It is clear that the former rule is more reliable and accurate than the latter one.

**Table 4.3** The integrated data of Table 4.1 and Table 4.2

ID	workplace	fever	cough	diagnosis
1	N <u>C</u>	yes <u>no</u>	no no	influenza <u>healthy</u>
2	C C	yes <u>no</u>	no no	influenza <u>healthy</u>
3	N <u>S</u>	yes yes	yes <u>no</u>	pneumonia <u>influenza</u>
4	N	yes <u>no</u>	yes <u>no</u>	pneumonia <u>healthy</u>
5	C <u>N</u>	yes yes	yes <u>no</u>	pneumonia pneumonia
6	N N	no no	yes yes	influenza influenza
7	N N	no <u>yes</u>	no no	healthy <u>pneumonia</u>
8	N N	no <u>yes</u>	no <u>yes</u>	healthy <u>pneumonia</u>
9	C <u>S</u>	no <u>yes</u>	no <u>yes</u>	healthy <u>SARS</u>
10	C <u>S</u>	no <u>yes</u>	no <u>yes</u>	healthy <u>SARS</u>

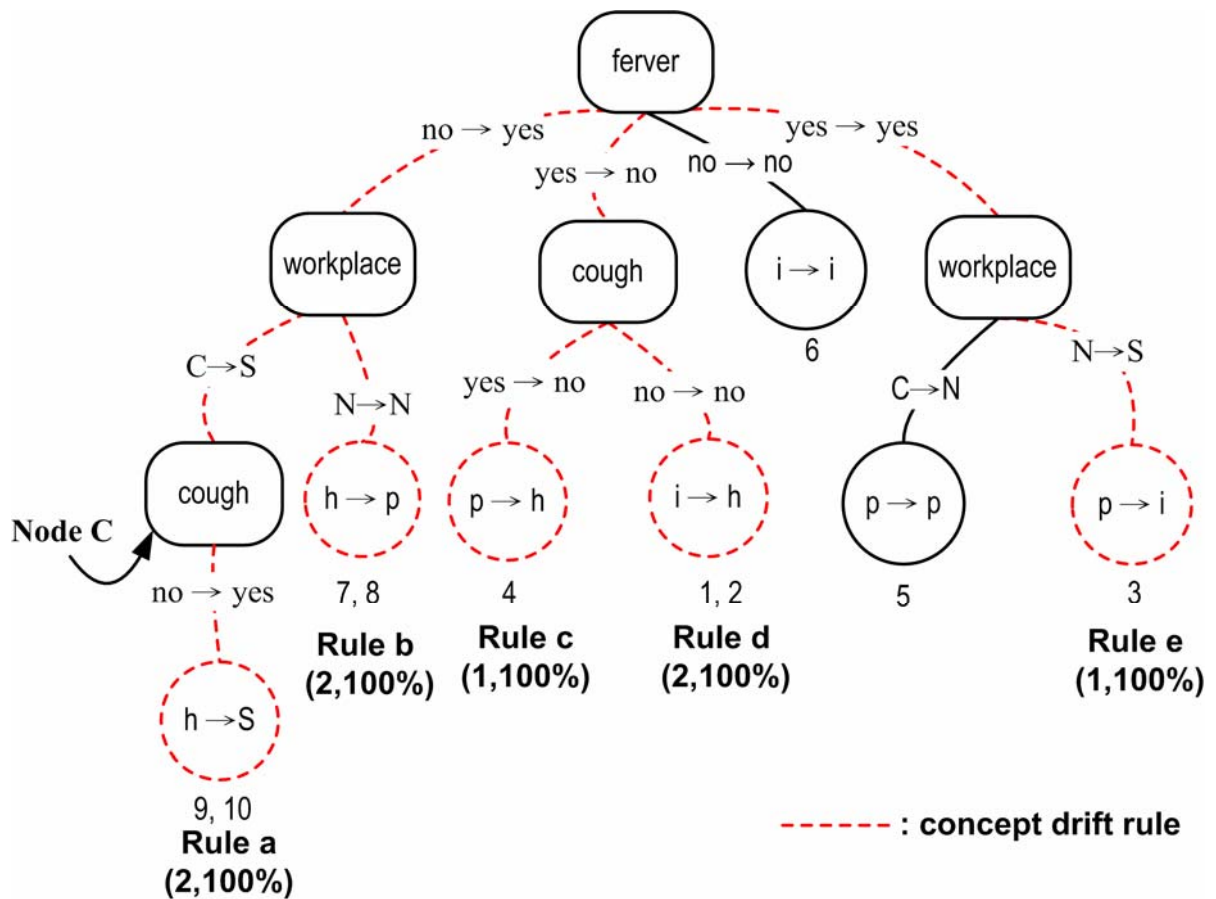


Figure 4.3 The CDR-Tree built using Table 4.3.

To solve this problem, CDR-Tree algorithm goes on splitting a pure node  $n_o$  in which all instances have some common attribute value but this attribute is never selected as a splitting attribute in this path from the node  $n_o$  to the root. The concept drifting rules are marked with dotted lines in the CDR-Tree in Figure 4.3. There are five concept drift rules as follows:

**Rule a:** If (fever = ‘no yes’) and (workplace = ‘C S’) and (cough = ‘no Yes’) then (diagnosis = ‘healthy SARS’);

**Rule b:** If (fever = ‘no yes’) and (workplace = ‘N N’) then (diagnosis = ‘healthy pneumonia’);

**Rule c:** If (fever = ‘yes no’) and (cough = ‘yes no’) then (diagnosis = ‘pneumonia healthy’);

**Rule d:** If (fever = ‘yes no’) and (cough = ‘no no’) then (diagnosis = ‘influenza

healthy’);

**Rule e:** If (fever = ‘yes      yes’) and (workplace = ‘N      S’) then (diagnosis = ‘pneumonia      influenza’).

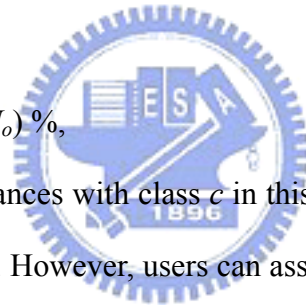
In the above rules, the value on the left and right side of ‘      ’ respectively represents the value in two different data blocks of a data stream. If observing carefully, we can find that the concept-drifting rules of the patients ID<sub>9</sub> and ID<sub>10</sub> mentioned in Example 4.1 are definitely mined by this CDR-Tree.

In order to provide users with meaningful and interesting rules of concept drift, CDR-Tree defines a rule support  $RS$  and a rule confidence  $RC$  to filter un-meaningful ones out. For a leaf node  $n_o$  in the CDR-Tree, suppose this node is assigned class label  $c$  and contains  $N_o$  instance, then:

$$RS = N_o \text{ and}$$

$$RC = (100N_c / N_o) \%,$$

where  $N_c$  is the number of instances with class  $c$  in this node  $n_o$ . The default values of  $RS$  and  $RC$  are 2 and 50% respectively. However, users can assign a larger threshold if they only want to check the notable rules. For example, by setting  $RS = 2$  and  $RC = 90\%$ , Rules c and e will be filtered out.  $RS$  and  $RC$  are shown in the form of  $(RS, RC)$  in Figure 4.3. Below is the pseudo code of CDR-Tree



### CDR-Tree Algorithm

$T^p, T^q$  : the data block of times  $p$  and  $q$  respectively;

$T_{ij}^p, T_{ij}^q$  : the value of attribute  $j$  of instance  $ID_i$  in data blocks  $T^p$  and  $T^q$  respectively;

$T_{ij}^{p,q}$  : the value of attribute  $j$  of instance  $ID_i$  in the combined data  $T^{p,q}$  ;

$T^{p,q}(o)$ : the instances in the node  $o$ ;

$df$ : default value for goal predicate;

$N_o$ : the total number of instances in the leaf node  $o$

$N_c$ : the number of instances in the leaf node  $o$  with class label  $c$ ;

$RS$ : the rule support  $RS$ , which is set to 2 as the default

$RC$ : the rule confidence  $RC$ , which is set to 50% as the default

$CDRTree(T^p, T^q, df, RS, RC)$

For each  $ID$  in the data block  $T_p$  and  $T_q$

Combine  $T_{ij}^p$  and  $T_{ij}^q$  as  $T_{ij} = (T_{ij}^p, T_{ij}^q)$ ;

If the combined data  $T^{p,q}$  is empty then

Return  $df$ ;

Else if  $T^{p,q}$  are all of the same class  $c$  then

Return the  $c$ ;

Else

For each node  $n_o$

Partition  $T^{p,q}(o)$  by the attribute  $A_i$  whose information gain is the highest;

If the node  $n_o$  is pure or can not be split then

While all instances in node  $n_o$  have common attribute value  $a_{ij}$  and the attribute  $A$  is never selected as a splitting attribute in the path form node  $n_o$  to the root

Do

Go on splitting node  $n_o$  by using attribute  $A_i$ ;

Assign the class label by majority vote;

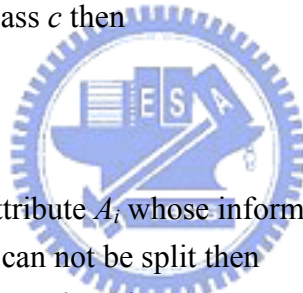
$RS_o = N_o$ ;

$RC_o = (100N_c / N_o) \%$

If  $RS_o \geq RS$  and  $RC_o \geq RC$  then

Mark the path form this leaf node  $n_o$  to root as a concept-drifting rule;

Return  $CDR-Tree$ ;



## 4.2.2 Extracting Decision Trees from a CDR-Tree

When users require the old or new decision tree, or both, in addition to the concept-drifting rules, CDR-Tree algorithm can provide them efficiently and accurately via the following extraction steps:

**Step 1.** To extract the old (new) classification model, the splitting attribute values of all internal nodes and the class labels in all leaf nodes of the new (old) instances are ignored.

**Step 2.** Check each node from the bottom-up and left-to-right.

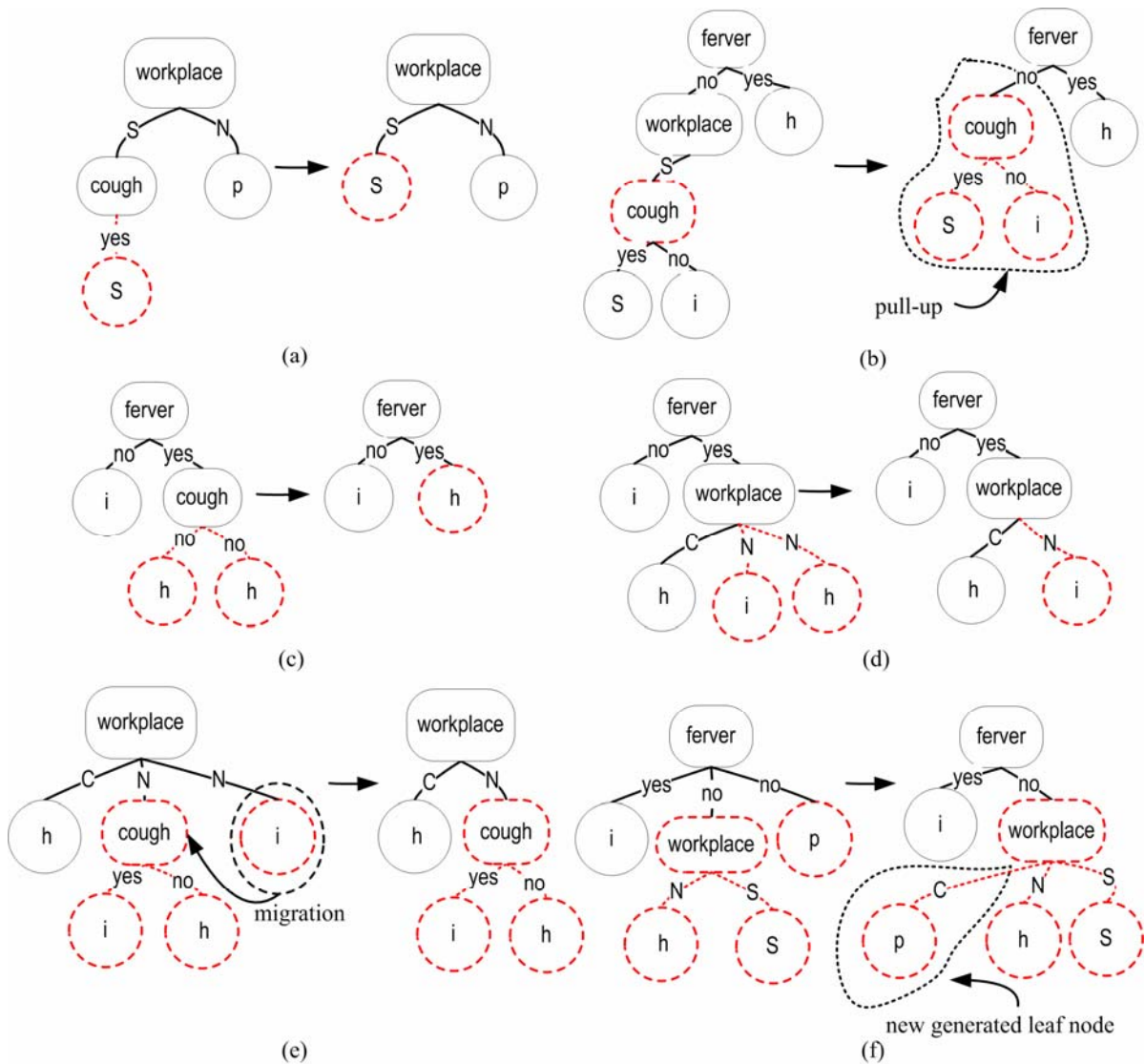
**Step 3.** For any node  $n_o$  and its sibling node(s)  $n_s$ ,

