Available online at www.sciencedirect.com
ScienceDirect
Expert Systems with Applications 36 (2009) 1164–1178
Expert Systems with Applications
www.elsevier.com/locate/eswa

Mining decision rules on data streams in the presence of concept drifts

Cheng-Jung Tsai [a,*], Chien-I. Lee [b], Wei-Pang Yang [c]

[a] Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan, ROC
[b] Department of Information and Learning Technology, National University of Tainan, Tainan, Taiwan, ROC
[c] Department of Information Management, National DongHwa University, Hualien, Taiwan, ROC

Abstract

In a database, the concept of an example might change along with time, which is known as concept drift. When the concept drift occurs, the classification model built by using the old dataset is not suitable for predicting a new dataset. Therefore, the problem of concept drift has attracted a lot of attention in recent years. Although many algorithms have been proposed to solve this problem, they have not been able to provide users with a satisfactory solution to concept drift. That is, the current research about concept drift focuses only on updating the classification model. However, real life decision makers might be very interested in the rules of concept drift. For example, doctors desire to know the root causes behind variation in the causes and development of disease. In this paper, we propose a concept drift rule mining tree, called CDR-Tree, to accurately discover the underlying rule governing concept drift. The main contributions of this paper are: (a) we address the problem of mining concept-drifting rules which has not been considered in previously developed classification schemes; (b) we develop a method that can accurately mine rules governing concept drift; (c) we develop a method that should classification models be required, can efficiently and accurately generate such models via a simple extraction procedure rather than constructing them anew; and (d) we propose two strategies to reduce the complexity of concept-drifting rules mined by our CDR-Tree.

Keywords: Data mining; Classification; Decision tree; Data stream; Concept drift

1. Introduction

With the rapid development and large distribution of electronic data, extracting useful information from many numerous and jumbled sources has become a major goal of many scholars. *data mining* (Han & Kamber, 2001; Menzies, 2003), an important technique for extracting information from databases, has been proposed to solve this problem. Among the several functions of data mining, *classification* (Lee, Tsai, Wu, & Yang, 2008; Tsai, Lee, Chen, & Yang, 2007) is crucially important and has been applied successfully to several areas; including the discovery of commodity deal dependence, customer relationship management, risk analysis, etc. A standard classification task can be divided into two steps. In the first step, a *training dataset* is given. Each example in the training dataset contains a number of attributes and a *target class*. Attributes can be classified into *continuous attributes* and *categorical attributes*. The main difference between them is that the relations are ordered in the continuous attributes, but not in the categorical attributes. With the given training dataset, a classification system will generate a classifier to demonstrate the relations between the attributes and the target class in that training dataset. Then, the second step is to evaluate the accuracy of the generated classifier from the first step by using a *testing dataset*. The famous techniques developed for classification include: Bayesian classification,

* Corresponding author. Present address: 4F., No. 167, Sec. 1, Funong St., East District, Tainan 70175, Taiwan, ROC. Tel.: +886 6 2133111x777; fax: +886 6 3017137.
E-mail addresses: tsaicj@cis.nctu.edu.tw (C.-J. Tsai), leeci@mail.nutn.edu.tw (C.-I. Lee), wpyang@mail.ndhu.edu.tw (W.-P. Yang).

0957-4174/$ - see front matter © 2007 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2007.11.034

neural networks, genetic algorithms, decision trees, etc. Among them, decision trees are the most popular due to the merits of rapid construction, comparable accuracy, and readable rules (Rastogi & Shim, 1998).

According to how the training dataset is obtained, the classification problem also can be sorted into *non-incremental learning* and *incremental learning*. Incremental learning is important for applications in which the training dataset comes in the form of a data stream (Cunningham & Nowlan, 2003; Domingos & Hulten, 2000; Jin et al., 2003). In such an instance, it is not feasible to collect all training data before applying an algorithm. Incremental learning (Furnkranz & Widmer, 1994; Maloof, 2003; Maloof et al., 2002; Utgoff, 1989; Utgoff, Berkman, & Clouse, 1997) is becoming ever more important since most of information in our lives is presented in data stream consists of data block. However, most proposed approaches to incremental learning assumed data streams come under *stationary distribution*; namely, the data concept remains unchanged. But in reality, any instance of applicable data, such as disease variation, weather forecasts, consumers' shopping habits, or virus detection may vary as time goes by. Such a change of concept is known as *concept drift* (Cunningham & Nowlan, 2003; Hulten, Spencer, & Ddmingos, 2001; Klinkenberg, 2001; Klinkenberg et al., 1998; Kolter et al., 2003; Koychev, 2000; Lane & Brodley, 1998; Lee, Tsai, Wu, & Yang, 2007; Wang, Fan, Yu, & Han, 2003; Widmer & Kubat, 1996).

The current solutions to the problem of concept drift focus on efficiently rebuilding classifiers to accurately predict new incoming datasets. Unfortunately, they are unable to discern the main reasons why a concept drifts. 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. The idea is simple but novel; to our knowledge, we are the first group to address the problem of mining concept-drifting rules. Note that, what we address is different from the *emerging pattern*; which is the itemset whose support increases significantly in association rule mining (Fan & Ramamohanarao, 2006; Wang, Zhao, Dong, & Li, 2006). As claimed in Freitas (2000), 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.

In this paper, we propose a concept drift rule mining tree algorithm called CDR-Tree to accurately discover the rules of concept drift. The main contributions of this paper are: (a) mining concept-drifting rules, an interesting problem has never been discussed in the past is addressed; (b) our CDR-Tree can accurately mine the rules of concept drift; (c) if classification models are required, CDR-Tree can efficiently and accurately generate them via a simple extraction procedure instead of building them from scratch; (d) two strategies are also proposed to reduce the complexity of concept-drifting rules mined by CDR-Tree. The remainder of this paper is organized as follows: Section 2 is the review of related works. In Section 3, the theory of concept drift rules is introduced and then our concept drift rule mining tree algorithm is elucidated. The performance evaluation is shown in Section 4. In the last section, our conclusions and future research directions are explained.

2. Related work

In this section, we will introduce some related works, including some dealing with the traditional decision tree, incremental learning algorithms, and the proposed techniques for solving the concept drift problem.

2.1. Decision trees

A traditional decision tree (Clark & Niblett, 1989; Quinlan, 1986; Quinlan, 1993), as shown in Fig. 1, is a flow chart-like tree structure consisting of a number of Boolean functions. Within the decision tree, each *internal node* denotes a test on an attribute; and each branch represents an outcome of the test. Each *path* from the root to a leaf node forms a *rule*; and each *leaf node* is associated with a *target class* (or *class*). Before building a decision tree, a training dataset is required as shown in Table 1 (which is used for the building of the decision tree in Fig. 1). Each instance in the training dataset includes a set of attribute values and a target class.

Traditional decision tree approaches consist of two phases: the *building phase* and the *pruning phase*. In the building phase, according to a *splitting function* such as *information gain*, each internal node finds a splitting attribute to partition the coming data into corresponding child nodes unless all the examples in this node are pure (i.e. all

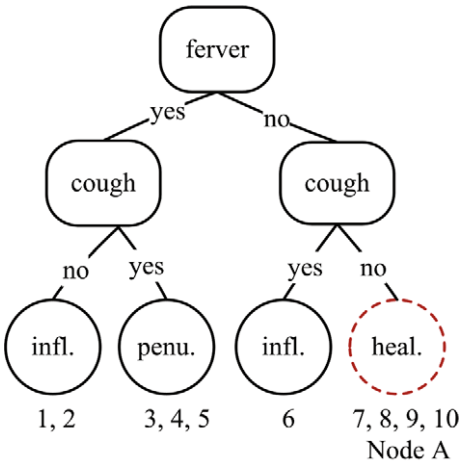

Fig. 1. The decision tree built using Table 1.

