Pattern Recognition Letters 30 (2009) 1347–1355

Contents lists available at ScienceDirect

Pattern Recognition Letters

journal homepage: www.elsevier.com/locate/patrec

Ambiguous decision trees for mining concept-drifting data streams

Jing Liu [a], Xue Li [b,*], Weicai Zhong [a]

[a] Institute of Intelligent Information Processing, Xidian University, Xi'an 710071, China
[b] School of Information Technology and Electrical Engineering, The University of Queensland, Brisbane, Qld 4072, Australia

ARTICLE INFO

Article history:
Received 11 July 2008
Received in revised form 16 July 2009
Available online 3 August 2009

Communicated by W. Pedrycz

Keywords:
Data streams
Data mining
Concept drift
Ambiguous decision trees
Incremental learning

ABSTRACT

In real world situations, explanations for the same observations may be different depending on perceptions or contexts. They may change with time especially when concept drift occurs. This phenomenon incurs ambiguities. It is useful if an algorithm can learn to reflect ambiguities and select the best decision according to context or situation. Based on this viewpoint, we study the problem of deriving ambiguous decision trees from data streams to cope with concept drift. CVFDT (Concept-adapting Very Fast Decision Tree) is one of the most well-known streaming data mining methods that can learn decision trees incrementally. In this paper, we establish a method called ambiguous CVFDT (aCVFDT), which integrates ambiguities into CVFDT by exploring multiple options at each node whenever a node is to be split. When aCVFDT is used to make class predictions, it is guaranteed that the best and newest knowledge is used. When old concepts recur, aCVFDT can immediately relearn them by using the corresponding options recorded at each node. Furthermore, CVFDT does not automatically detect occurrences of concept drift and only scans trees periodically, whereas an automatic concept drift detecting mechanism is used in aCVFDT. In our experiments, hyperplane problem and two benchmark problems from the UCI KDD Archive, namely Network Intrusion and Forest CoverType, are used to validate the performance of aCVFDT. The experimental results show that aCVFDT obtains significantly improved results over traditional CVFDT.

1. Introduction

Applications involving streaming data often grow continuously over time. Examples of such applications include sensor network monitoring, financial data analysis, telephone records, and large sets of web pages requested by HTTP calls. Research into how to mine streaming data is attracting ever-increasing attention. Practical data streams would constantly exist over months or years. The underlying processes may change during this period, sometimes radically, and consequently concept drift (Schlimmer and RichardII, 1986) always occurs in data streams. Many algorithms have been proposed for handling this phenomenon, such as various algorithms based on ensemble methods (Kolter and Maloof, 2003; Street and Kim, 2001; Wang et al., 2003), support vector machines (Klinkenberg, 2004), and decision trees (Fan et al., 2004; Gama et al., 2005; Gama et al., 2006; Hulten et al., 2001; Natwichai and Li, 2004).

Among the methods of deriving decision trees, CVFDT (Concept-adapting Very Fast Decision Tree) (Hulten et al., 2001) has attracted much attention. By using the Hoeffding bound (Hoeffding, 1963; Maron and Moore, 1994), CVFDT can build decision trees incrementally from data streams. To cope with concept drift, CVFDT keeps a decision tree up-to-date with a window of examples and periodically scans the current decision tree to find out whether concept drift is occurring. When concept drift occurs in certain nodes, an alternate subtree is started at each of those nodes. When the alternate subtrees become more accurate for treating new and incoming data than the old ones, they replace the old ones.

In CVFDT, alternate subtrees cannot be used to predict the class labels of incoming examples. They are only a candidate set for current tree branches, and they can become part of the current tree only when their prediction accuracies are higher than those of current tree branches. However, during this period, it may be better to use alternate subtrees instead of current tree branches to make class predictions. Moreover, if old concepts recur after alternate subtrees have replaced the old ones, CVFDT must learn them again from scratch. In this paper, we integrate ambiguities into the CVFDT algorithm, leading to a new approach that we term ambiguous CVFDT (aCVFDT). This method involves exploring multiple options at each node and effectively integrating alternate subtrees into the current tree. Thus, whenever aCVFDT is used to make class predictions, it is guaranteed that the newest and the best performing knowledge can be used. Whenever old concepts recur, aCVFDT can immediately relearn them from the corresponding options recorded at each node.

* Corresponding author. Fax: +86 29 8820 1023.
E-mail addresses: neouma@163.com (J. Liu), xueli@itee.uq.edu.au (X. Li).