- (a.) If node  $n_o$  is a leaf and singleton node (i.e. it does not have any sibling node), its parent node will be removed from the CDR-Tree and node  $n_o$  will be pulled-up. This situation is illustrated in Figure 4.4(a).
- (b.) If node  $n_o$  is an internal and singleton node, the parent node of  $n_o$  will be removed and the sub-tree rooted at  $n_o$  will be pulled-up. This situation is illustrated in Figure 4.4(b).
- (c.) If  $n_s$  has the same splitting value as that of  $n_o$  and  $n_o, n_s$  are all leaf nodes. CDR-Tree will merge them into a single node  $n_m$ . The class label of  $n_m$  is assigned by a majority vote. This situation is illustrated in Figures 4.4(c) and (d).
- (d.) If  $n_s$  has the same splitting value as that of  $n_o$  but not all of them are leaf nodes, CDR-Tree will pick out the internal node  $n_m$ , which contains the most instances among all internal nodes  $n_s$ . Except for the sub-tree  $ST_m$  rooted at  $n_m$ , all sibling nodes and their sub-trees are then removed from the CDR-Tree. The instances, which belong to these removed leaf nodes and sub-trees, are migrated into the internal node  $n_m$  and will follow the path of  $ST_m$  until they reach a leaf node as Figure 4.4(e) illustrates. Note that a migrant instance may stop in an internal node

$n_l$  of  $ST_m$  if there is no branch to proceed. In such a condition, the CDR-Tree will use the splitting attribute in  $n_l$  to generate a new branch and accordingly a new leaf node as illustrated in Figure 4.4(f). The target class of the leaf nodes in  $ST_m$  and the newly generated leaf node(s) are then assigned by a majority vote.

**Step 4.** Repeat Step 2 until no more nodes can be merged.

**Step 5.** If there is a leaf node that is not pure, continue splitting it.



**Figure 4.4** Illustrations of the extraction strategy in CDR-Tree algorithm.

Due to the merging strategy, some leaf nodes in a CDR-Tree might be not pure. The goal

of Step 5 is to solve this problem. However, this step can be omitted if users do not really need an overly detailed decision tree. Note that the CDR-Tree keeps the count information in each node during its building step; therefore, the computational cost for this extraction procedure is small. Compare this to building a decision tree from the beginning; CDR-Tree can generate the decision tree much more efficiently and quickly. Below is the pseudo code of the CDR-Tree's extraction procedure.



## The extraction procedure of CDR-Tree

CDRTreeExtract (*CDR-Tree*)

If the decision tree of old instances is requested then

    The splitting attribute values in all internal nodes and the class labels in all leaf nodes of the new instances are ignored;

Else

    The splitting attribute values in all internal nodes and the class labels in all leaf nodes of the old instances are ignored;

End if

For node  $n_o$  and its sibling node(s)  $n_s$  in the CDR-Tree

    If node  $n_o$  is a leaf and singleton node then

        Remove its parent node from the CDR-Tree;

        Pull-up node  $n_o$ ;

    If node  $n_o$  is an internal and singleton node then

        Remove its parent node from the CDR-Tree;

        Pull-up the sub-tree rooted at  $n_o$ ;

    If  $n_s$  has the same splitting value to that of  $n_o$  and  $n_o, n_s$  are all leaf nodes then

        Merge them into a single node  $n_m$ ;

        Assign a class label to  $n_m$  by the majority vote;

    If  $n_s$  has the same splitting value to that of  $n_o$  but not all of them are leaf nodes then

        Pick out the internal node  $n_m$  with the most instances among all internal nodes;

        Remove all the sibling nodes and their sub-trees except for the sub-tree  $ST_m$  rooted at  $n_m$ ;

        Migrate the instances belong to these removed leaf nodes and sub-tree into the internal node  $n_m$ ;

    For each migrant instance in  $ST_m$

        If it can reach a leaf node

            Migrate it into the leaf node;

        Else

            Migrate it into the internal node where no branch can be proceeded;

        End if

    For each node in the path of  $ST_m$

        If it is a leaf node and contains migrant instances then

            Assign a target class to it by the majority vote;

        If it is a internal node and contains migrant instances then

            Create new branch and corresponding leaf nodes;

            Assign a class label to the new leaf nodes by the majority vote;

For node leaf node in the extracted CDR-Tree

    If it is not pure then

        Go on splitting it;

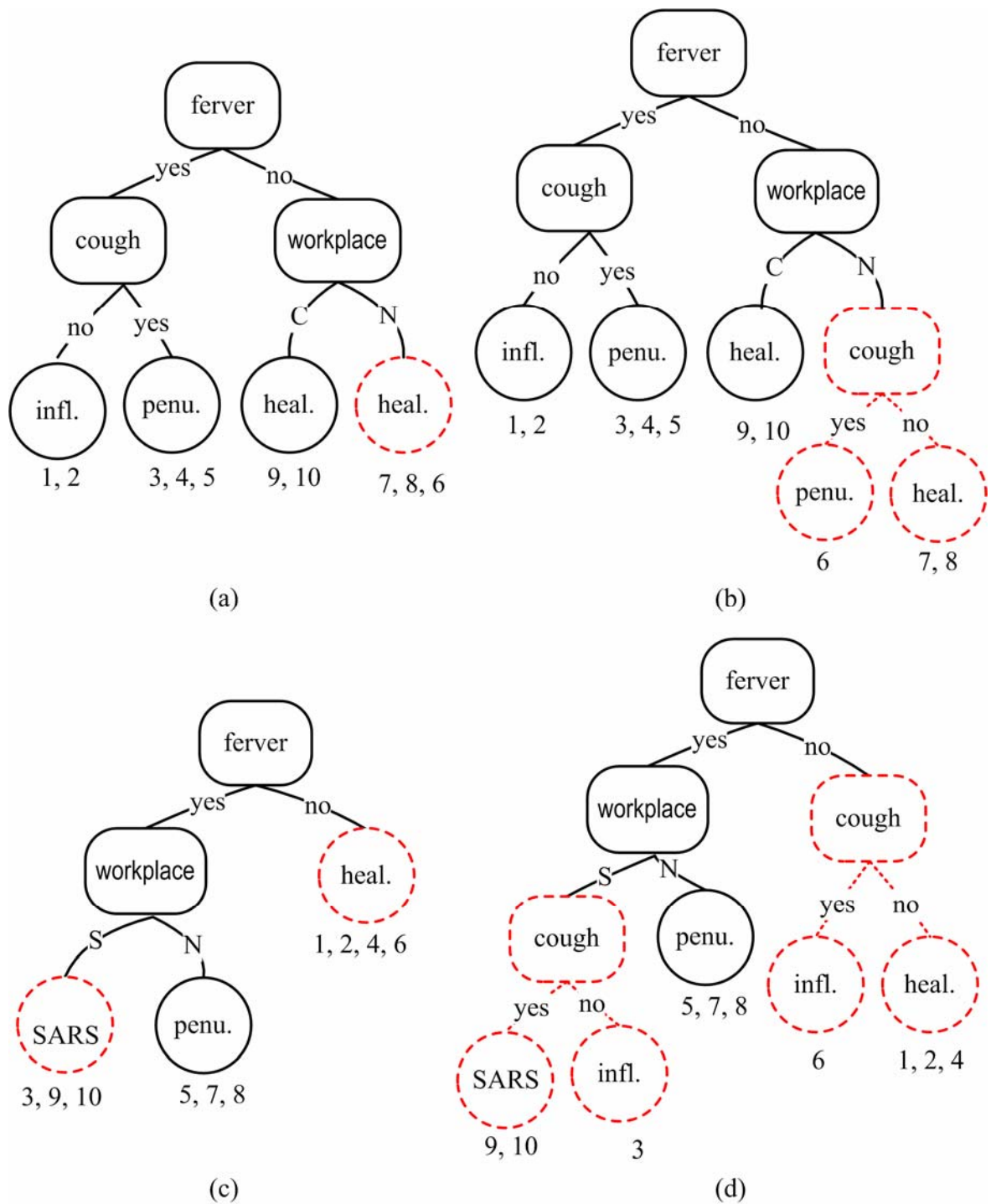
    End if

Return extracted decision tree



**Example 4.3:** Taking the CDR-Tree in Figure 4.3 as an example, the extract decision trees are shown in Figure 4.5, where Figure 4.5(a) is the old classification model for Table 4.1 without implementing Step 5; Figure 4.5(b) is still the model for Table 4.1 but with the implementation of Step 5; and Figures 4.5(c) and (d) correspond to Table 4.2. By comparing these results to those in Figures 4.1 and 4.2, we can find that without the implementation of Step 5, there are only 1 misclassified instance in Figure 4.5(a) and 2 ones in Figure 4.5(c). When Step 5 is executed, Figure 4.5(b) and Figure 4.5(d) reach 100% accuracy as are Figure 4.1 and 4.2. Furthermore, Figure 4.5(d) is identical to Figure 4.2, but Figure 4.5(b) is a little different from Figure 4.1. From this example we determine that although the decision tree extracted from the CDR-Tree is not proved to be identical to that built from the beginning, it can reach a comparable accuracy even without the implementation of Step 5.





**Figure 4.5** The extracted decision trees from Fig. 4.3: (a) the model of Table 4.1 without implementing Step 5; (b) the model of Table 4.1 with the implementation of Step 5; (c) the model of Table 4.2 without implementing Step 5; (d) the model of Table 4.2 with the implementation of Step 5.

### 4.3 Experiment and Analysis

We implement CDR-Tree algorithm in Microsoft Visual C++ 6.0 for experimental analysis and performance evaluation. The experimental environment and datasets are clearly described in Section 4.1. In Section 4.2, we demonstrate how the accuracy of CDR-Trees is affected by different drifting levels. We compare the accuracy of C4.5 to that of the model extracted from the CDR-Tree in Section 4.3. Finally, the comparison of execution time among CDR-Tree, the model extracted from the CDR-Tree, and C5.0 is given in Section 4.4.

All experiments here are done on a 3.0GHz Pentium IV machine with 512 MB DDR memory, running Windows 2000 professional. Due to the lack of a benchmark containing concept-drifting datasets, our experimental datasets are generated by IBM data generator. We use IBM data generator because we want to generate several kinds of datasets to evaluate our CDR-Tree. In our experiment, four classification functions, Functions F3, F5, F43 and F45, are randomly selected to generate the experimental datasets.