Table 1
The patients' diagnostic data

ID	Sex	Location	Fever	Cough	Diagnosis
1	Male	New York	Yes	No	Influenza
2	Female	Chicago	Yes	No	Influenza
3	Female	New York	Yes	Yes	Pneumonia
4	Male	New York	Yes	Yes	Pneumonia
5	Male	Chicago	Yes	Yes	Pneumonia
6	Male	New York	No	Yes	Influenza
7	Female	New York	No	No	Healthy
8	Male	New York	No	No	Healthy
9	Female	Chicago	No	No	Healthy
10	Female	Chicago	No	No	Healthy

examples in this node have the same target class) or cannot be further partitioned. Nevertheless, many of the branches reflect anomalies in the training data due to noise data or outliers. To prevent such an *overfitting* problem, a decision tree should prune its model to remove the least reliable branches. This generally results in faster classification and an improvement in the ability of the tree to correctly classify unseen data. To classify an unknown instance, beginning with the root node, successive internal nodes are visited until this example has reached a leaf node. The class of this leaf node is then assigned to the corresponding example as a prediction.

2.2. Incremental learning algorithms

A traditional decision tree algorithm such as ID3 (Utgoff, 1989) is not precise enough without taking incremental learning into consideration. Therefore, once the new instances are obtained, the decision tree needs to be re-built by using an integrated version of new and old datasets (Utgoff et al., 1997). Based on ID3, Schlimmer and Fisher proposed ID4 to solve this problem (Utgoff, 1989). The major difference with ID4 is that the new instances are added directly into the existing decision tree without generating a new one. ID4 keeps some information in every node and judges if the best splitting attribute is still the same after the addition of new instances. If they are the same, the new instances continue going to the next node and are evaluated again; but if they are not, the sub-tree root in this node is discarded and a new sub-tree is then built. However, the final decision tree is also influenced by the different arrangement sequence of instances even they are from the same new training dataset.

Regarding the solutions to ID4 problems mentioned above, Utgoff upgraded and developed a new decision tree evolution called ID5 (Utgoff, 1989). Similar to ID4, ID5 integrates new instances into decision trees. Also, ID5 preserves the useful information in each node and determines whether the best splitting attribute can still achieve optimal performance. If the best splitting attributes change, ID5 will replace them instead of deleting the sub-tree. The substitution made by ID5 is called pull-up. For pull-up to function properly, it is essential to have all instances stored in the decision tree. Therefore, the recording cost of ID5 is more expensive than that of ID4. Another disadvantage of ID5 is that it generates a larger decision tree and rules of greater complexity than either ID3 or ID4. In order to reduce the size of the decision tree generated by ID5, Utgoff has proposed ID5R (Utgoff, 1989). The primary purpose of ID5R is to ensure that the splitting attribute in each internal node is optimal and that the decision tree after revisions is identical to the one re-built by ID3 by applying the same training dataset.

2.3. Concept drift

Most incremental learning algorithms assume data stream under *stationary distribution*. But in reality, the concept of any given instance might either gradually or quickly vary. While the concept of data starts drifting, the classification model constructed by using old datasets becomes unsuitable for the new one. Thus, it is imperative that the old classification model be revised or a new one be re-built. At present, the solutions to the problem of concept drift can be generally divided into three categories:

2.3.1. Window-based approaches

Window-based approaches, like WAH (Widmer & Kubat, 1996) and DNW (Klinkenberg et al., 1998), pick up the training dataset within fixed or dynamic window sizes to construct a classification model (Hulten et al., 2001; Lazarescu & Venkatesh, 2004; Maloof, 2003). WAH properly adjusts window sizes based on the accuracy of classification. When the concept has drifted due to new input instances, WAH will make the window size about 20% smaller. Also, WAH will reduce window size to avoid retaining redundant and unnecessary instances while the concept remains stable. Once the concept has drifted, but while its variation still remains smaller than a given threshold, WAH will retain the original window size. The above mentioned conditions being unfulfilled indicates that more instances are required to construct the classifier. In the meanwhile, WAH will combine all new and old instances together into a whole new training dataset. Although WAH can resolve the problem of concept drift, it can only be appllied to small datasets. Therefore, Klinkenberg and Renz developed DNW to remedy this deficiency. DNW constructs the predicted model by the way of data blocks; its learning method is similar to WAH, but with different methods for adjusting the window sizes. DNW builds a classifier in each block and compare accuracy, recall, and precision between the current classifier and the previous one. After a full comparison, DNW will review the extent of variations and make proper adjustments of window sizes.

2.3.2. Weight-based approaches

According to certain factors such as how long the has been data stored, this kind of approach will assign each training instance with a distinctive weight (Kolter et al.,

2003; Koychev, 2000). Based on the weights, some outdated instances will be opportunistically discarded. Thus, its learning curve is quite similar to that of window-based approaches.

2.3.3. Ensemble classifiers

An ensemble classifier (Lee, Tsai, & Ku, 2006) deals with concept drift by utilizing multiple classifiers and by voting to construct a proper predictive model (Fan, 2004; Street & Kim, 2001; Wang et al., 2003). For each data block, there will be a classifier built. According to the accuracy and the period of construction of each classifier, each classifier will be assigned a weight. These weights not only influence the final voting results, but also are main factors for consideration as to whether classifiers are eliminated or not.

3. Concept drift rule mining tree algorithm

The currently proposed solutions for concept drift problems discussed in Section 2 are feasible to effectively and precisely predict the target class of new coming instances, but they are incapable of informing users which rules cause concept drift. However, as stated in the introduction, users may be much more interested in the rules governing concept drift.

3.1. The rules of concept drift

Take the patients' diagnostic data in Table 1 as the example again and assume that Table 2 is the new diagnostic dataset from the same patients. In Table 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. Fig. 2 is the decision tree constructed by using Table 2. Comparing Fig. 1 with Fig. 2, we find that patients ID_9 and ID_{10} 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 (work = "Shanghai") and (cough = "yes") then (diagnosis = "SARS").