Another weakness of CVFDT is that it does not automatically detect occurrences of concept drift and only scans the trees periodically. That is to say, even without concept drift, CVFDT still needs to scan the whole tree periodically. This unnecessarily increases the computational cost. To improve the performance of CVFDT, an automatic concept drift detecting mechanism is used in aCVFDT. In our experiments, the hyperplane problem and two benchmark problems from the UCI KDD Archive (Hettich and Bay, xxxx), namely Network Intrusion and Forest CoverType, are used to validate the performance of aCVFDT with significantly improved results compared to CVFDT.

The rest of this paper is organized as follows. The next section describes the related work. Section 3 presents our extensions to CVFDT leading to the proposed aCVFDT. We detail how to integrate ambiguities into CVFDT and how to automatically detect concept drift. An overview of the implementation of aCVFDT and the method for making class predictions by aCVFDT are also presented. The experimental evaluation is described in Section 4. The last section presents our conclusions. For implementation purposes, the aCVFDT pseudo-codes is listed in Appendix A.

2. Related work

Decision trees are one of the most popular classification methods currently in use today. A traditional decision tree is learned by recursively replacing leaves with test nodes, starting from the root. The attribute to test at a node is chosen by comparing all available attributes and selecting the best one according to certain heuristic measure. To compute the heuristic measures, all training data must be available, which is not applicable for streaming data. To solve this problem, Domingos and Hulten (2000) adopted the Hoeffding bound (Hoeffding, 1963; Maron and Moore, 1994) in the growing process of decision trees, allowing decision trees to be built incrementally. This was is a significant contribution to the application of decision trees to the field of streaming data mining.

Hulten et al. (2001) considered the problem of concept drift in VFDT leading to CVFDT. CVFDT is one of the best-known systems that can efficiently classify streaming data. It represents a significant methodology that continuously adapts the prediction model to incoming examples. However, upon identifying a concept change, CVFDT builds a new prediction model from scratch. In cases where history repeats itself, CVFDT cannot take advantage of previous experiences and hence may be less efficient than desired. In the gap where the old model is outdated and the new model has not yet matured, the prediction accuracy may be severely impaired (Yang et al., 2005).

To further improve VFDT, Gama et al. (2003) extended the VFDT system in two directions: the ability to deal with continuous data and the use of more powerful classification techniques at tree leaves. The proposed system, VFDTc, can incorporate and classify new information online, with a single scan of the data, in a constant time per example.

Furthermore, Gama et al. (2004) applied the main idea of VFDT, i.e., the Hoeffding bound, to build a forest of trees from data streams. A hybrid adaptive system for the induction of a forest of trees from data streams, namely the Ultra Fast Forest Tree system (UFFT) was proposed. UFFT is an incremental algorithm that offers constant time performance to process each example, works online, and uses the Hoeffding bound to decide when to install a splitting test at a leaf leading to a decision node. It uses analytical techniques to choose the splitting criteria and uses the information gain to estimate the merit of each possible splitting test. UFFT was extended in (Gama et al., 2005), which added the ability to detect concept drift.

VFDT has also been used for some practical applications. Wang et al. (2006) applied VFDT to the link quality estimation problem in the context of wireless sensor networks. Li et al. (2007) used CVFDT to provide recommendations using collaborative filtering over an incoming data stream that contained user ratings for various items.

Many other approaches have been proposed to cope with streaming data mining and concept-changing scenarios. More comprehensive reviews can be found in (Gaber et al., 2005; Keogh and Kasetty, 2002; Tsymbalb, 2004).

3. Ambiguous CVFDT

The classification problem is generally defined as follows (Hulten et al., 2001). A set of N training examples of the form $(\boldsymbol{X}, y)$ is assumed, where y is a discrete class label and $\boldsymbol{X}$ is a vector of d attributes, each of which may be symbolic or numeric. In this paper, only symbolic attributes are considered. The goal is to produce a model $y = f(\boldsymbol{X})$ from these examples which can predict the class labels of future examples with high prediction accuracies. In the following, we first offer a brief overview of CVFDT and we then present our contributions.