In order to analyze the performance of our CDR-Tree under different drifting ratios  $R\%$  (i.e. the proportion of drifting instances to all instances), we use the four functions mentioned above to generate required experimental datasets. For each function, the noise level is set to 5% and the dataset generated by IBM data generator is regarded as the original/first dataset in the data stream. Then we code a program to amend the first dataset and generate the second ones as a new dataset. Our program works as follows: First, it randomly picks up one instance  $S$  in the original dataset and randomly selects attributes  $a_m$  ( $1 \leq m \leq 5$ ) for reference. Instances, which have the same and values in all attributes  $a_m$  to that of  $S$ , are picked out. The class label and values belonging to  $a_m$  of these picked out instances are then replaced by a random value in the corresponding value-domain. The main principle of our program is that concept drifts are caused by the variances of some attributes. We limit the number of referable attributes less

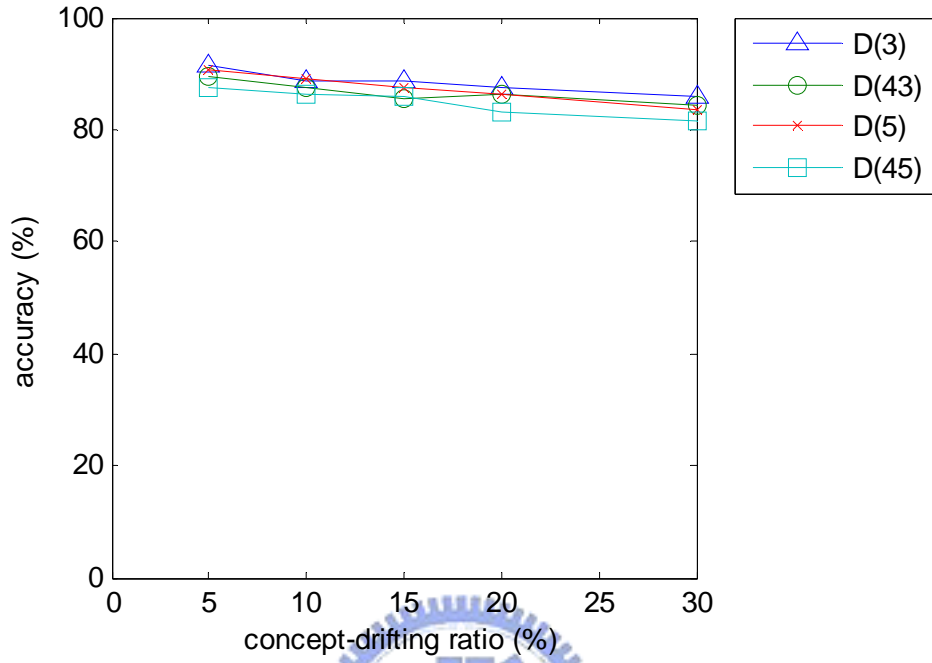
than five since drifting concepts should be caused by some but not a lot attribute values and there are only nine basic attributes in IBM data generator. If the number of drifting instances is less than the requirement, the program goes on next loop to get more drifting instances. On the contrary, if there are more instances satisfy the requirement,  $R$  % instances are randomly picked up as drifting ones. As a result, each function generates five second datasets with different drifting ratios. A total 4 old datasets and 20 new datasets are generated in our experiments. Every dataset includes 10000 instances and the 10-fold cross-validation test method is applied to all experiments. That is, each experimental dataset is divided into 10 parts of which nine parts are used as training sets and the remaining one as the testing set. In the following experiments, we will use  $D(i)$  to denote a dataset generated by Function  $F_i$  and  $D(i,R)$  to represent a dataset with  $R\%$  drifting ratio resulting from  $D(i)$ .

### 4.3.1 The Analysis of CDR-Tree

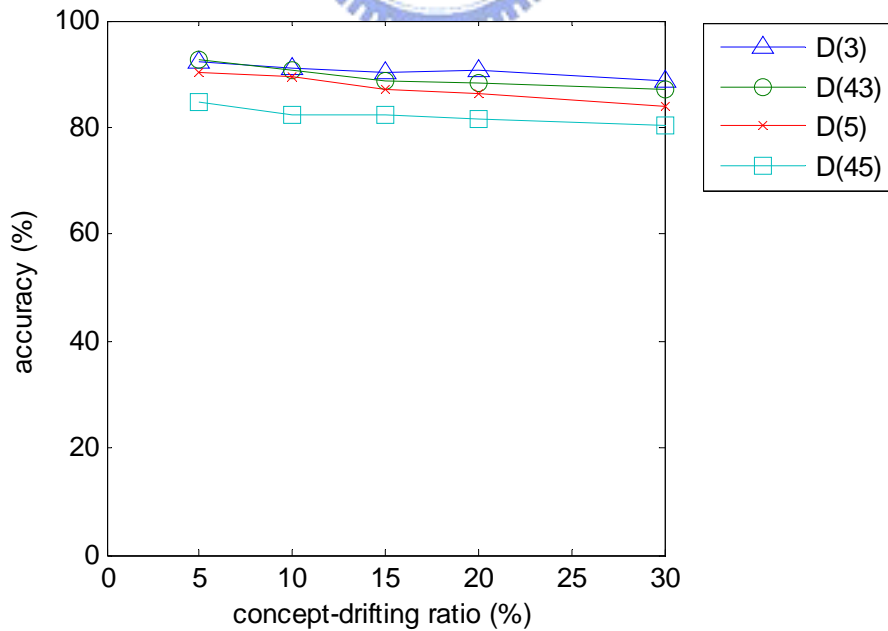


In this section, we use the 24 datasets mentioned in Section 4.3.1 to evaluate the accuracy of CDR-Tree and to analyze whether it can precisely explore the concept-drifting rules. At first, focusing on five different drift levels, the accuracy of the CDR-Tree in 20 integrated datasets is shown in Figure 4.6. As can be found in this figure, CDR-Tree maintains high accuracy in all 20 datasets. However, it is worth noting that the higher the concept-drifting ratio is, the lower the accuracy of CDR-Tree will be. This is because a higher drifting ratio makes the CDR-Tree more complex. To further analyze whether the concept-drifting rules produced by CDR-Tree can accurately predict the drifting instances, for each experimental dataset, we only select the instances that really have a drifting concept from the testing data to calculate the accuracy. The experimental result is shown as Figure 4.7. As expected, the concept drift rules mined by CDR-Tree can accurately predict those drifting

instances.



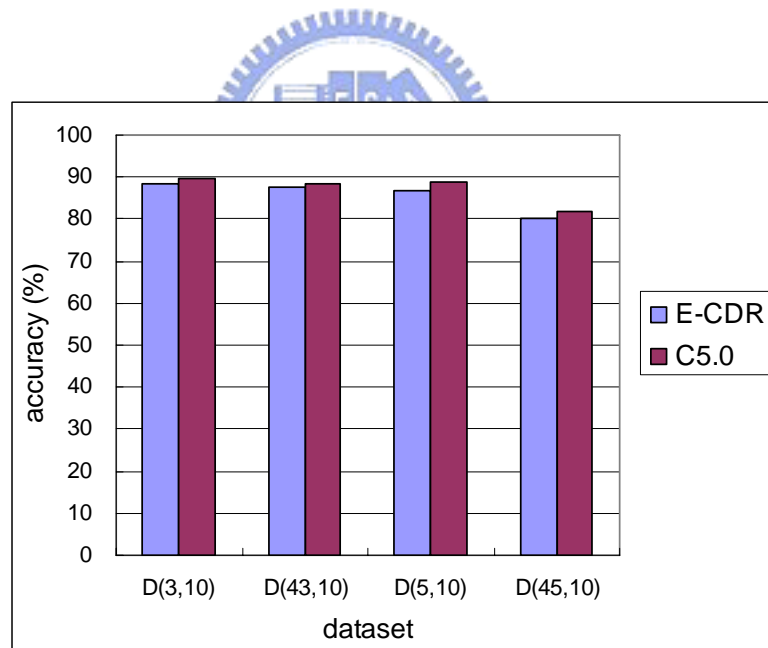
**Figure 4.6** The accuracy of CDR-Trees under five different drifting ratios.



**Figure 4.7** The accuracy of concept-drifting rules produced by CDR-Trees.

### 4.3.2 The Comparison of Accuracy between E-CDR-Tree and C5.0

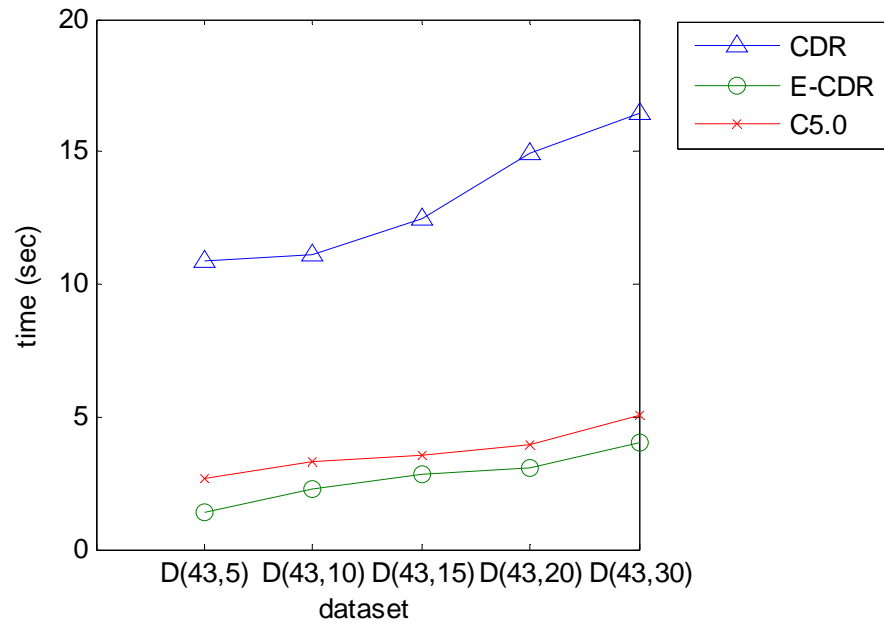
In this experiment, we evaluate whether our approach mentioned in Section 4.2.2 can accurately extract classification models from CDR-Trees. First, all 24 datasets are used by C5.0 to build the decision tree. For 20 CDR-Trees, the old and new classification models are extracted as E-CDR-Trees. Since the results of the 24 datasets are very similar, due to the limitation of content, we only show the accuracy of datasets with 10% drifting ratio. The results are shown in Figure 4.8. From Figure 4.8 we can see that the accuracy of E-CDR-Trees is similar to that of C5.0. This demonstrates the accuracy of our extracting strategy as described in Section 4.2.2.



**Figure 4.8** The comparison of accuracy between E-CDR-Tree and C5.0 using four datasets with 10% drifting ratio.

### 4.3.3 The Comparison of Execution Time among CDR-Tree, E-CDR-Tree, and C5.0

The motivation behind CDR-Tree and C5.0 is inherently different: CDR-Tree mainly aims at providing concept-drifting rules and quickly extracts the decision tree model if it is required by users; but C5.0 is primarily designed to build a decision model to predict the unseen data. Thus comparing the execution time between them might be unfair. However, in order to give readers a clear overview of our approach, we show the comparison of execution time among a CDR-Tree, an E-CDR-Tree, and C5.0 in Figure 4.9. The execution time for C5.0 on dataset  $D(i,R)$  denotes the total building time of two models on datasets  $D(i)$  and  $D(i,R)$ ; that for E-CDR denotes the total time to extract the old and new decision trees. Similarly, due to the limitation of content and the fact that the results of all datasets are very similar, we only show the execution time of datasets generated by Function F43. As expected, the CDR-Tree needs more execution time than C5.0 since the training dataset is more complicated than that used by C5.0. However, the time required for the CDR-Tree to extract the decision tree is much less than that required for C5.0. This demonstrates that with the given CDR-Tree, our extraction strategy proposed in Section 4.2.3 can efficiently elucidate the classification model than building it from scratch.



**Figure 4.9** The comparison of execution time among CDR-Tree, E-CDR-Tree, and C5.0 by using datasets D(43).





# Chapter 5

## Ordered Multi-valued and Multi-labeled Discretization Algorithm

In Chapter 4, we propose CDR-Tree to accurately discover the rules of concept drift. Nevertheless, if there are mass drifting concepts, CDR-tree becomes more complicated and requires much more learning time than a traditional decision tree algorithm. In order to speed up the building procedure of CDR-Tree and reduce the complexity of concept-drifting rules produced by CDR-Tree, we propose an ordered multi-valued and multi-labeled discretization algorithm, named OMMD, to discretize the training data of CDR-Tree. We first give some formal discussions and definitions about the multi-valued and multi-labeled data in Section 5.1. OMMD is then presented in Section 5.2. Section 5.3 is the empirical evaluations of OMMD.