Table 2
The new coming diagnostic data from the same patients

ID	Sex	Work	Fever	Cough	Diagnosis
1	Male	**Chicago**	**No**	No	**Healthy**
2	Female	Chicago	**No**	No	**Healthy**
3	Female	**Shanghai**	Yes	**No**	**Influenza**
4	Male	New York	**No**	**No**	**Healthy**
5	Male	**New York**	Yes	**No**	Pneumonia
6	Male	New York	No	Yes	Influenza
7	Female	New York	**Yes**	No	**Pneumonia**
8	Male	New York	**Yes**	**Yes**	**Pneumonia**
9	Female	**Shanghai**	**Yes**	**Yes**	**SARS**
10	Female	**Shanghai**	**Yes**	**Yes**	**SARS**

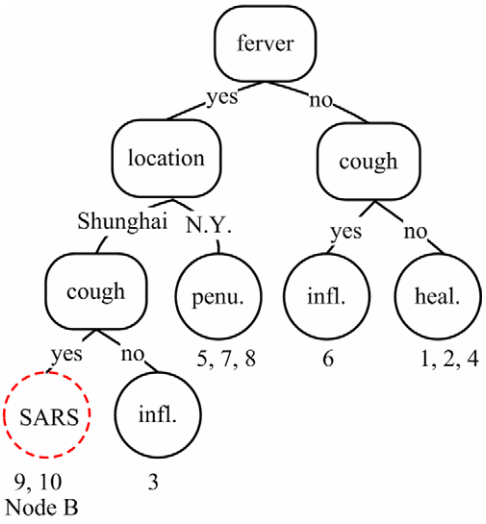

Fig. 2. The decision tree built using Table 2.

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 Shanghai, and had a bad cough. Simply stated, the variations of those three attributes, fever, working location, 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 (location = "New York → Shanghai") 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 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.

3.2. Concept drift rule mining tree

In order to mine the concept drift rules mentioned in Section 3.1, here we propose our CDR-Tree algorithm. Section 3.2.1 is the building step of CDR-Tree. The idea is simple but novel. 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, in addition to the concept drift rules, users might also require the classification model of each data block. CDR-Tree algorithm can do that via a quick and simple extraction step as will described in Section 3.2.2. Finally, two strategies are proposed in Section 3.2.3 to reduce the complexity of concept drift rules mined by CDR-Tree algorithm.

3.2.1. Building a CDR-Tree

To mine concept drift rules, CDR-Tree algorithm initially integrates new and old instances from different times

Table 3
The integrated data of Tables 1 and 2

ID	Location	Fever	Cough	Diagnosis
1	New York → **Chicago**	Yes → **No**	No → No	Influenza → **Healthy**
2	Chicago → Chicago	Yes → **No**	No → No	Influenza → **Healthy**
3	New York → **Shanghai**	Yes → Yes	Yes → **No**	Pneumonia → **Influenza**
4	New York → New York	Yes → **No**	Yes → **No**	Pneumonia → **Healthy**
5	Chicago → **New York**	Yes → Yes	Yes → **No**	Pneumonia → Pneumonia
6	New York → New York	No → No	Yes → Yes	Influenza → Influenza
7	New York → New York	No → **Yes**	No → No	Healthy → **Pneumonia**
8	New York → New York	No → **Yes**	No → **Yes**	Healthy → **Pneumonia**
9	Chicago → **Shanghai**	No → **Yes**	No → **Yes**	Healthy → **SARS**
10	Chicago → **Shanghai**	No → **Yes**	No → **Yes**	Healthy → **SARS**

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 pairs 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. Taking Tables 1 and 2 as our example again, the integrated data of the two tables are shown in Table 3, and Fig. 3 is the corresponding CDR-Tree. As described in Section 3.1, for the patients ID_9 and ID_{10}, there is a drifting rule:

If (fever = "no → yes") and (Work = "Chicago → Shanghai") and (cough = "no → yes") then (diagnosis = "healthy → SARS").

However, a traditional decision tree will stop splitting at the node C in Fig. 4 and then produce a rule:

If (fever = "no → yes") and (Work = "Chicago → Shanghai") then (diagnosis = "healthy → SARS").

It is clear that the former rule is more reliable and accurate than the latter one. To solve this problem, CDR-Tree algorithm goes on splitting a pure node n_o in which all instances have some common attribute value a_i, but this attribute a 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 Fig. 3. There are five concept drift rules as follows:

Rule a: If (fever = "no → yes") and (work = "Chicago → Shanghai") and (cough = "no → Yes") then (diagnosis = "healthy → SARS").
Rule b: If (fever = "no → yes") and (work = "New York → New York") then (diagnosis = "healthy → pneumonia").
Rule c: If (fever = "yes → no") and (cough = "yes → no") then (diagnosis = "pneumonia → healthy").

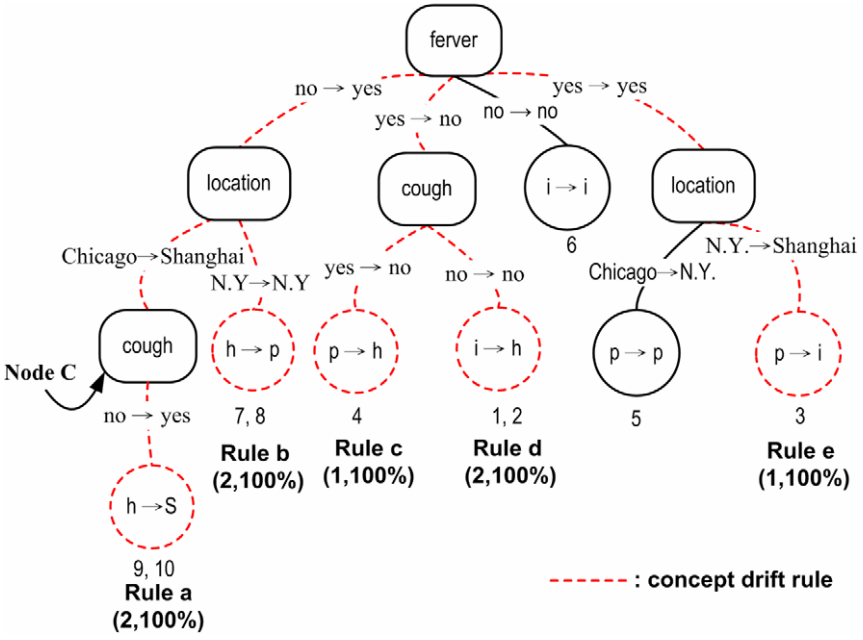

Fig. 3. The CDR-Tree built using Table 3.

Definition
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, is set to 50% as the default
Input: T^p, T^q, RS, RC
Output: *CDR-Tree*
Procedure CDRTree(T^p, T^q, df, RS, RC)
/*initial */
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 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_i 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;
      Assign the target class label by majority vote;
      $RS_o = N_o$;
      $RC_o = (100 N_c / N_o)\%$
      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*

Fig. 4. The pseudo code of CDR-Tree algorithm.

Rule d: If (fever = "yes → no") and (cough = "no → no") then (diagnosis = "influenza → healthy").
Rule e: If (fever = "yes → yes") and (work = "New York → Shanghai") 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 drift rules of the patients ID_9 and ID_{10} mentioned in Section 3.1 are definitely mined in this CDR-Tree.

In order to provide users with meaningful and interesting rules of concept drift, CDR-Tree algorithm defines a rule support RS and a rule confidence RC to filter unmeaningful 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:

$$\mathrm{RS} = N_o \text{ and}$$
$$\mathrm{RC} = (100 N_c / N_o)\%,$$

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 Fig. 3 and the pseudo code of CDR-Tree algorithm is presented in Fig. 4.

3.2.2. Extracting the decision tree from CDR-Trees

When users require the old or new classification model, 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 target class 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-Tee and node n_o will be pulled-up. This situation is illustrated in Fig. 5a.

(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 Fig. 5b.