3.1. CVFDT

To build decision trees on streaming data, the first examples must be used to choose the root test; once the root attribute is chosen, the succeeding examples will be passed down to the corresponding leaves and used to choose the appropriate attributes there, and so on, recursively. Domingos and Hulten (2000) solved the difficult problem of deciding exactly how many examples are necessary at each node using a statistical result known as the Hoeffding bound (Hoeffding, 1963; Maron and Moore, 1994). Consider a real-valued random variable r whose range is R. Suppose we have made n independent observations of this variable and computed their mean $\bar{r}$. The Hoeffding bound states that, with probability $1 - \delta$, the true mean of the variable is at least $\bar{r} - \varepsilon$, where

$$\varepsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n}} \tag{1}$$

Let $G(X_i)$ be the heuristic measure used to choose test attributes. The goal is to ensure that, with high probability, the attribute chosen using n examples is the same as that chosen using infinite examples. Assume G is to be maximized, and let X_a be the attribute with highest observed $\overline{G}$ after seeing n examples and X_b be the second-best attribute. Let $\triangle\overline{G} = \overline{G}(X_a) - \overline{G}(X_b) \geqslant 0$ be the difference between their observed heuristic values. Then, given a desired δ, the Hoeffding bound guarantees that X_a is the correct choice with probability $1 - \delta$ if n examples have been seen at this node and $\triangle\overline{G} > \varepsilon$. At this point, the node can be split using the current best attribute, and succeeding examples will be passed to the new leaves.

CVFDT uses the above method to select an attribute at each node and works by keeping its model consistent with a sliding window of examples. Each time a new example arrives, CVFDT updates the sufficient statistics at each node by increasing the counts corresponding to the new example and decreasing the counts corresponding to the oldest example in the window. If the concept is changing, some splits that previously passed the Hoeffding test will no longer be true, because an alternative attribute now exhibits a higher gain or the two attributes are too close to differentiate. Therefore, CVFDT periodically scans the tree and all alternate subtrees to search for internal nodes whose sufficient statistics indicate that there are new attributes that would lead to higher prediction accuracy than the chosen split attribute. An alternate subtree is started at every such node. When the alternate subtrees become more accurate on new data than the old subtrees, they are used to replace the old ones. For more details about CVFDT, refer to (Domingos and Hulten, 2000; Hulten et al., 2001).

3.2. Integrating ambiguities into CVFDT

In real world situations, explanations for identical events maybe different depending on perceptions, or they may change with time. This phenomenon incurs ambiguities. When a decision needs to be made, current context and information must be taken into account to eliminate ambiguities and select the most suitable explanation. For a learning algorithm, it is useful if the learned model can reflect ambiguities and make the best decision based on current context and new data.

In CVFDT, each node splits only on one attribute. Thus, at anytime, it can only reflect one concept. If each node is allowed to have multiple options, that is, if a node can split on multiple attributes, then the tree can reflect multiple concepts, which can be in different time periods, different perceptions, etc. Based on the above notion, a node of an ambiguous decision tree can be defined as follows:

aCVFDTnode = {

$\boldsymbol{X}'$:	The set of multiple options, which is a subset of $\boldsymbol{X}$. This node can split on each element of $\mathbf{X}'$; X'^{i} is the ith element of $\boldsymbol{X}'$;
$\boldsymbol{n}_{ijk}$	The set of sufficient statistics used to compute heuristic measures for each element of $\boldsymbol{X}'$; n^{l}_{ijk} corresponds to the lth element of $\mathbf{X}'$;
ST:	The starting time. ST^{i} denotes the time at which X'^{i} enters into $\boldsymbol{X}'$;
Acc:	The prediction accuracy. Acc^{i} denotes the prediction accuracy of X'^{i};
LT:	The most recent time instant at which the prediction accuracy was improved. LT^{i} denotes the most recent time that the prediction accuracy of X'^{i} increased;

}

$\boldsymbol{X}'$ offers multiple options for each node. At the beginning, $\boldsymbol{X}'$ is empty for each node. When examples arrive, if the Hoeffding bound test indicates that a certain attribute can be used, that attribute will enter $\boldsymbol{X}'$, and the corresponding information will be recorded at $\boldsymbol{n}_{ijk}$ and at ***ST***, as appropriate. From then on, ***Acc*** and ***LT*** are updated with the incoming examples. A child node of ***aCVFDTnode*** that corresponds to each value of that attribute will be created. When the algorithm detects that certain attributes perform better but are not yet in $\boldsymbol{X}'$, they are added into $\boldsymbol{X}'$.