### 5.1 Problem Formulation

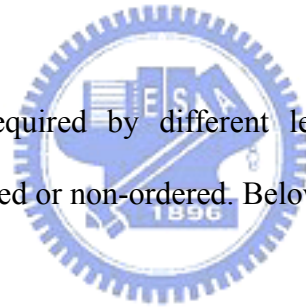
#### 5.1.1 Multi-valued and Multi-labeled Discretization

Although Liu, Hussain and Dash [49] provide a very good overview of proposed discretization algorithms, we believe two new dimensions should be added to make the

discretization literature more complete. The sixth dimension is *univariate* versus *multivariate* [3][7][20][25]. Most past discretization algorithms are univariate in that they consider each attribute independently and do not consider interactions with other features. Bay [3] claims that discretization schemes should be semantically meaningful to human expert. He also uses the XOR data to illustrate that univariate approaches will fail to discretize such data. The only solution to this problem is to use multivariate discretization. Finally, the seventh dimension is the main motivation of this chapter: *single-valued and single-labeled* versus *multi-valued and multi-labeled*.

### 5.1.2 Ordered versus Non-ordered data

According to the demand required by different learning algorithms, multi-valued and multi-labeled data can be ordered or non-ordered. Below we give the formal definitions.



**Definition 5.1:** For a given attribute  $A$ , let  $V = \{v_i \mid i = 1, \dots, p; p \geq 1\}$  be the domain of this attribute and  $L = \{l_j \mid j = 1, \dots, q; q \geq 1\}$  be the set of all class labels, where  $v_i$  is an attribute value and  $l_j$  is a class label. Suppose that a record  $R$  is in the form of  $(\{v'\}, \{l'\})$ , where  $v' \subset V$ ,  $l' \subset L$  and  $v', l' \neq \emptyset$ . If  $|v'| = v$  and  $|l'| = l$ ,  $R$  is called a *v-valued and l-labeled record*.

**Definition 5.2:** Continuing with Definition 5.1, for a dataset  $D$  containing  $v$ -valued and  $l$ -labeled records, if each attribute value  $v'$  in these records corresponds to a class label  $l'$ , then  $D$  is an *ordered multi-valued and multi-labeled dataset* and is a *non-ordered multi-valued and multi-labeled dataset* otherwise.

In a non-ordered multi-valued and multi-labeled dataset, records may have different

number of values  $|v'|$  and different number of labels  $|l'|$ . For example, a platinum credit card user may have several hobbies; a customer may buy two brands of diapers in a transaction. MMC [10] and MMDT [8] are two the-state-of-art algorithms to mine such datasets. Both of MMC and MMDT are decision tree-based approaches and MMDT is the extension of MMC. MMDT refines the measurement of the goodness of a splitting attribute proposed in MMC to improve the classification accuracy. On the other hands, ordered multi-valued and multi-labeled datasets are introduced by CDR-Tree mentioned in Chapter 4. It combines two datasets obtained on different time points and then mines the combined datasets to discover concept-drifting rules. In addition to the datasets used in CDR-Tree, ordered multi-valued and multi-labeled datasets indeed exist in our real life. For example, a user has two kinds of consumption if he has a platinum and a golden credit cards. It is worth to note that since each attribute value of an ordered multi-valued and multi-labeled record corresponds to a class label, the number of values and the number of labels will be the same (i.e.  $|v'| = |l'|$ ). Unfortunately, all past discretization algorithms discussed in Section 2.3 are feasible to discretize multi-valued and multi-labeled datasets, whether ordered or non-ordered. However, there is multi-valued and multi-labeled data in our real life. Tables 5.1 and 5.2 are two simple examples of non-ordered and ordered multi-valued and multi-labeled datasets.

**Table 5.1** A non-ordered multi-valued and multi-labeled dataset

Record	Hobby	Class label(s)
R <sub>1</sub>	{1}	{platinum}
R <sub>2</sub>	{1}	{platinum}
R <sub>3</sub>	{1}	{platinum, normal}
R <sub>4</sub>	{1, 2}	{platinum, golden}
R <sub>5</sub>	{2}	{golden}
R <sub>6</sub>	{2}	{golden}
R <sub>7</sub>	{2}	{golden, normal}
R <sub>8</sub>	{2,3}	{golden, normal}
R <sub>9</sub>	{3}	{normal}
R <sub>10</sub>	{3}	{normal}

**Table 5.2** An ordered multi-valued and multi-labeled dataset

Record	Weight	Class label(s)
R <sub>1</sub>	{60, 44}	{health, sick}
R <sub>2</sub>	{60,46}	{health, sick}
R <sub>3</sub>	{60,56}	{health, health}
R <sub>4</sub>	{60,66}	{health, health}
R <sub>5</sub>	{60,72}	{health, sick}
R <sub>6</sub>	{60,80}	{health, sick}
R <sub>7</sub>	{60,81}	{health, sick}
R <sub>8</sub>	{70,68}	{health, health}
R <sub>9</sub>	{70,76}	{health, health}
R <sub>10</sub>	{70,82}	{health, sick}

## 5.2 Ordered Multi-valued and Multi-labeled Discretization Algorithm

### 5.2.1 A New Discretization Metric

Before we formally introduce OMMD, we first discuss the discretization metric used in OMMD. In our knowledge, CAIM and FCAIM are the best splitting discretization algorithms since empirical evaluations show that their discretization schemes can generally maintain the highest interdependence between target class and discretized attributes, result to the least number of generated rules, and attain the highest classification accuracy [43][44]. However, both CAIM and FCAIM have two drawbacks. First of all, CAIM and FCAIM give a high factor to the number of generated intervals when they discretize a continuous attribute. Recall the discretization metric used in CAIM and FCAIM in Formula 2.1, we can find that caim discretization metric has the variable  $I$  in the denominator, where  $I$  denotes the number of the intervals  $I$ . Thus, CAIM and FCAIM usually generate a simple discretization scheme in which the number of intervals is very close to the number of class labels. Secondly, we can find that Formula 2.1 considers only major labels and ignores all the other labels. Such a consideration ignores the true data distribution and would decrease the quality of the produced discretization scheme in some cases.

**Example 5.1:** We use the age dataset in Table 5.3 as the training data to describe the first disadvantage of CAIM. The discretization scheme of CAIM is presented in Table 5.4. Form Table 5.4 we can find CAIM divide the age dataset into three intervals: [3.00,10.50], (10.50,61.50], and (61.50,71.00]. Interval [3.00,10.50] contains instances 1 to 3, interval (10.50,61.50] contains instances 4 to 12, and interval (61.50,71.00] has instances 13 to 15.

However, this discrete result is not good; obviously, the age dataset should be discretized into five intervals: instances 1 to 3, 4 to 6, 7 to 9, 10 to 12, and 13 to 15. If a classifier learning with this discretized dataset produced by CAIM, the accuracy would be worse. It is also worthy to note that a discretization scheme containing fewer intervals might result in a larger decision tree since more internal nodes might be required to split the training data.

**Table 5.3** Age dataset.

ID	Age	Class label
1	3	Care
2	5	Care
3	6	Care
4	15	Edu
5	17	Edu
6	21	Edu
7	35	Work
8	45	Work
9	46	Work
10	51	Edu
11	56	Edu
12	57	Edu
13	66	Care
14	70	Care
15	71	Care

**Table 5.4** The discretization scheme of age dataset generated by CAIM.

Class \ Interval	[3.00,10.50]	(10.50,61.50)	(61.50,71.00]	Sum
Care	3	0	3	6
Edu	0	6	0	6
Work	0	3	0	3
Sum	3	9	3	15

**Example 5.2:** Another problem of CAIM is illustrated in Table 5.5 and 5.6. For the Interval  $I_1$  of both Dataset  $D_1$  and  $D_2$ , since CAIM discrete formula uses only the 5 instances belonging to Class  $C_1$  to compute the *caim* value (the two instances with Class  $C_2$  and the three instances with Class  $C_3$  are ignored), the two datasets have the same *caim* value in spite of the different

data distribution. Such an unreasonable condition also occurs when the famous CAIR standard is considered. As shown in Table 5.7, the two datasets  $D_3$  and  $D_4$  have the same *caim* value even if their *cair* value is different.

**Table 5.5** Two datasets with equal *caim* values but different data distribution.

Dataset D <sub>1</sub> : $caim(I_1) = caim(I_2) = 2.5$			
Class \ Interval	I <sub>1</sub>	I <sub>2</sub>	Sum
C <sub>1</sub>	5	5	10
C <sub>2</sub>	2	3	5
C <sub>3</sub>	3	2	5
Sum	10	10	20

Dataset D <sub>2</sub> : $caim(I_1) = caim(I_2) = 2.5$			
Class \ Interval	I <sub>1</sub>	I <sub>2</sub>	Sum
C <sub>1</sub>	5	5	10
C <sub>2</sub>	1	4	5
C <sub>3</sub>	4	1	5
Sum	10	10	20

**Table 5.6** Two datasets with equal *caim* values but different *cair* values

Dataset D <sub>3</sub> :			
$caim(I_1) = caim(I_2) = 5;$			
$cair(I_1) = cair(I_2) = 0.$			
Class \ Interval	I <sub>1</sub>	I <sub>2</sub>	Sum
C <sub>1</sub>	5	5	10
C <sub>2</sub>	0	0	0
Sum	5	5	10

Dataset D <sub>4</sub> :			
$caim(I_1) = caim(I_2) = 5;$			
$cair(I_1) = cair(I_2) = 1.$			
Class \ Interval	I <sub>1</sub>	I <sub>2</sub>	Sum
C <sub>1</sub>	5	0	5
C <sub>2</sub>	0	5	5
Sum	5	5	10

In order to generator more reasonable discretization schemes, we use the following discretization metric.

$$C' = \{1 + [((\sum_{i=1}^S \sum_{r=1}^I \frac{n_{ir}^2}{N_{i+} N_{+r}}) - 1) / \log(I)]\}^{1/2}. \quad (5.1)$$

This metric is inspired by the statistical contingency coefficient, which has been theoretically proven that it is a good metric to measure the strength of dependence between variables. Compared to the metric used in CAIM, an obvious advantage of Formula 5.1 is that it indeed considers the label distribution of all records. Please note, we do not directly use contingency coefficient but instead, we modify it by adding  $\log(I)$  in the denominator. This modification is motivated by two reasons: 1) speed up the discretization process; 2) a discretization scheme

containing too many intervals could suffer from an overfitting problem. Actually, CAIM also take these reasons into account, but CAIM makes its discretization schemes unreasonable due to the huge influence of variable  $I$ .

## 5.2.2 Discretizing Continuous Attributes and the Computational Complexity

In our knowledge, CDR-Tree is the first algorithm to mine ordered multi-valued and multi-labeled datasets. Without loss of generality, here we use ordered two-valued and two-labeled data ( $\{v_1, v_2\}, \{l_1, l_2\}$ ) to illustrate how OMMD discretize continuous attributes. First, OMMD sorts data according to  $v_1$  and then  $v_2$ . After sorting, continuous attributes are in the form of  $\{v_1, v_2\}, \dots, \{v_1, v_p\}, \dots, \{v_q, v_2\}, \dots, \{v_q, v_p\}, \dots$ , where  $v_2 < v_p, v_1 < v_q$ . The dataset in Table 5.2 has been sorted according to this principle. After sorting, iterative splitting discretization by using Formula 5.1 is applied to records whose  $v_1$  values are identical. In other words, records are split into  $|v_1|$  subsets and discretization is carried out in each subset by referring to  $v_2$  and the label sets.

**Example 5.3:** In Table 5.2, there are two subsets  $\{R_1, R_2, R_3, R_4, R_5, R_6, R_7\}$  and  $\{R_8, R_9, R_{10}\}$ . The discretization scheme after applying iterative splitting discretization is shown in Table 5.7.

**Table 5.7** The discretization scheme of Table 5.2 after iterative splits

interval	[60,44~51]	[60,52~69]	[60,70~81]	[70,68~79]	[70,80~86]
variance	[-16,-9]	[-8, +9]	[+10, +21]	[-2, +9]	[+10, +16]
label set	{h, s}	{h, h}	{h, s}	{h, h}	{h, s}



Moreover, a merging procedure is implemented after iterative splits in OMMD. This merging procedure is inspired by the observation that there are *variant attributes* in real data. A variant attribute means that label sets can be identified by using variance of attribute values. The formal definition of variant attributes is given in Definition 5.3. As mentioned in Chapter 1, a good discretization algorithm should produce a concise summarization of attributes and to help the experts and users understand the data more easily. This merging procedure enables OMMD to produce discretization schemes which are simpler and easily understood by users. In our real life, there are many variant attributes such as weight, blood pressure, body temperature and so on. For example, if two people of different body temperatures demonstrate an abnormal increase synchronously, both of them are sick.