(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 target class of n_m is assigned by a majority vote. This situation is illustrated in Fig. 5c 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 Fig. 5e illustrates. Note that a migrant instance may stop in an internal node n_I of ST_m if there is no branch to proceed. In such a condition, the CDR-Tree will use the splitting attribute in n_I to generate a new branch and accordingly a new leaf node as illustrated in Fig. 5f. 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.

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

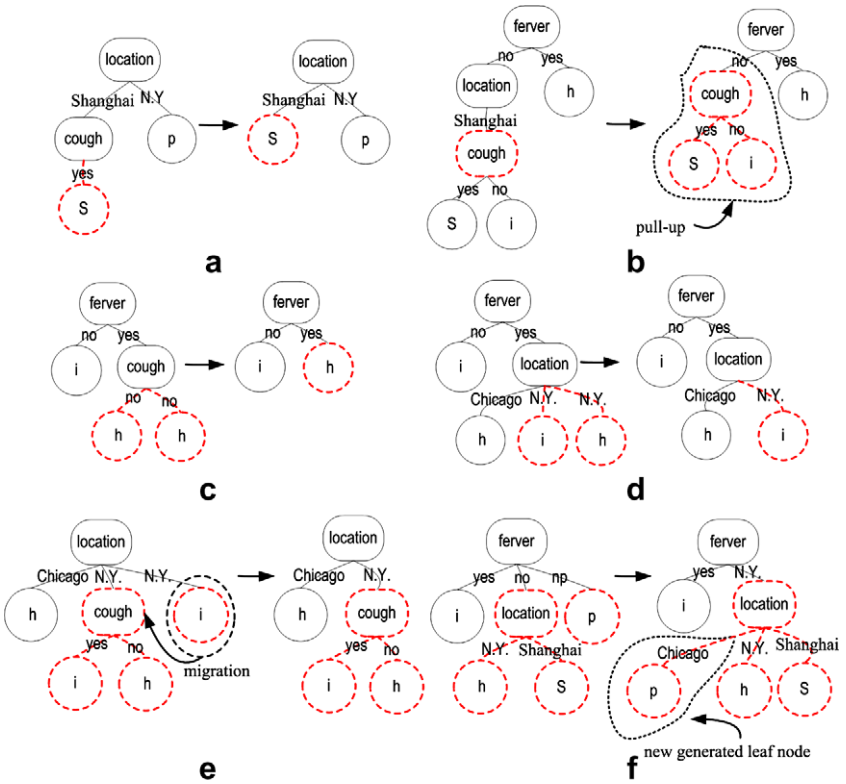

Fig. 5. Illustrations of the extraction strategy in CDR-Tree algorithm.

the decision tree much more efficiently and quickly. Fig. 6 is the pseudo code of the CDR-Tree's extraction procedure. Taking the CDR-Tree in Fig. 3 as an example, the extract decision trees are shown in Fig. 7, where Fig. 7a is the old classification model for Table 1 without implementing Step 5; Fig. 7b is still the model for Table 1 but with the implementation of Step 5; and Fig. 7c and d correspond to Table 2. By comparing these results to those in Figs. 1 and 2, we can find that without the implementation of Step 5, there are only 1 misclassified instance in Fig. 7a and 2 in Fig. 7c. When Step 5 is executed, Fig. 7b and d reach 100% accuracy as are Figs. 1 and 2. Furthermore, Fig. 7d is identical to Fig. 2, but Fig. 7b is a little different from Fig. 1. From the 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.

3.2.3. Reducing the complexity of concept-drifting rules

Even though the main purpose of CDR-Trees is to clearly discover concept drift rules and to utilize rule support (RS) and rule confidence (RC) to filter un-meaningful ones, readers might wonder whether our approach has a

Input: *CDR-Tree*
Output: *decision tree*
Procedure CDRTreeExtract (*CDR-Tree*)
If the decision tree of old instances is requested then
    The splitting attribute values in all internal nodes and the target class in all leaf nodes of the new instances are ignored;
Else
    The splitting attribute values in all internal nodes and the target class 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-Tee;
    Pull-up node n_o;
  If node n_o is an internal and singleton node then
    Remove its parent node from the CDR-Tee;
    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 themintoa singlenode n_m;
    Assigna targetclass to n_m by the majority vote;
  If n_s has the same splitting value to that of n_o but not all of themare 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 belongtothese removed leafnodes and sub-treeintothe internal node n_m;
    For each migrant instance in ST_m
      If it can reach a leafnode
        Migrate it intothe leafnode;
      Else
        Migrate it into the internal node where no branch can be proceeded;
      End if
    For each node in the path of of ST_m
      If it isa leafnode 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
        Createnew branchand corresponding leafnodes byusing the splitting attribute in n_I;
        Assign a targetclass 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

Fig. 6. The pseudo code of the extraction procedure in CDR-Tree algorithm.

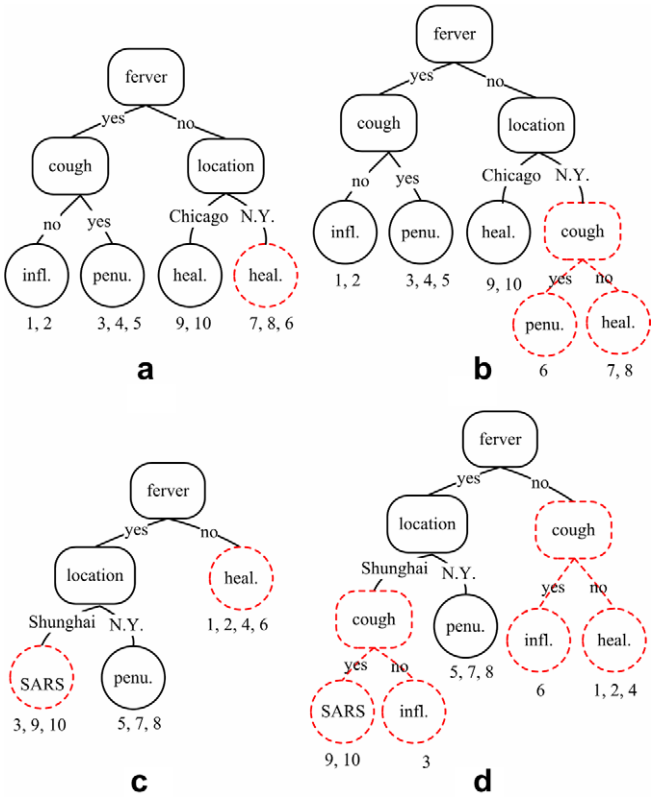

Fig. 7. The extracted decision trees from Fig. 3: (a) the model of Table 1 without implementing Step 5; (b) the model of Table 1 with the implementation of Step 5; (c) the model of Table 2 without implementing Step 5; and (d) the model of Table 2 with the implementation of Step 5.

higher computational cost due to the fact that CDR-Trees are more complex than a traditional decision trees. Nevertheless, it is not as bad as imagined. Here we discuss the corresponding computational cost under the conditions of concept stability and concept drift. First, suppose that there are no drifting concepts: the number of attributes, the number of instances, and the number of values for an attribute in the integrated dataset would be all the same as that in the old or new dataset. Therefore, the computational cost is very similar. The extra cost of CDR-Trees is the integration of new and old instances from different time points and the extraction procedure. However, such computational costs are trivial when compared to those of building a decision tree. When there are drifting instances, the number of attributes and the number of instances remain the same, but the number of values for an attribute might increase. The increase of attribute values burdens the building of a node. Suppose a dataset contains i attributes and each attribute has j kinds of values, the computational cost for building a node in a CDR-Tree would be ij times that of a traditional decision tree. The worst case occurs only when the drifting ratio (i.e. the proportion of drifting instances to all instances) is 100% and all values of an attribute in the old dataset change to different values in the new dataset. In the worst case, each attribute in the integrated dataset would contain j^2 kinds of values. However, the worst case should rarely happen since in most real datasets: (a) the drifting ratio should be not too large; (b) drifting instances should gather in some specific areas in the dimensional space of attributes, i.e., there must be some meaningful attribute-values to cause concept drift. For example, age and salary may influence credit card applications but weight and height will not; IP address and the number of sending packages may be the main basis for finding a PC which sends virus packages.