ST and ***Acc*** can be used to predict class labels. In other words, decisions can be made by selecting one element according to the measures computed based on starting times and prediction accuracies. Thus, when users take advantage of the learned model to make class predictions, they can select the newest element, or the element with highest prediction accuracy, or combine the two cases using different weights consistent with their requirements.

If attributes are continually added to $\boldsymbol{X}', \boldsymbol{X}'$ may grow too large. In the end, it will include all attributes or at least contain very old knowledge that is no longer useful. Thus, ***LT*** is recorded to check the most recent time instant at which the prediction accuracy of each element of $\boldsymbol{X}'$ last increased. If the prediction accuracies of some elements do not increase for a long time, we conclude that these options are not suitable to describe the most recent concepts, and it is therefore better to remove them.

In the aforementioned ***aCVFDTnode***, we record information about sufficient statistics, starting time, prediction accuracy, and the most recent updating time. The information can be used to maintain multiple options during the tree building process or to make class predictions, etc. For different applications, contextual domain information can also be recorded so that the tree will reflect additional alternative concepts.

To further compare the differences between aCVFDT and CVFDT, we summarize the information recoded by a CVFDT node as follows:

CVFDTnode = {

X':	The element of $\boldsymbol{X}$ on which this node splits;
$\boldsymbol{X}^{c}$:	The candidate set for this node, which is a subset of $\boldsymbol{X}$. Once the Hoeffding bound test indicates that an element in $\boldsymbol{X}^{c}$ exceeds X', the new element replaces X';
n'_{ijk}:	The sufficient statistics for X';
$\boldsymbol{n}^{c}_{ijk}$:	The set of sufficient statistics for each element of $\boldsymbol{X}^{c}$;
Acc':	The prediction accuracy of X';
$\boldsymbol{Acc}^{c}$:	The prediction accuracy of each element of $\boldsymbol{X}^{c}$.

}

In general, the information recorded in a ***CVFDTnode*** can be divided into two parts, namely that for the current splitting element and that for the candidate set. In contrast to an ***aCVFDTnode***, although the candidate set in a ***CVFDTnode*** may contain multiple choices, only one can be selected as the current splitting element. That is to say, there exists only one element on which a ***CVFDTnode*** can split, whereas there is a set of elements on which an ***aCVFDTnode*** can split. The elements of a ***CVFDTnode***'s candidate set can come into effect only after they have been selected. Moreover, there is no information about start time and the most recent time in a ***CVFDTnode***, and the candidate elements are selected based solely on prediction accuracies. On the other hand, an ***aCVFDTnode*** can be considered as syncretizing the two information elements in a ***CVFDTnode*** so that the candidate set can come into effect as soon as possible to reduce the effect of concept drift. To make the concept clearer, these two kinds of trees are compared in Fig. 1. The dark nodes can be used to predict class labels in current time, while the grey ones cannot.

3.3. Automatically detecting concept drift

Generally, when concept drift occurs, the prediction accuracy of the learned model drops sharply. In most cases, the faster an algorithm can detect such a drop and take action to adjust the learned model, the better the algorithm's performance. Let the current learned model be ***AT***, namely, an ambiguous decision tree. ***AT*** grows dynamically. At any time point, while each training example

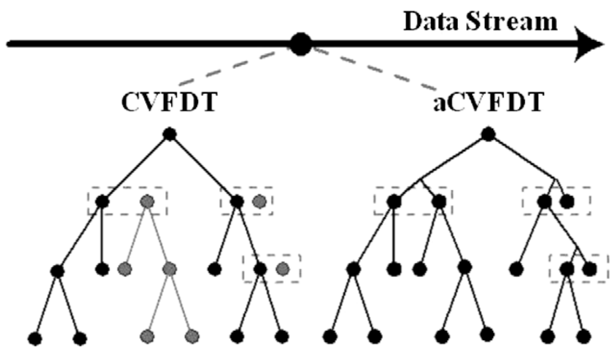


Fig. 1. Comparison between CVFDT and aCVFDT.

successively arrives, ***AT*** can first be used to predict its class label, and then the label becomes apparent. Thus, the prediction accuracy of ***AT*** is changed. This change can be used to detect the occurrences of concept drift. Accordingly, we use the following method, which was first introduced for Bayesian classifiers (Orrego, 2004).