**Definition 5.3:** Given an ordered and inseparable multi-valued and multi-labeled dataset  $D$  and a continuous attribute  $A$  in  $D$ . Let  $V = \{v_i \mid i = 1, \dots, p; p \geq 1\}$  be the domain of  $A$  and  $L = \{l_j \mid j = 1, \dots, q; q \geq 1\}$  be the set of all class labels, we can get that records in  $D$  are in the form of  $(\{v\}, \{l\})$ , where  $v' \subset V, l' \subset L$  and  $|v'| = |l'|$ . Suppose that there are records  $\in D$  which have a identical label set  $\{l'\}$ , we can get  $min_o \leq v_o - v_{o-1} \leq max_o$ , where  $v_o$  is the  $o$ -th value in  $v'$  and  $min_o$  and  $max_o$  respectively denotes the minimal and maximal variance between  $v_o$  and  $v_{o-1}$  of these records. If there does not exist a record  $R$  satisfies  $min_o \leq v_o - v_{o-1} \leq max_o$  but its label set is  $\{l'\}$ , the attribute  $A$  is called a *variant attribute*.

**Example 5.4:** Take Table 5.7 as the example again, the discrimination scheme after merging is shown in Table 5.8. The discrimination scheme is output in the form of  $[var_{i-1}, var_i] \cup \dots$ , which means that records with variance between  $var_{i-1}$  and  $var_i$  have the identical label set. In Table 5.8, records which have an increase from 10 to 21 kilograms and a decrease from 9 to 16 kilograms in weight have the same label set {health, sick}. Similarly, records which have a variance from -8 to +9 kilograms can be grouped together. Obviously, the discretization

scheme in Table 5.8 is simpler and more meaningful to users than that in Table 5.7.

**Table 5.8** The discretization scheme of Table 5.2 after merging

variance	$[-16, -9] \cup [+10, +21]$	$[-8 \sim +9]$
label	$\{h, s\}$	$\{h, h\}$

Below is the pseudocode of OMMD for discretizing continuous attributes in an ordered two-valued and two-labeled dataset. It is worth to note that instead of adopting a simple greedy search method as that used in CAIM, NOMMD uses *simulated annealing* approach to reduce the probability of sticking on local maxima. The complexity of NOMMD for discretizing a continuous attribute is estimated as follows. The computational complexity of sort in Line 2 is  $O(N \log N)$ , where  $N$  is the number of records in  $D$ . Divide a continuous attribute into  $S$  subsets in Line 3 and the initial setups in Lines 4 and 5 require  $O(N)$ . Similar to OMMD, the expected running time of iterative partition from Line 6 to Line 32 and the expected running time can be estimated as  $O(S \cdot N / S \cdot L^3) = O(N)$ , where  $S$  is the number of subsets. Compared to NOMMD, the extra time required by OMMD is caused by the merging procedure in Line 35 to Line 42. Intuitively, this merging procedure takes  $O(N^3)$  time, which is the computational complexity required by typical merging discretization algorithms such as the original ChiMerge. Since this merging procedure works on splitting discretization schemes, it can be optimized to  $O(N)$ . Consider the discretization scheme generated after iterative partition from Line 6 to Line 32, there are two important properties of the produced discretization schemes.

1. The variances of discretization intervals in each subset can be regarded as a list consists of ordered values  $(var_0, var_1, \dots, var_i)$ , where  $var_i$  is a variance values.
2. For each list, the label distribution of any two adjacent intervals is different.

With the two proprieties, it is easily to prove that  $S$  discretization intervals in different subset

can be merged together in one step if and only if

1. All  $S$  intervals have identical label distribution.
2. Intervals which do not contain the maximum variance  $max_S$  or the minimum variance  $min_S$  of subset  $S$  has the same variance interval  $[var_{i-1}, var_i]$ .
3. Intervals which contain  $max_S$  or  $min_S$  are in the form of  $[var_{i-1}, max_S]$  or  $[min_S, var_i]$ .

Therefore, the computational complexity for merging intervals in different subset from Line 35 to Line 39 can be estimated as  $O(L \cdot N)$ . Above-mentioned merging intervals are further merged in Line 40 to Line 42 if their label distribution is the same. For example, the merging of variance  $[-16, -9]$  and  $[+10, +21]$  in Table 5.7. This step requires only a small constant time. Since the number of class labels in most real data is a small constant, the total computational complexity of the merging procedure can be estimated as  $O(N)$ . As a result, the expected computational complexity of OMMD for discretizing a continuous attribute is  $O(N \log N)$ . Please note that for an inseparable continuous attributes, it is possible that only parts of values can be represented by the variance. In other words, the output discretization scheme may be in the form of  $[var_{i-1}, var_i] \cup [v_1, v_2 \sim v_{2j}] \cup \dots$

### OMMD algorithm for inseparable continuous attributes

$D$ : an non-ordered two-valued and two-labeled dataset. Records in  $D$  are in the format of  $(\{a_1, a_2\}, \{l_1, l_2\})$ ;

$i$ : the number of continuous attributes in  $D$ ;

$S$ : the number of subsets.

$L$ : the number of class labels in each subset;

$Cut[]$ : the set of possible cut-points which are the midpoints of adjacent values belonging to different class labels in a continuous attribute;

$MMDSS$ : the multi-valued and multi-labeled discretization scheme of subset  $S$ ;

$MMDS$ : a multi-valued and multi-labeled discretization scheme;

#### OMMD ( $D$ )

1. For each continuous attribute  $A_i$
2. Sort  $A_i$  according to  $a_1$  and then  $a_2$ ;
3. Divide attribute  $A_i$  into  $S$  subsets; /\*  $S = |v_l|$
4. Get the minimum value  $v_0$  and the maximum value  $v_n$  of  $a_2$  for each subset;
5.  $MMDS = \{[a_i, [v_0, v_n]]\}$ ;
6. For each subset  $S$
7. Get  $Cut[]$ ;
8.  $MMDSS = \{[v_0, v_n]\}$ ;
9.  $C' = 0$ ; /\* the  $C'$  of initial discretization scheme  $\{[v_0, v_n]\}$  is 0
10.  $I[S] = 1$ ;
11.  $MMDSS_{temp} = \{\phi\}$ ;
12.  $I_{temp} = 0$
13. For each possible cut-point in  $Cut[]$
14. Add it into  $MMDSS$ ;
15. Calculate the corresponding  $C'$  in Equation 5.1;
16. Get the cut-point  $cut$  whose  $C'$  value is the maximum  $maxC'$ ;
17.  $\Delta C = maxC' - C'$
18. If  $\Delta C' > 0$  or  $I < L$  then
19. Remove  $cut$  from  $Cut[]$ ;
20. Add  $cut$  and the cut-point(s) in  $MMDSS_{temp}$  into  $MMDSS$ ;
21.  $C' = maxC'$ ;
22.  $I[S] = I[S] + I_{temp} + 1$ ;
23.  $MMDSS_{temp} = \{\phi\}$ ;
24.  $I_{temp} = 0$
25. Goto Line 13;
26. Else
27. Remove  $cut$  from  $Cut[]$ ;
28. Add  $cut$  into  $MMDSS_{temp}$ ;
29.  $I_{temp} = I_{temp} + 1$ ;
30. Goto Line 13 with probability  $e^{\Delta C}$ ;
31. End If

32. Update *MMDS* by referring to *MMDSS*;
33. For each discretization interval in *MMDS*
34. Calculate the variances;
35. For  $S$  intervals belong to different subset in *MMDS*
36. If all intervals in  $S$  subsets have the same label distribution and have identical variance  $[var_{i-1}, var_i]$  or are in the form of  $[var_{i-1}, max_S]$  or  $[min_S, var_i]$  then
37. Merge these intervals;
38. Update *MMDS*;
39. End if
40. For variant intervals in *MMDS* which have the same label distribution
41. Merge these intervals;
42. Update *MMDS*;
43. Output *MMDS* for continuous attribute  $A_i$ ;



### 5.2.3 Discretize Categorical Attributes and the Computational Complexity

Without loss of generality, we also use ordered two-valued and two-labeled data to illustrate how OMMD discretize categorical attributes in an ordered multi-valued and multi-labeled dataset. When discretizing categorical attributes, the iterative splitting approach is infeasible since there is not an order relation among categorical values. For categorical attributes which originally have  $c$  distinct values, there are  $(c^2 - c)/2$  kinds of values in a two-valued and two-labeled dataset. In other words, there should be some categorical values which can be merged together to form a more concise summarization of categorical attributes. Unfortunately, if we use traditional merging approaches such as ChiMerge to discretize categorical attributes, the computational complexity is terrible. Recall that traditional merging strategies are designed to discretize continuous attributes. In each loop, all neighboring continuous values are evaluated to find the best two adjacent values and then the two values are merged into one interval. The computational complexity of this merging step in one loop is  $O(N^2)$ , where  $N$  is the number of records in  $D$ . The algorithm terminates when the chosen stopping criterion satisfies. When merging categorical values, we have to try all possible combinations in one loop to find the best two values since there is not an order relation among values. The computational complexity of merging two single-valued categorical values in one loop grows into  $O(N^3)$  and gets worse when discretizing multi-valued and multi-labeled categorical attributes.

In order to efficiently discretize inseparable categorical attributes, OMMD uses a novel merging strategy. OMMD uses the discretization metric in Formula 5.1 to decide if two categorical values should be merged. Two categorical values which reaches the maximum  $\Delta C$  and have an improvement of  $C'$  are merged. OMMD uses Formula 5.1 as the merging metric for three reasons. 1) As mentioned in Section 5.1, contingency coefficient has been theoretically proven a good metric to measure the strength of dependence between variables.

2) It makes OMMD a fully automated discretization algorithm since users do not need to give any predefined threshold such as significant level. 3) Most importantly, it enables OMMD to discretize inseparable categorical attributes on  $O(N)$  time if some information is kept in memory. Recall Formula 5.1, when we merge two values  $v_a$  and  $v_b$  into  $v_c$ , the variance of  $C'$  (i.e.  $\Delta C'$ ) is mainly caused by

$$(\sum n_{ia}^2 / N_{i+} \cdot N_{+a}) + (\sum n_{ib}^2 / N_{i+} \cdot N_{+b}) - (\sum n_{ic}^2 / N_{i+} \cdot N_{+c}),$$

where  $n_{ij}$  is the number of records belongs to label  $i$  and value  $v_j$ . If we calculate all possible variances in advance and record them and the corresponding quanta matrix in the memory, the problem of finding two best values can be reduced to the problem of sorting all variances  $\Delta C'$ .

The pseudocode of OMMD for discretizing categorical attributes belong to an ordered two-valued and two-labeled dataset  $D$  is shown below. Lines 1 to 7 are responsible for some initial setups. Then NOMMOD iteratively merge two categorical values from Line 8 to Line 26. The simulated annealing approach is used in Line 25 to decide the termination of the merging procedure. The time bound of this algorithm is dominated by the initial setups in Lines 4 to 6, the update of quanta matrix in Lines 16 or 24, and the update of variances and in Lines 17 or 23. The calculation of initial  $C'$  value in Line 4 requires  $O(N \cdot L)$ , where  $N$  is the number of records and  $L$  is the number of class labels in  $D$ . Since the quanta matrix of  $D$  is then recorded, the calculation of all variances in Line 5 takes  $O(c^2 \cdot L)$  time, where  $c$  is the number of categorical attributes in  $D$ . The complexity of sort in Line 6 is  $O(c^2 \log c^2)$ . Update the quanta matrix in Lines 16 or 24 requires  $O(1)$  time and update the variances in Lines 17 or 23 takes  $O(c^2)$  time. In summary, the expected computational complexity of OMMD for discretizing a categorical attribute can be reduced to  $O(N)$  if all possible  $\Delta C'$  and the corresponding quanta matrix are kept in the memory.