Of course, many good *discretization algorithms* (Kurgan & Cios, 2004; Lee, Tsai, Yang, & Yang, 2007; Liu, Hussain, Tan, & Dash, 2002) can be used to preprocess the integrated dataset to speed-up the construction of the CDR-Tree. In this paper, we also propose two strategies: T-strategy and V-strategy. The main goal of T-strategy is to reduce the number of concept-drifting rules and it is adopted after a CDR-Tree is built; on the contrary, V-strategy is used to simplify the training dataset to speed-up the building of CDR-Tree.

3.2.3.1. T-strategy. If the taxonomy tree of an attribute is given, we can remove some branches from the CDR-Tree and merge some produced drifting rules. For example, suppose the taxonomy tree of attribute "location" is shown in Fig. 8a: the CDR-Tree in Fig. 8b can be simplified as Fig. 8c. Similarly, the two discovered concept drift rules:

> If (fever = "no → yes") and (work = "Chicago → Shanghai") then (diagnosis = "healthy → influenza"),
> If (fever = "no → yes") and (work = "New York → Shanghai") then (diagnosis = "healthy → influenza")

can be merged into only one rule:

> If (fever = "no → yes") and (work = "USA → Shanghai") then (diagnosis = "healthy → influenza").

3.2.3.2. V-strategy. If two people of different weights demonstrate an abnormal increase of K kilograms synchronously, their health is at risk. That is to say, for some attributes, we can regard the variance K as the cause of concept drift even if the original values are different. We define such an attribute as *variance attribute* in the following:

Definition 1. For a continuous attribute M and two data blocks T_p and T_q, assume that the attribute values of instance i and j are respectively M_i^p, M_j^p in T_p, and M_i^p,M_j^p, respectively, varies into M_i^q and M_j^q in T_q, where $M_i^p \neq M_i^q$ and $M_j^q \neq M_j^p$. If $M_i^p - M_i^q = M_j^p - M_j^q = v(v \geqslant 1)$, and the variance v of attribute M would make the concept of both instance i, j drift form c to c', attribute M is called a *variance attribute*.

A scheme governing the variable attribute "weight" is illustrated in Fig. 9a. It means an increase of 0 or 5 kg in weight has the same influence on concept drift. Similarly,

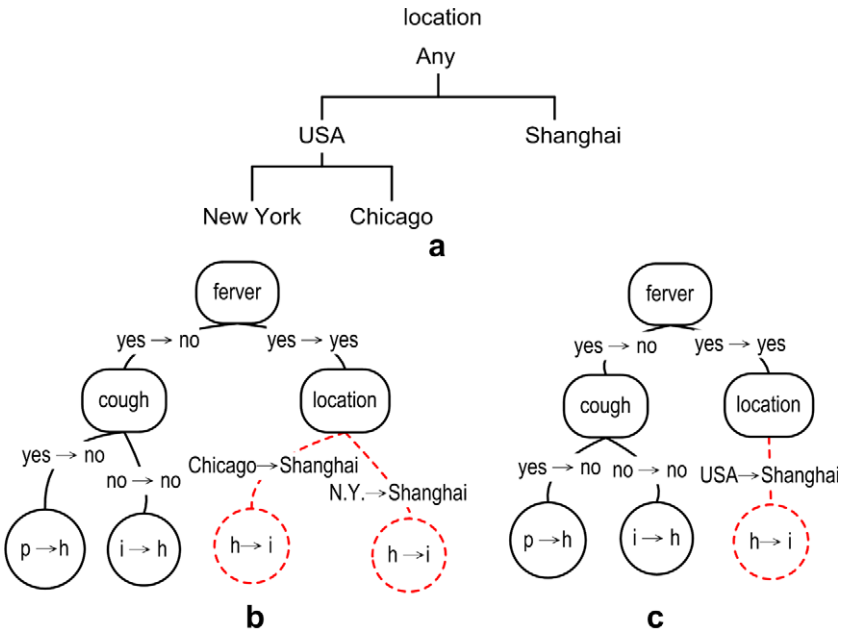

Fig. 8. Illustrations of T-strategy: (a) The taxonomy tree of attribute "location"; (b) a CDR-tree; and (c) the simplified CDR-tree of (b) by using T-strategy.

the increase of 12 or 25 kg would cause the same degree of concept drift. For the real datasets, there may be some variable attributes such as weight and blood pressure. With the given scheme, the CDR-Tree can reduce the number of attribute values after data integration and prevent the decision tree from being too immense and complex. Fig. 9b and c are the illustrations of a CDR-tree, where Fig. 9c is the simplified version of Fig. 9b created by applying the V-strategy. Note that, the V-Strategy is different from the proposed discretization algorithms.

4. Experimental analysis and performance evaluation

In this section, 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 b y different drift levels. The effectiveness of the V-strategy is evaluated in Section 4.3. We compare the accuracy of C4.5 to that of the model extracted from the CDR-Tree in Section 4.4. Finally, the comparison of execution time among CDR-Tree, the model extracted from the CDR-Tree, and C5.0 is given in Section 4.5.

4.1. Experimental environment and datasets

All experiments in this paper are done on a 3.0 GHz Pentium IV machine with 512 MB DDR memory, running Windows 2000 professional. Experimental datasets are generated by IBM Data Generator (Agrawal, Ghosh, Imielinski, & Swami, 1992). We use IBM Data Generator

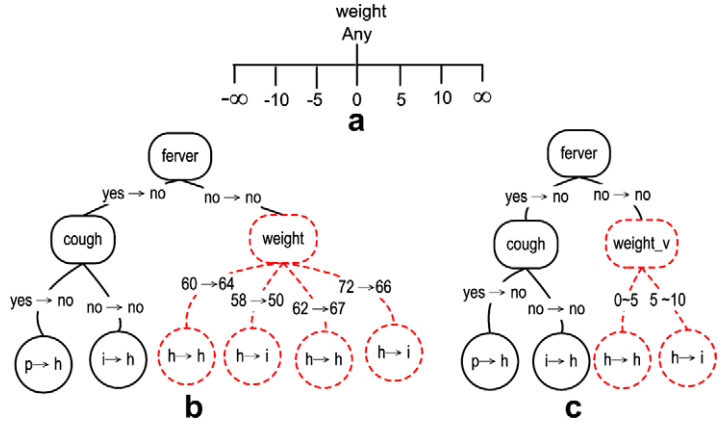

Fig. 9. Illustrations of the V-strategy: (a) the scheme of the variable attribute "weight"; (b) a CDR-Tree; and (c) the simplified CDR-Tree of (b) by using the V-strategy.

instead of the hyperplane synthetic data used in (Kolter et al., 2003) since IBM Data Generator is a public and widely used data generator. In addition, IBM Data Generator has several well-defined classification functions and parameters which can be used to generate different characteristics of datasets. The dataset generated by IBM Data Generator contains one Boolean target class and nine basic attributes: salary, commission, loan, age, zip code, h-years, h-value, e-level, and car. Among the nine attributes, zip code, e-level, and car are categorical attributes, and others are the continuous attributes. 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, P3, P5, P43, and P45 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 $(0 < m < 6)$ 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 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 will generate 5 s datasets with different drifting ratios. A total of 4 old datasets and 20 new datasets are generated in our experiments. Every dataset includes 10,000 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 the classification function Pi and $D(i, R)$ to represent a dataset with $R\%$ drifting ratio resulting from $D(i)$.