### OMMD algorithm for inseparable categorical attributes

$D$ : an ordered two-valued and two-labeled dataset. Records in  $D$  are in the format of  $(\{a_1, a_2\}, \{l_1, l_2\})$ ;

$c$ : the number of categorical attributes in  $D$ ;

$v_c$ : original categorical values in  $D$ ;

$QM[]$ : the quanta matrix;

$Variance[]$ : all variances of any two categorical values;

$MMDS$ : a multi-valued and multi-labeled discretization scheme;

#### OMMD ( $D$ )

1. For each categorical attribute  $A_i$
2.  $MMDS = \{[v_0], [v_1], \dots, [v_c]\}$ ; /\* the initial discretization scheme
3.  $MMDSStemp = \{\phi\}$ ;
4. Calculate the initial  $C'$  of  $MMDS$ ;
5. Calculate and Record  $Variance[]$ ;
6. Sort  $Variance[]$ ;
7. Record the initial  $QM[]$ ;
8. For each pair values in  $MMDS$
9. Find the pair which has the maximum value in  $Variance[]$
10. Calculate the corresponding  $C'$  according to Equation 1 as  $maxC'$ ;
11.  $\Delta C' = maxC' - C'$
12. If  $\Delta C' > 0$  then
13. Merge the two values;
14. Merge the pair values in  $MMDSStemp$
15. Update  $MMDS$ ;
16. Update  $QM[]$ ;
17. Update  $Variance[]$ ;
18.  $MMDSStemp = \{\phi\}$ ;
19.  $C' = maxC'$ ;
20. Goto Line 8;
21. Else
22. Regard the two values as a pair value and add it into  $MMDSStemp$ ;
23. Update  $Variance[]$ ;
24. Update  $QM[]$ ;
25. Goto Line 8 with probability  $e^{\Delta C'}$ ;
26. End If
27. Output  $MMDS$  for categorical attribute  $A_i$ ;



## 5.3 Experiment and Analysis

Here, we first compare the discretization schemes among 7 famous discretization algorithms. We then evaluate if NOMMD can improve the performance of CDR-Tree.

### 5.3.1 The Comparison of Discretization Scheme

We implement the following seven discretization algorithms in Microsoft Visual C++ 6.0 to evaluate the performance of Equation 5.1. Among the seven discretization algorithms, CACC is our approach. CACC has a same main framework to that of CAIM except that it uses a different discretization metric.

1. Equal Width and Equal Frequency: two typical unsupervised top-down methods;
2. CACC: our approach;
3. CAIM: the newest top-down method;
4. IEM: a famous and widely used top-down method;
5. ChiMerge: a typical bottom-up method;
6. Extended Chi2: the newest bottom-up approach.

Among the seven discretization algorithms, Equal Width, Equal Frequency and ChiMerge require the user to specify in advance some parameters of discretization. For the ChiMerge algorithm, we set the level of significance to 0.95. For the Equal Width and Equal Frequency methods, we adopted the heuristic formula used in CAIM to estimate the number of discrete interval. All experiments were run on a PC equipped with Windows XP operating system, Pentium IV 1.8GHz CPU, and 512mb SDRAM memory.

Our experimental data includes thirteen UCI real datasets. Seven of them were used in CAIM and the rest were gathered from the U.C. Irvine repository [59]. The details of the thirteen UCI experimental datasets are listed in Table 5.9. The 10-fold cross-validation test

method was applied to all experimental datasets. The discretization was done using the training sets and the testing sets were discretized using the generated discretization scheme. In addition, we also used C5.0 to evaluate the generated discretization schemes. In our experiments, C5.0 was chosen since it was conveniently available and widely used as a standard for comparison in machine learning literature. Finally, we used the Friedman test and the Holm’s post-hoc tests with significance level  $\alpha = 0.05$  to statistically verify the hypothesis of improved performance.

**Table 5.9** The summary of thirteen UCI real datasets

Dataset	Number of continuous attributes	Number of attributes	Number of class labels	Number of examples
breast	9	9	2	699
bupa	6	6	2	345
glass	10	10	6	214
hea	6	13	2	270
ion	32	34	2	351
iris	4	4	3	150
optdigit	64	64	10	5620
page-blocks	10	10	5	5473
pendigit	16	16	10	10992
pid	8	8	2	768
sat	36	36	6	6435
thy	6	21	3	7200
wav	21	21	3	5000

The comparisons of the generated discretization schemes are shown in Figure 5.1. Due to the content limit, we only showed for each dataset the mean of *cair* value, the mean of execution time and the mean number of discrete intervals. We used the Friedman test to check if the measured mean ranks reached statistically significant differences. If the Friedman test showed that there was a significant difference, the Holm’s post-hoc test was used to further analyze the comparisons of all the methods against CACC. Although we also showed the

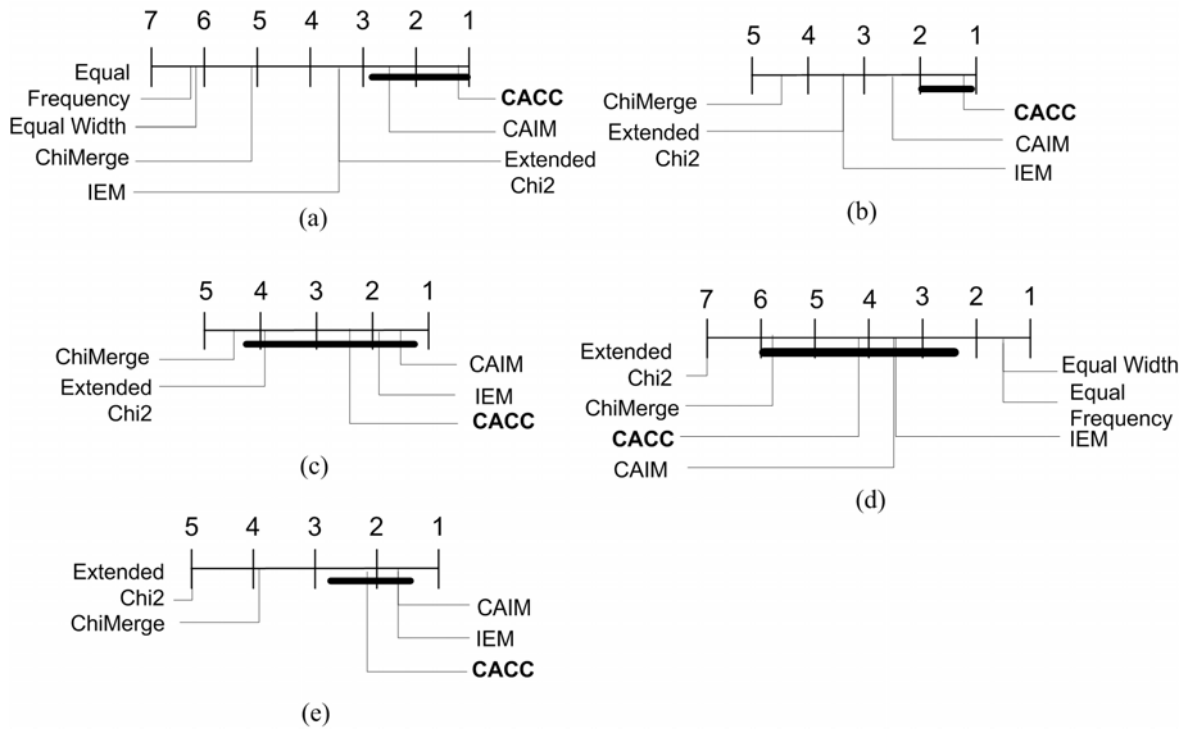
number of discrete intervals in this experiment, it was not our main concern. Recall that in the Introduction, we stated that the general goals of a discretization algorithm should be: a) generate a better discretization scheme (measured by *cair* value in Equation 2.2; b) the generated discretization scheme should lead to the improvement of accuracy and efficiency of a learning algorithm; and, c) the discretization process should be as fast as possible. A discretization scheme with fewer intervals may not only lead to a worse quality of discretization scheme and a decrease in the accuracy of a classifier, but also increase the produced rules in a classifier.

In Figure 5.1 the top line in the diagram is the axis on which we plotted the average ranks of all the methods while a method on the right side means that it performs better. A method with rank outside the marked interval in Figure 5.1 means that it is significantly different from CACC. The comparison results in Figure 5.1(a) showed that on the average, CACC reached the highest *cair* value from among the seven discretization algorithms. This was a very exhilarating result that demonstrated that the CACC criterion can indeed produce a high quality discretization scheme. The corresponding value of Friedman test was 58.714 ( $p$ -value  $< 0.0001$ ), which was larger than the threshold 12.592. From Figure 5.1(a) we can see that the mean *cair* of CACC was statistically comparable to that of CAIM and significantly better than that of all the other five methods. The comparison between CAIM and CACC did not achieve significant difference since we compared all seven algorithms. If we removed the two unsupervised algorithms from this comparison, we can obtain Figure 5.1(b) in which CACC performed significantly better than all of the other four methods. It is also worth noting that although we only showed the mean *cair* in the present paper, for all of the 228 continuous attributes in Table 5.9, the *cair* value of CACC is always equal to or better than that of CAIM.

Regarding the number of discrete intervals, on the average CAIM generated the least number of intervals. This result was not surprising since CAIM usually generated a simple discretization scheme in which the number of intervals was very close to the number of

classes. The corresponding value of Friedman test was 8.192 ( $p$ -value = 0.228), which was smaller than the threshold 12.592, and meant that there were no significant differences among the number of generated intervals of the seven algorithms. However, if we removed the two unsupervised algorithms, in which the number of generated intervals was decided in advance, from this comparison, the Friedman test reached statistical significance and we obtained Figure 5.1(c). From Figure 5.1(c), we can see that the generated number of intervals of CACC was significantly less than that of ChiMerge and comparable to that of CAIM, IEM and Extended chi2.

Finally, the two unsupervised methods were the fastest since they did not consider the processing of any class related information. The discretization time of CACC was a little longer than that of CAIM but the difference did not reach statistical significance. If we compare all seven algorithms, the Holm's post-hoc test in Figure 5.1(d) showed that CACC was significantly faster than Extended Chi2, significantly slower than Equal Width and Equal Frequency, and comparable to CAIM, IEM and ChiMerge. When we removed the two unsupervised algorithms from this comparison, we obtained a little different result as shown in Figure 5.1(e). In Figure 5.1(e), CACC was significantly faster than both bottom-up approaches Extended Chi2 and ChiMerge, and comparable to CAIM, IEM. This result corresponded to our previous discussions that the computational complexity of the bottom-up methods is usually worse than that of the top-down methods. It is also worth noting that compared to the ChiMerge algorithm, although the Extended Chi2 algorithm had a better discretization quality and generated fewer intervals, it required more execution time to check the merged inconsistency rate in every step.



**Figure 5.1** The comparison of CACC against the other discretization methods with the Holm's post-hoc tests ( $\alpha = 0.05$ ): (a) and (b) *cair* value; (c) number of intervals; (d) and (e) execution time.

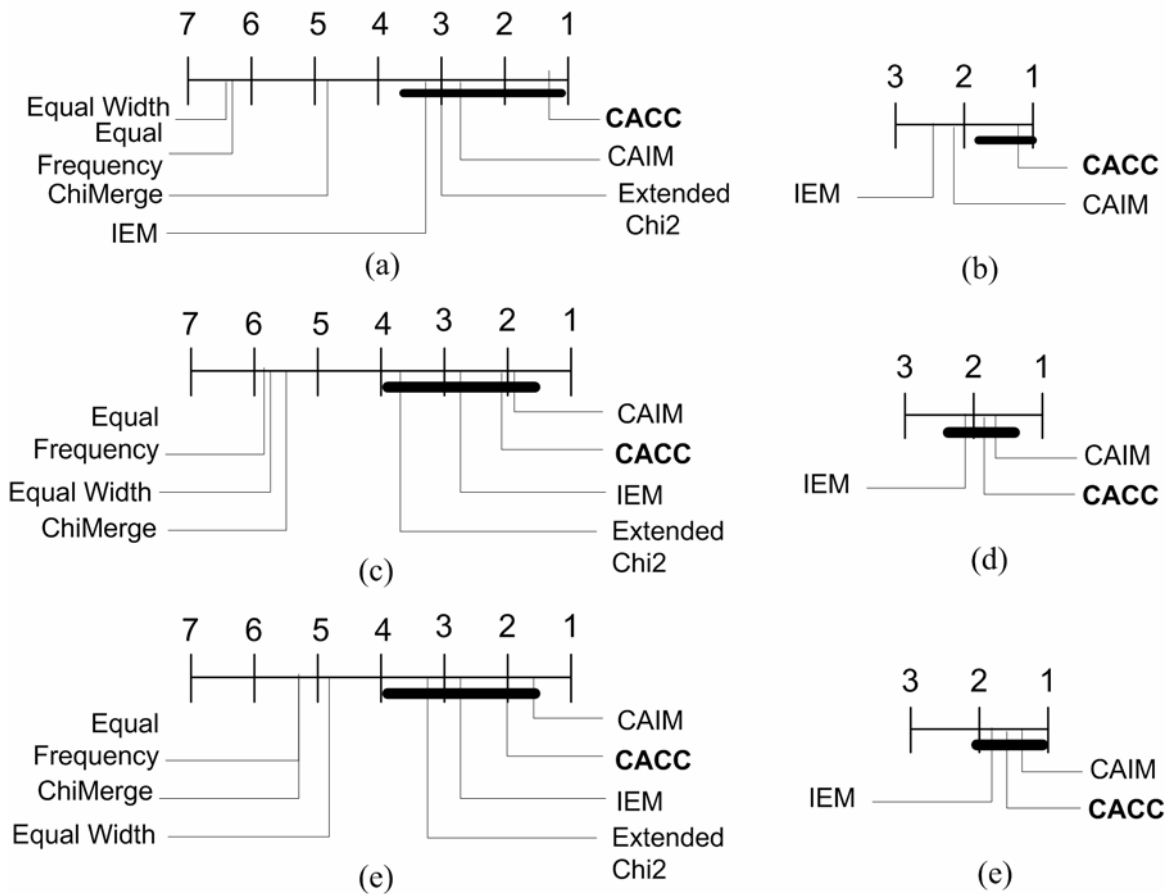
To evaluate the effect of generated discretization schemes on the performance of the classification algorithm, we used the discretized datasets to train C5.0. The testing datasets were then used to calculate the accuracy, the number of rules, and the execution time. Similarly, the Friedman test and the Holm's post-hoc tests with significance level  $\alpha = 0.05$  were used to check if these comparisons reached significant differences.