4.2. The analysis of CDR-Tree

In this section, we use the 24 datasets mentioned in Section 4.1 to evaluate the accuracy of CDR-Trees and to analyze whether it can precisely explore the concept drift rules.

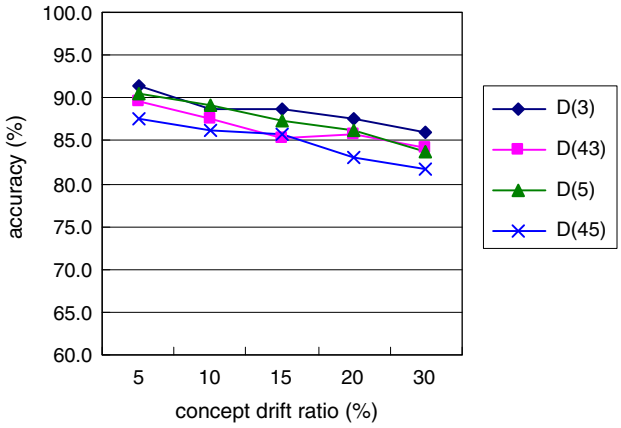

Fig. 10. The accuracy of CDR-Trees under five different drifting ratios.

At first, focusing on five different drift levels, the accuracy of the CDR-Tree in 20 integrated datasets is shown in Fig. 10. 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 Fig. 11. As expected, the concept drift rules mined by CDR-Tree can accurately predict those drifting instances.

4.3. The comparison between CDR-Trees and V-CDR-Trees

Due to the lack of background knowledge, we are not able to produce a proper taxonomy tree for our experimental dataset to evaluate our T-strategy. However, to analyze whether the V-Strategy can effectively reduce the complexity of CDR-Tree, 4 old datasets and 24 new ones mentioned in Section 4.1 are again utilized. We use V-CDR to represent a CDR-Tree with V-Strategy in this experiment. A simple variance scheme $|v| = 1$ for all continuous

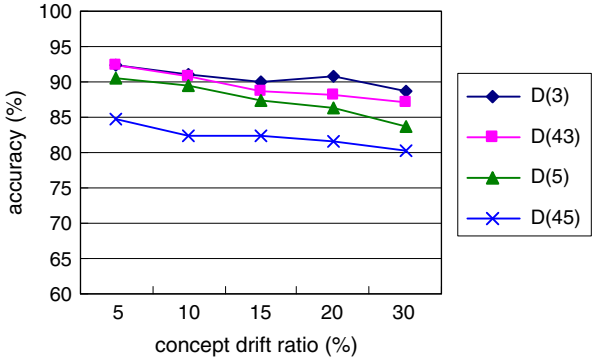

Fig. 11. The accuracy of concept-drifting rules produced by CDR-Trees.

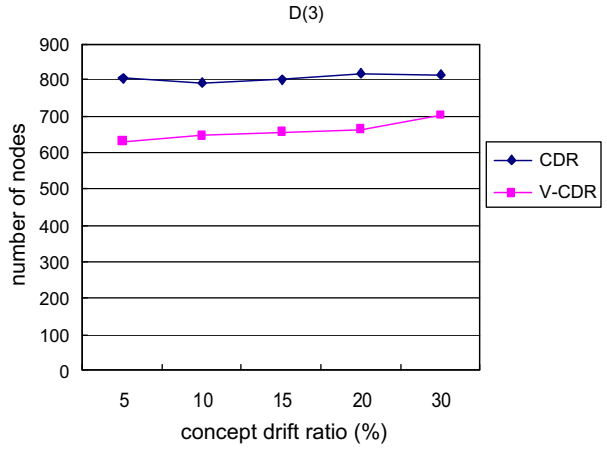

Fig. 12. The comparison of number of nodes using dataset $D(3)$.

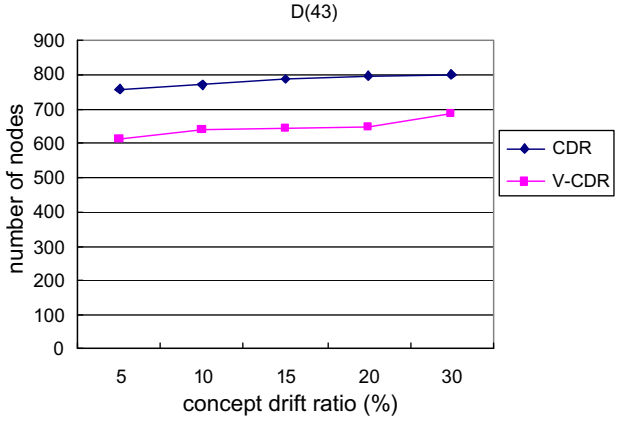

Fig. 13. The comparison of number of nodes using dataset $D(43)$.

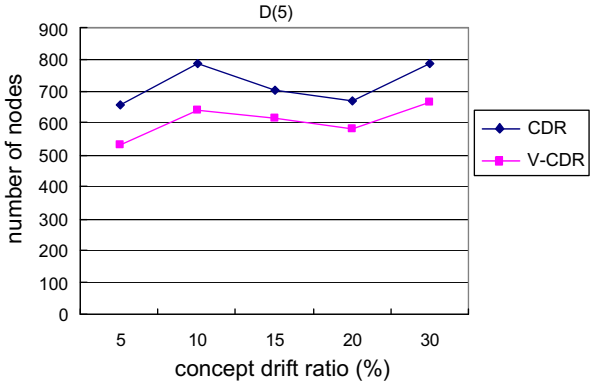

Fig. 14. The comparison of number of nodes using dataset $D(5)$.

attributes is used. The experimental results are shown from Figs. 12–15. As can be seen from the four figures, the number of nodes in V-CDR-Trees is significantly smaller than that of CDR-Trees in all drift ratios. In conclusion, it is shown that V-Strategy can effectively lower the complexity of the CDR-Tree.

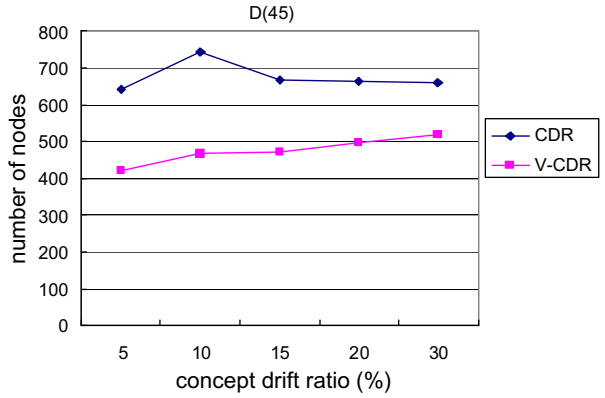

Fig. 15. The comparison of number of nodes using dataset $D(45)$.

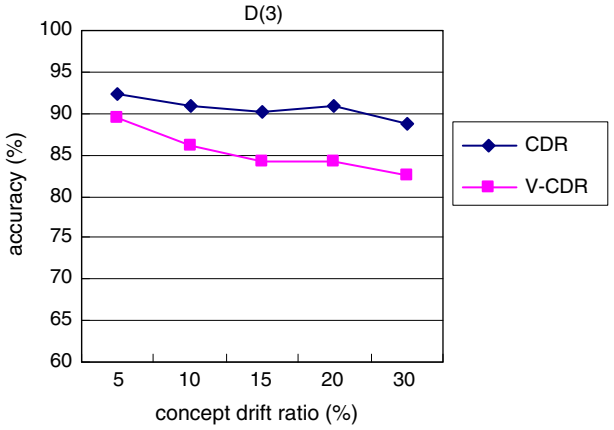

Fig. 16. The comparison of accuracy of concept drift rules using dataset $D(3)$.

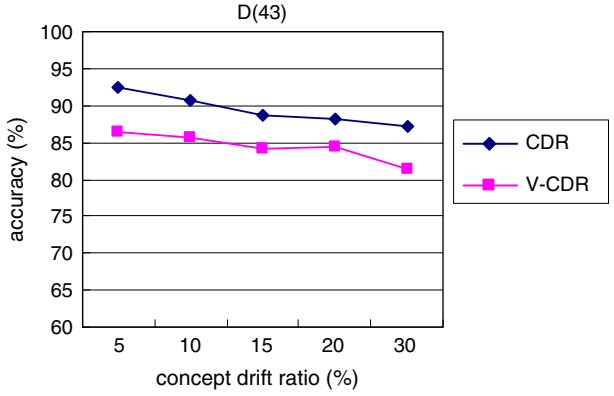

Fig. 17. The comparison of accuracy of concept drift rules using dataset $D(43)$.

In order to compare the accuracy of the mined concept drift rules between CDR-Trees and V-CDR-Trees, among each experimental dataset, the testing instances that really have a drifting concept are used to calculate accuracy. The experimental results are shown from Figs. 16–19 and we can see that the accuracy of the V-CDR-Tree is slightly lower than that of the CDR-Tree in these cases even though we use a very simple variance scheme.

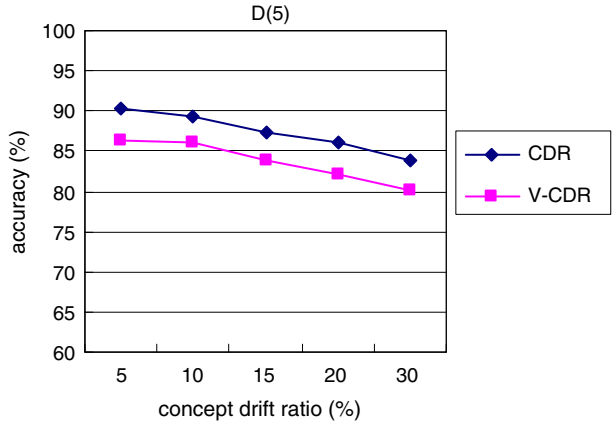

Fig. 18. The comparison of accuracy of concept drift rules using dataset $D(5)$.

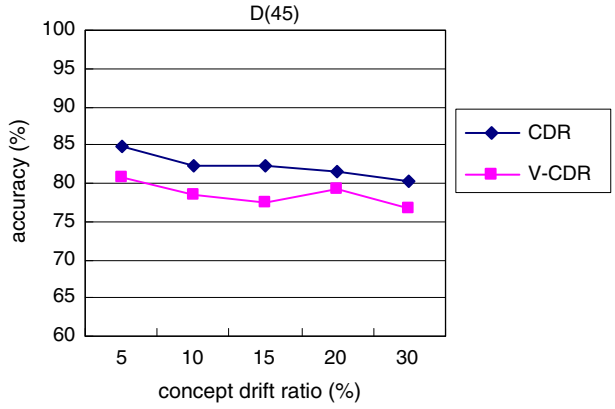

Fig. 19. The comparison of accuracy of concept drift rules using dataset $D(45)$.

4.4. The comparison of accuracy between E-CDR-Trees and C5.0

In this experiment, we evaluate whether our approach mentioned in Section 3.2.2 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 Fig. 20. From Fig. 20 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 3.2.2.

4.5. The comparison of execution time among CDR-Trees, E-CDR-Trees, and C5.0

The motivation behind CDR-Trees and C5.0 is inherently different: CDR-Tree algorithm mainly aims at providing concept-drifting rules and quickly extracts the prediction 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 Fig. 21. 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 P43. 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 3.2.2 can efficiently elucidate the classification model than building it from scratch.

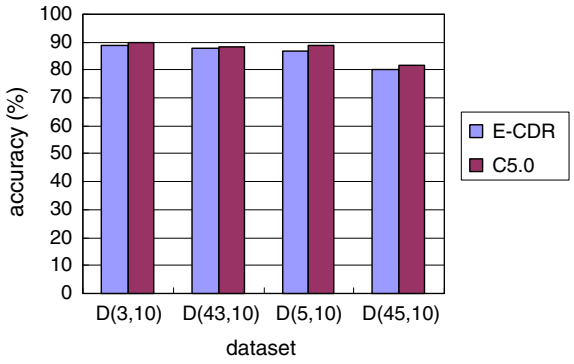

Fig. 20. The comparison of accuracy between E-CDR-Tree and C5.0 using four datasets with 10% drifting ratio.

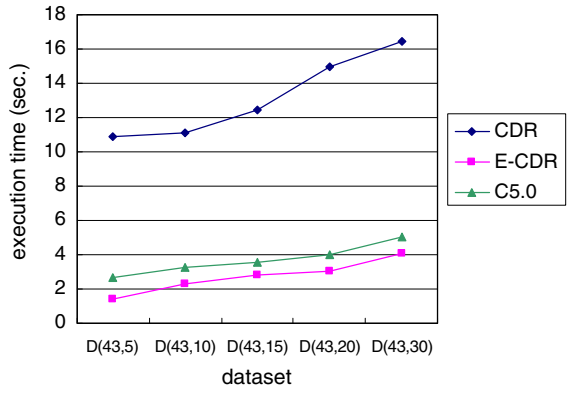

Fig. 21. The comparison of execution time among CDR-Tree, E-CDR-Tree, and C5.0 by using datasets $D(43)$.

5. Conclusions and future research direction

Recently, concept drift has become a popular research issue in the field of data mining. Even though many schol-

ars have proposed different methods, they all focus only on updating the classification model and are unable to elucidate the main causes why concept drifts. However, the decision makers might be more interested in the rules of concept drift. In this paper, we address this issue and propose the Concept Drift Rule Mining Tree algorithm to solve this problem. Our CDR-Tree cannot only produce drifting rules, but also efficiently extract the classification model of each data block for decision makers to have wide application. T-strategy and V-strategy, which need the support of users in the corresponding domain, are also proposed to simplify the CDR-Tree and the mined rules. The experimental results in Section 4 show the accuracy of the CDR-Tree and the efficiency of our extracting strategies.