The visualizations of the Holm's post-hoc test are illustrated in Figure 5.2. The comparison results in Figure 5.2(a) show that on the average, CACC reached the highest accuracy from among the seven discretization algorithms. This was a very exhilarating result that demonstrated that the discretization schemes generated by CACC can indeed improve the accuracy of classification. In Figure 5.2(a) we can see that the accuracy of CACC was significantly better than Equal Width, Equal Frequency and ChiMerge, and comparable to

CAIM, IEM and Extended Chi2. However, when we removed the two unsupervised methods and the two slowest bottom-up methods from this comparison, we obtained a little different result. The mean rank of CACC, CAIM and IEM was 1.2, 2.3, and 2.5 respectively. The Friedman test and the Holm's post-hoc tests in Figure 5.2(b) showed that among the tree top-down approaches, the accuracy of CACC was significantly better than that of CAIM and IEM.

As regards to the number of generated rules of C5.0, the CAIM reached the best performance and CACC was ranked secondly. The Friedman test and the Holm's post-hoc tests in Figure 5.2(c) showed that C5.0 produced significantly more rules when it used the discretization schemes of ChiMerge, Equal Width and Equal Frequency. Figure 5.2(c) also showed that C5.0 generated statistically comparable numbers of rules when it used the discretization schemes of CACC, CAIM, IEM and Extended Chi2. When we only compared the three top-down approaches, the Holm's post-hoc tests also showed that there were no significant differences among them as shown in Figure 5.2(d). Note that we have stated that a discretization scheme with fewer intervals does not mean that it will result to a simpler decision tree. On the contrary, it might even increase the produced rules. Our inference can be found in this experiment. For example, CACC generated more intervals than CAIM but resulted to fewer rules in the datasets thy, wav and hea.

Finally as illustrated in Figure 5.2(e), when C5.0 used the training data discretized by CACC, CAIM, IEM and Extended Chi2, the training times were statistically comparable. C5.0 required significantly more training time when the training data were discretized by ChiMerge, Equal Width and Equal Frequency. When we only compared the three top-down approaches, the Holm's post-hoc tests also showed that there were no significant differences among CACC, CAIM and IEM.



**Figure 5.2** The comparison of C5.0 performance on CACC against C5.0 performance on the other discretization methods with the Holm’s post-hoc test ( $\alpha = 0.05$ ): (a) and (b) accuracy; (c) and (d) number of rules; (e) and (f) execution time.

### 5.3.2 The Performance Evaluation of OMMD

Finally, we evaluate the effect of discretization schemes generated by OMMD on the performance of CDR-Tree. We implement CDR-Tree and OMMD in Microsoft Visual C++ 6.0 for performance analysis. All evaluations were done on a PC equipped with Windows XP operating system, Pentium IV 3.0GHz CPU, and 512mb DDR memory. We follow the experiment setups in CDR-Tree in Section 4.3 and use the same synthetic datasets. Experiments are evaluated in accuracy, the number of rules and the building time.

For OMMD, the discretization was done using the training sets and the testing sets were

discretized using the generated discretization scheme. The comparison results are shown in Table 5.12 and a quick comparison can be obtained by checking the average in the last row. The comparison results in Table 5.9 show that OMMD significantly reduces the number of rules produced by CDT-Tree and at the same time maintains the accuracy of CDR-Tree. This result corresponds to the general goals of a discretization algorithm.

**Table 5.10** The experimental results of CDR-Tree with/without NOMMD

Function	R %	Accuracy (%)		Number of rules	
		CDR-Tree	OMMD	CDR-Tree	OMMD
F3	5	91.5	90.4	133	108
	10	88.7	88.7	142	104
	15	88.7	86.8	166	112
	20	87.5	87.8	158	116
	30	85.9	86.1	182	134
F43	5	89.6	88.6	121	98
	10	87.6	87.8	123	106
	15	85.4	85.3	132	112
	20	85.7	85.3	147	128
	30	84.3	83.9	155	126
F5	5	90.5	90.8	109	80
	10	89.1	89.1	125	88
	15	87.4	88.1	122	92
	20	86.3	86.2	148	109
	30	83.7	83.5	160	128
F45	5	87.6	85.8	224	182
	10	86.2	84.9	231	186
	15	85.8	84.4	252	189
	20	83.1	83.8	264	210
	30	81.7	80.2	276	232
<i>average</i>		<b>86.8</b>	<b>86.4</b>	<b>168.5</b>	<b>132.0</b>



# Chapter 6

## Conclusions and Future Work

### 6.1 Conclusions

Mining data streams has become a novel research topic of growing interest in the field of data mining. In this dissertation, we aim to improve the performance of classification on mining concept-drifting data streams. We first propose SCRIPT to sensitively and accurately build decision trees for concept-drifting data streams. Based on the variation of CDAVs (class distribution on attribute values), SCRIPT can be applied to large scale and high speed applications which also require the sensitivity to handle drifting concepts. Proposition 3.1 verifies the validity of SCRIPT to determine if concept drift exists between two data blocks. The experiments in Section 3.3 show that SCRIPT can sensitively, accurately, and efficiently handle the drifting concepts in data streams.

Although SCRIPT can sensitively and efficiently handle the concept-drifting problem in data streams, as with all proposed approaches to concept drift, it focuses on updating the classification model to accurately predict incoming data and is unable to elucidate the main causes of concept drifts. In some real applications, users might be more interested in the rules of concept drift. To solve this problem, we then propose CDR-Tree to elucidate concept-drifting rules. CDR-Tree can not only produce drifting rules, but also efficiently

extract the classification model of each data block for decision makers, resulting in wide applicability. The experimental results in Section 4.3 show the accuracy of the CDR-Tree and the efficiency and accuracy of our extraction strategy.

Finally, in order to reduce the construction time of CDR-Tree and reduce the rules generated by CDR-Tree, we shift our attention to discretization techniques. Discretization is an important preprocessing technique for data mining algorithms which are highly sensitive to the size of data. It is also crucial to learning methods which can only handle categorical attributes. Unfortunately, proposed discretization algorithms have been designed to discretize single-valued and single-labeled data and therefore are all infeasible for the discretization of the multi-valued and multi-labeled data used in CDR-Tree. We propose OMMD as the solution to discretize the input data of CDR-Tree. OMMD uses the simulated annealing search and a new discretization metric, which is based on the statistical contingency coefficient, to generate high quality discretization schemes. OMMD also combines the ideas of merging and splitting discretization techniques to further simplify discretization schemes. Empirical evaluations in Section 5.3 show that our new discretization metric is superior to other state-of-the-art discretization algorithms in that it can produce a better discretization scheme to attain improvement in the accuracy of C5.0. Concerning the execution time of discretization, the number of generated rules, and the execution time of C5.0, our discretization metric also achieves promising results. Experiments in Section 5.3 show that OMMD is an accurate and effective discretization algorithm since it significantly reduces the number of rules produced by CDR-Tree and at the same time maintains the accuracy of CDR-Tree.

## 6.2 Future Work

There are still many issues worth further investigation. First of all, although SCRIPT can efficiently and accurately amend the original classifier when the concepts of instances drift, the resulting decision tree does not guarantee to be the same to the one built from scratch. It would be interesting to find an efficient way by which one can obtain an identical decision tree. In addition, when there are two-way drifts, we need not to amend the original sub-trees by use of incoming instances, but switch the classification rules. Therefore, further analyzing the drifting conditions and proposing a more efficient and accurate correction mechanism is considered in our future work.

Secondly, CDR-Tree considers the cases in which there are only two data blocks in a data stream. If analysis of greater than two is required, CDR-Trees will become much larger and more complicated. Therefore, another future focus is to extend the use of CDR-Tree which can more efficiently process multi-block concept-drifting problems. Besides, although our extraction method can efficiently extract the classification model from CDR-Trees and the extracted model can reach accuracy comparable to the decision tree built from the beginning, it would be interesting to find a way by which the extracted decision tree is the same to that built from the beginning.

Finally, OMMD is only applicable to ordered multi-valued and multi-labeled datasets. However, there are non-ordered multi-valued and multi-labeled data such as the input data of MMC (multi-valued and multi-labeled classifier) [10] and MMDT (multi-valued and multi-labeled decision tree) [8]. Since many available datasets in our real life are ordered multi-valued and multi-labeled, we intend to design an ordered multi-valued and multi-labeled discretization algorithm in the future.

# Bibliography

- [1] R. Agrawal, S. Ghosh, T. Imielinski, B. Iyer and A. Swami, “An interval classifier for database mining applications”, in *Proceedings of the 18th International Conference on Very Large Databases*, pp. 560-573, 1992.
- [2] R. Agrawal, T. Imielinski and A. Swami, “Database mining: a performance perspective”, *IEEE Transactions on Knowledge and Data Engineering*, vol. 5, no. 6, pp. 914-925, 1993.
- [3] S. D. Bay, “Multivariate discretization of continuous variables for set mining,” in: *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 315-319, 2000.
- [4] F. Berzal, J. C. Cubero, N. Marín and D. Sánchez, “Building multi-way decision trees with numerical attributes,” *Information Sciences*, vol. 165, no. 1-2, pp. 73-90, 2004.
- [5] A. Blum, “Empirical support for winnow and weighted-majority algorithms: results on a calendar scheduling domain,” *Machine Learning*, vol. 26, pp. 5-23, 1997.
- [6] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*, Wadsworth, 1984.
- [7] S. Chao and Y. Li, “Multivariate interdependent discretization for continuous attribute,” in: *Proceedings of the Third International Conference on Information Technology and Applications*, Vol. 1, pp. 167-172, 2005.
- [8] S. Chou and C. L. Hsu, “MMDT: a multi-valued and multi-labeled decision tree classifier for data mining,” *Expert System with Application*, vol. 28, no. 4, pp. 799-812, 2005.