There are still many issues worth further investigation. First, in this paper we only analyze the cases in which there are only two data blocks in a data stream. If analysis of greater than two is required, the CDR-Tree will become much larger and more complicated. Therefore, one of our future focuses is to extend the use of the CDR-Tree which can more efficiently process multi-block concept drift problems. Secondly, although we propose two strategies to reduce the complexity of our CDR-Tree algorithm, for a given dataset, there might be no given taxonomic tree or variance attributes. Therefore, another interesting problem is to propose a method to generate a taxonomic tree or to find variance attributes automatically. Thirdly, an existing discretization algorithm can also be used to reduce the complexity of CDR-Trees. However, since the integrated dataset used in CDR-Trees is different than a traditional dataset, another future focus is to propose a discretization algorithm which is more suitable to CDR-Trees. Finally, although our extraction methods can efficiently extract the classification model for each data block 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 in which one can extract an identical tree.

References

Agrawal, R., Ghosh, A., Imielinski, T., Iyer B., & Swami, A. (1992). An interval classifier for database mining applications. In *Proceedings of the 18th conference on very large databases*.

Clark, P., & Niblett, T. (1989). The CN2 induction algorithm. *Machine Learning, 3*(4), 261–283.

Cunningham, P., & Nowlan, N. (2003). A case-based approach to spam filtering that can track concept drift. In *Proceedings of the ICCBR workshop on long-lived CBR systems*.

Domingos, P., & Hulten, G. (2000). Mining high-speed data streams. In *Proceedings of the sixth international conference on knowledge discovery and data mining* (pp. 71–80). Boston.

Fan, H., & Ramamohanarao, K. (2006). Fast discovery and the generalization of strong jumping emerging patterns for building compact and accurate classifiers. *IEEE Transactions on Knowledge and Data Engineering, 18*(6), 721–737.

Fan, W. (2004). 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).

Freitas, A. A. (2000). Understanding the crucial differences between classification and discovery of association rules. *SIGKDD Explorations, 2*(1), 65–69.

Furnkranz, J., & Widmer, G. (1994). Incremental reduced error pruning. In *Proceedings of the 11th international conference on machine learning* (pp. 70–77). San Francisco.

Han, J., & Kamber, M. (2001). *Data mining: Concepts and techniques*. Morgan Kaufmann Publisher.

Hulten, G., Spencer, L., & Ddmingos, P. (2001). Mining time-changing data streams. In *Proceedings of the seventh ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 97–106). San Francisco.

Jin, R., & Agrawa, G. (2003). Efficient decision tree construction on streaming data. In *Proceedings of the nineth ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 571–576). Washington.

Klinkenberg, R. (2001). Using labeled and unlabeled data to learn drifting concepts. Workshop notes of the IJCAI-01 workshop on learning from temporal and spatial data (pp. 16–24). CA.

Klinkenberg, R., & Renz, I. (1998). Adaptive information filtering: Learning in the presence of concept drifts. Workshop notes of the ICML-98 workshop on learning for text categorization (pp. 33–40). CA.

Kolter, J. Z., & Maloof, M. A. (2003). Dynamic weighted majority: A new ensemble method for tracking concept drift. In *Proceedings of the third international IEEE conference on data mining* (pp. 123–130). Melbourne, FL.

Koychev, I. (2000). Gradual forgetting for adaptation to concept drift. In *Proceedings of ECAI 2000 workshop current issues on spatio-temporal reasoning*. Germany.

Kurgan, L., & Cios, K. J. (2004). CAIM discretization algorithm. *IEEE Transactions on Knowledge and Data Engineering, 16*(2), 145–153.

Lane, T., & Brodley, C. E. (1998). Approaches to online learning and concept drift for user identification in computer security. In *Proceedings of the fourth international conference on knowledge discovery and data mining* (pp. 259–263). New York.

Lazarescu, M., & Venkatesh, S. (2004). Using multiple windows to track concept drift. *Intelligent Data Analysis Journal, 8*(1), 29–59.

Lee, C. I., Tsai, C. J., Wu, T. Q., & Yang, W. P. (2008). A multi-relational classifier for imbalanced database. *Expert Systems with Applications, 36*(3), 2008.

Lee, C. I., Tsai, C. J., Yang, Y. R., & Yang, W. P. (2007). A top-down and greedy method for discretization of continuous attributes. In *Proceedings of the fourth international conference on fuzzy systems and knowledge discovery*. Haikou, China.

Lee, C. I., Tsai, C. J., Wu, J. H., & Yang, W. P. (2007). A decision tree-based approach to mining the rules of concept drift. In *Proceedings of the fourth international conference on fuzzy systems and knowledge discovery*. Haikou, China.

Lee, C. I., Tsai, C. J., & Ku, C. W. (2006). An evolutionary and attribute-oriented ensemble classifier. In *Proceedings of the international conference on computational science and its applications* (pp. 1210–1218).

Liu, H., Hussain, F., Tan, C. L., & Dash, M. (2002). Discretization: An enabling technique. *Journal of Data Mining and Knowledge Discovery, 6*(4), 393–423.

Maloof, M. (2003). Incremental rule learning with partial instance memory for changing concepts. In *Proceedings of the international joint conference on neural networks*. CA.

Maloof, M.A., and Michalski, R.S. (2002). Incremental learning with partial instance memory. In *Proceedings of the 13th international symposium on methodologies for intelligent systems*. Lyon, France.

Menzies, T. (2003). Data mining for very busy people. In *Proceedings of the international IEEE conference on data mining* (pp. 22–29).

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning, 1*(1), 81–106.

Quinlan, J. R. (1993). *C4.5: Program for machine learning*. San Mateo, CA: Morgen Kaufmann Publisher.

Rastogi, R., & Shim, K. (1998). PUBLIC: a decision tree classifier that integrates building and pruning. In *Proceedings of the 24th international conference on very large databases* (pp. 404–415).

Street, W., & Kim, Y. (2001). A streaming ensemble algorithm for large-scale classification. In *Proceedings of the seventh international conference on knowledge discovery and data mining* (pp. 377–382). NY.

Tsai, C. J., Lee, C. I., Chen, C. T., & Yang, W. P. (2007). A multivariate decision tree algorithm to mine imbalanced data. *WSEAS Transactions on Information Science and Applications, 4*(1), 50–58.

Utgoff, P. E. (1989). Incremental induction of decision trees. *Machine Learning, 4*(2), 161–186.

Utgoff, P., Berkman, N., & Clouse, J. (1997). Decision tree induction based on efficient tree restructuring. *Machine Learning, 29*(1), 5–44.

Wang, H., Fan, W., Yu, P. S., & Han, J. (2003). Mining concept-drifting data streams using ensemble classifiers. In *Proceedings of the nineth ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 226–235). Washington, DC.

Wang, L., Zhao, H., Dong, G., & Li, J. (2006). On the complexity of finding emerging patterns. *Theoretical Computer Science, 335*(1), 15–27.

Widmer, G., & Kubat, M. (1996). Learning in the presence of concept drift and hidden contexts. *Machine Learning, 23*(1), 69–101.