- [9] N. V. Chawla, L. O. Hall, K. W. Bowyer, T. E. Moore and W. P. Kegelmeyer, "Distributed pasting of small Votes," *Multiple Classifier Systems*, pp. 52-61, 2002.
- [10] Y. L. Chen, C. L. Hsu and S. C. Chou, "Constructing a multi-valued and multi-labeled decision tree," *Expert Systems with Applications*, vol. 25, no. 2, pp. 199-209, 2003.
- [11] J. Y. Ching, A. K. C. Wong and K. C. C. Chan, "Class-dependent discretization for inductive learning from continuous and mixed mode data," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 17, no. 7, pp. 641-651, 1995.
- [12] D. Chiu, A. Wong and B. Cheung, "Information discovery through hierarchical maximum entropy discretization and synthesis," *Knowledge Discovery in Databases*, G. Piatetsky-Shapiro and W. J. Frawley (EDs.), MIT/AAAI Press, pp. 125-140, 1991.
- [13] K. J. Cios and L. A. Kurgan, "CLIP4: hybrid inductive machine learning algorithm that generates inequality rules," *Information Science*, vol. 163, no. 1-3, pp. 37-83, 2004.
- [14] K. J. Cios and L. A. Kurgan, "Hybrid inductive machine learning: an overview of clip algorithms," *New Learning Paradigms in Soft Computing*, L.C. Jain and J. Kacprzyk (EDs.), Physica-Verlag (Springer), pp. 276-322, 2001.
- [15] P. Clark and T. Niblett, "The CN2 induction algorithm," *Machine Learning*, vol. 3, no. 4, pp. 261-283, 1989.
- [16] W. Cohen, "Learning rules that classify e-mail," in: *Proceedings of the AAAI Spring Symposium on Machine Learning in Information Access*, Menlo Park, CA, AAAI Press, Technical Report SS96-05, pp. 18-25, 1996.
- [17] P. Cunningham, and N. Nowlan, "A case-based approach to spam filtering that can track concept drift," in: *Proceedings of the ICCBR Workshop on Long-Lived CBR Systems*, 2003.
- [18] P. Domingos and G. Hulten, "Mining high-speed data streams," in: *Proceedings of 6th International Conference on Knowledge Discovery and Data Mining*, pp. 71-80, 2000.
- [19] J. Dougherty, R. Kohavi and M. Sahami, "Supervised and unsupervised discretization of

- continuous features,” in: *Proceeding of the 12th International Conference on Machine Learning*, pp. 194-202, 1995.
- [20] T. Elomaa, J. Kujala and J. Rousu, “Practical approximation of optimal multivariate discretization,” in: *Proceedings of the 16th International Symposium on Foundations of Intelligent Systems*, pp. 612-621, 2006.
- [21] H. Fan and K. Ramamohanarao, “Fast discovery and the generalization of strong jumping emerging patterns for building compact and accurate classifiers,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 6, pp. 721-737, 2006.
- [22] W. Fan, “Systematic data selection to mine concept-drifting data streams,” in: *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 128-137, 2004.
- [23] U. M. Fayyad and K. B. Irani, “Multi-interval discretization of continuous-valued attributes for classification learning,” in: *Proceeding of the 13th International Conference on Artificial Intelligence*, pp. 1022-1027, 1993.
- [24] U. M. Fayyad and K. B. Irani, “On the handling of continuous-valued attributes in decision tree generation,” *Machine Learning*, vol. 8, pp. 87-102, 1992.
- [25] S. Ferrandiz and M. Boullé, “Multivariate discretization by recursive supervised bipartition of graph,” in: *Proceedings of the 4th International Conference on Machine Learning and Data Mining in Pattern Recognition*, pp. 253-264, 2005.
- [26] A. A. Freitas, “Understanding the crucial differences between classification and discovery of association rules,” *SIGKDD Explorations*, vol. 2, no. 1, pp. 65-69, 2000.
- [27] J. Furnkranz and G. Widmer, “Incremental reduced error pruning,” in: *Proceedings of the 11th International Conference on Machine Learning*, pp. 70–77, 1994.
- [28] J. Gehrke, R. Ramakrishnan and V. Ganti, “RainForest: a framework for fast decision tree construction of large datasets,” *Data Mining and Knowledge Discovery*, vol. 4, no. 2/3, pp. 127-162, 2000.

- [29] D. Gómez, J. Montero and J. Yáñez, “A coloring fuzzy graph approach for image classification,” *Information Sciences*, vol. 176, no. 24, pp. 3645-3657, 2006.
- [30] J. Han and M. Kamber, *Data Mining: Concepts and Techniques*, Morgan Kaufmann Publisher, 2001.
- [31] M. B. Harries, C. Sammut and K. Horn, “Extracting hidden context,” *Machine Learning*, vol. 32, no.2, pp. 101-126, 1998.
- [32] G. Hulten, L. Spencer and P. Domingos, “Mining time-changing data streams,” in: *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 97-106, 2001.
- [33] R. Jin and G. Agrawa, “Efficient decision tree construction on streaming data,” in: *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 571-576, 2003.
- [34] N. Japkowicz and S. Stephen, “The class imbalance problem: a systematic study,” *Intelligent Data Analysis*, vol. 6, no. 5, pp. 429-450, 2002.
- [35] K. A. Kaufman and R. S. Michalski, “Learning from inconsistent and noisy data: the AQ18 approach,” in: *Proceeding of the 11th International Symposium on Methodologies for Intelligent Systems*, 1999.
- [36] R. Kerber, “ChiMerge: discretization of numeric attributes,” in: *Proceeding of the 9th International Conference on Artificial Intelligence*, pp. 123-128, 1992.
- [37] D. Kifer, S. Ben-David and J. Gehrke, “Detecting change in data streams,” in: *Proceedings of the 30th International Conference on Very Large Databases*, pp. 180-191, Toronto, Canada, 2004.
- [38] R. Klinkenberg, “Learning drifting concepts: example selection vs. example weighting,” *Intelligent Data Analysis*, vol. 8, no. 3, pp. 281-300, 2004.
- [39] R. Klinkenberg and I. Renz, “Adaptive information filtering: learning in the presence of concept drifts,” in: *Proceedings of International Conference on Machine Learning*, pp.



33-40, Menlo Park, California, 1998.

- [40] J. Z. Kolter and M. A. Maloof, "Dynamic weighted majority: a new ensemble method for tracking concept drift," in: *Proceedings of the 3rd International IEEE Conference on Data Mining*, pp. 123-130, Melbourne, FL, 2003.
- [41] I. Koychev, "Gradual forgetting for adaptation to concept drift," in: *Proceedings of ECAI 2000 Workshop in Spatio-Temporal Reasoning*, Berlin, Germany, 2000.
- [42] A. Kuh, T. Petsche and R. L. Rivest, "Learning time-varying concepts," *In Advances in Neural Information Processing Systems 3*, vol. 3, San Francisco, CA: Morgan Kaufmann, pp. 183-189, 1991.
- [43] L. Kurgan and K. J. Cios, "Fast class-attribute interdependence maximization (CAIM) discretization algorithm," in: *Proceeding of International Conference on Machine Learning and Applications*, pp. 30-36, 2003.
- [44] L. Kurgan and K. J. Cios, "CAIM discretization algorithm," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 2, pp. 145-153, 2004.
- [45] T. Lane and C. E. Brodley, "Approaches to online learning and concept drift for user identification in computer security," in: *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining*, pp. 259-263, New York, 1998.
- [46] M. Lazarescu and S. Venkatesh, "Using multiple windows to track concept drift," *Intelligent Data Analysis Journal*, vol. 8, no. 1, pp. 29-59, 2004.
- [47] C. I. Lee, C. J. Tsai, T. Q. Wu and W. P. Yang, "A multi-relational classifier for imbalanced database," *Expert Systems with Applications*, accepted, to appear in 36(3).
- [48] C. I. Lee, C. J. Tsai and C. W. Ku, "An evolutionary and attribute-oriented ensemble classifier," in: *Proceedings of International Conference on Computational Science and its Applications*, pp. 1210-1218, 2006.
- [49] H. Liu, F. Hussain, C.L. Tan and M. Dash, "Discretization: an enabling technique," *Journal of Data Mining and Knowledge Discovery*, vol. 6, no. 4, pp. 393-423, 2002.



- [50] H. Liu and R. Setiono, "Feature selection via discretization," *IEEE Transactions on Knowledge and Data Engineering*, vol. 9, no. 4, pp. 642-645, 1997.
- [51] M. A. Maloof and R. S. Michalski, "Incremental learning with partial instance memory," *Artificial Intelligence*, vol. 154, no. 1-2, pp. 95-126, 2004.
- [52] M. A. Maloof, "Incremental rule learning with partial instance memory for changing concepts," in: *Proceedings of the International Joint Conference on Neural Networks*, pp. 2764-2769, Los Alamitos, CA, IEEE Press, 2003
- [53] M. A. Maloof and R.S. Michalski, "Selecting examples for partial memory learning," *Machine Learning*, vol. 41, no. 1, pp. 27-52, 2000.
- [54] M. Mehta, R. Agrawal and J. Rissanen, "SLIQ: a fast scalable classifier for data mining," in: *Proceedings of the 5th International Conference on Extending Database Technology*, pp. 18-32, 1996.
- [55] M. Mehta, J. Rissanen, and R. Agrawal, "MDL-Based Decision Tree Pruning," in: *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*, pp. 216-221, 1995.
- [56] S. Mehta, S. Parthasarathy and H. Yang, "Correlation preserving discretization data mining," in: *Proceeding of the 4th IEEE International Conference on Data Mining*, pp. 479-482, 2004.
- [57] T. Menzies, "Data mining for very busy people," in: *Proceedings of the International IEEE Conference on Data Mining*, pp. 22-29, 2003.
- [58] R. S. Michalski, I. Mozetic, J. Hong and N. Lavrac, "The multipurpose incremental learning system AQ15 and its testing application to three medical domains," in: *Proceeding of the 5th National Conference on Artificial Intelligence*, pp. 1041-1045, 1986.
- [59] P. M. Murphy and D. W. Aha, "*UCI Repository of Machine Learning Databases*," Irvine, CA: University of California, Department of Information and Computer Science, 1992.

- [60] A. Paterson and T. B. Niblett, *ACLS Manual*, Edinburgh: Intelligent Terminals, Ltd, 1987.
- [61] B. Pfahringer, "Compression-based discretization of continuous attributes," in: *Proceeding of the 12th International Conference on Machine Learning*, pp. 456-463, 1995.
- [62] J. R. Quinlan, *C4.5: Program for Machine Learning*, Morgan Kaufmann Publisher, San Mateo, CA, 1993.
- [63] J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, no. 1, pp. 81-106, 1986.
- [64] R. Rastogi and K. Shim, "PUBLIC: a decision tree classifier that integrates building and pruning," in: *Proceedings of the 24th International Conference on Very Large Databases*, pp. 404-415, 1998.
- [65] J. C. Shafer, R. Agrawal and M. Mehta, "SPRINT: a scalable parallel classifier for data mining," in: *Proceedings of the 22th International Conference on Very Large Databases*, pp. 544-555, 1996.
- [66] J. C. Schlimmer and D. H. Fisher, "A case study of incremental concept induction," in: *Proceedings of the 5th International Conference on Artificial Intelligence*, pp. 496-501, Philadelphia, PA, 1986 .
- [67] J. C. Schlimmer and R. H. Granger, "Beyond incremental processing: tracking concept drift," in: *Proceedings of 5th National Conference on Artificial Intelligence*, pp. 502-507, Philadelphia, PA., 1986.
- [68] W. Street and Y. Kim, "A streaming ensemble algorithm for large-scale classification," in: *Proceedings of 7th International Conference on Knowledge Discovery and Data Mining*, pp. 377-382, New York, 2001.
- [69] C. T. Su and J. H. Hsu, "An extended chi<sup>2</sup> algorithm for discretization of real value attributes," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 3, pp.

437-441, 2005.

- [70] F. Tay and L. Shen, "A modified chi2 algorithm for discretization," *IEEE Transactions on Knowledge and Data Engineering*, vol. 14, no. 3, pp. 666-670, 2002.
- [71] P. E. Utgoff, "Incremental induction of decision trees," *Machine Learning*, vol. 4, no. 2, pp. 161-186, 1989.
- [72] P. E. Utgoff, N. C. Berkman and J. A. Clouse, "Decision tree induction based on efficient tree restructuring," *Machine Learning*, vol. 29, no. 1, pp. 5-44, 1997.
- [73] H. Wang, W. Fan, P. S. Yu and J. Han, "Mining concept-drifting data streams using ensemble classifiers" in: *Proceedings of 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 226-235, Washington, DC, 2003.
- [74] L. Wang, H. Zhao, G. Dong and J. Li, "On the complexity of finding emerging patterns," *Theoretical computer science*, vol. 335, no. 1, pp. 15-27, 2006.
- [75] G. Widmer and M. Kubat, "Learning in the presence of concept drift and hidden contexts," *Machine Learning*, vol. 23, no. 1, pp. 69-101, 1996.
- [76] A. K. C. Wong and D. K. Y. Chiu, "Synthesizing statistical knowledge from incomplete mixed-mode data," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 9, pp. 796-805, 1987.
- [77] Q. X. Wu, D. A. Bell, T. M. McGinnity, G. Prasad, G. Qi and X. Huang, "Improvement of decision accuracy using discretization of continuous attributes," in: *Proceedings of the Third International Conference on Fuzzy Systems and Knowledge Discovery*, pp. 674-683, Lecture Notes in Computer Science 4223, 2006.
- [78] S. Zadrozny and J. Kacprzyk, "Computing with words for text processing: an approach to the text categorization," *Information Sciences*, vol. 176, no. 4, pp. 415-437, 2006.