Suppose the prediction accuracy of ***AT*** is p_0. Let ***AT*** be used to predict the class labels of m incoming examples. Then, the prediction accuracy exhibits the binomial distribution $b(m, p_0)$. The null hypothesis, H_0, states that there is no change in the performance of ***AT***, which means that the current mean prediction accuracy p will not drop compared with p_0. The alternate hypothesis, H_1, states that the mean prediction accuracy has dropped, that is, $p < p_0$. The decision rule is that we reject H_0 and accept H_1 if the mean prediction accuracy p for m incoming examples drops with a significance level of $\alpha = 0.05$. Since under $H_0, \mu(= mp)$ is $N(mp_0, mp_0(1 - p_0))$ after the binomial distribution is approximated to the normal, the critical region is calculated as follows:

$$\frac{\mu - \mu_0}{\sigma/\sqrt{m}} \leqslant -z(\alpha) \tag{2}$$

where $\mu_0 = mp_0$ and $\sigma = mp_0(1 - p_0)$.

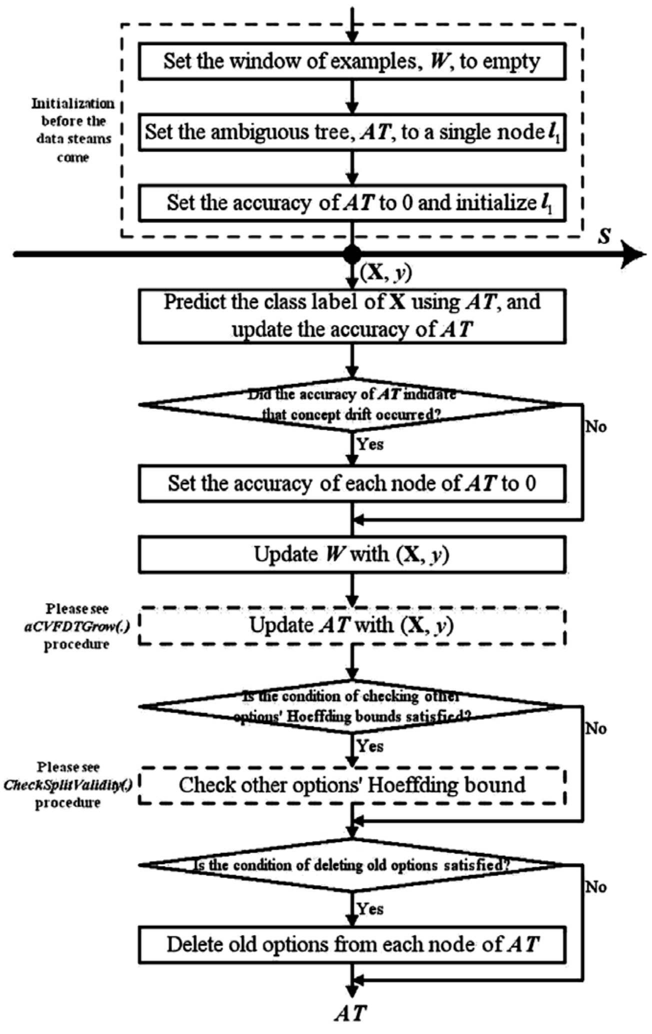
Eq. (2) is termed the standardized test statistics. If the observed value $z = (\mu - \mu_0)/(\sigma/\sqrt{m})$ is less than $-z(\alpha) = -z(0.05) = -1.645$, we reject H_0 in favor of H_1: $\mu < \mu_0$. In other words, concept drift occurs. Otherwise, if z is greater than $-z(\alpha)$, then there is insufficient evidence to reject H_0, which suggests there is no concept drift.

3.4. Implementation of ambiguous CVFDT

On the basis of the above considerations, aCVFDT is implemented as follows. The initialization is first executed, and then examples from data stream ***S*** are continuously processed. When each training example arrives, its class label is first predicted to update the prediction accuracy of ***AT***. The resulting prediction accuracy of ***AT*** is used to check whether concept drift occurs using the method discussed in Section 3.3. The predicting method is the same as using ***AT*** to predict the class labels of new examples, and it will be described in Section 3.5. The flowcharts of the aCVFDT algorithm and two important procedures, namely *aCVFDTGrow*$(\cdot)$ and *CheckSplitValidity*$(\cdot)$, are shown in Figs. 2–4, respectively.

When concept drift is detected, we want to find the element of each node's $\boldsymbol{X}'$ that is most suitable for the current concept. Thus, it is useful to let each element of $\boldsymbol{X}'$ be free from the effect of old training examples. Therefore, *ClearATnodeAccuracy*$(\cdot)$ procedure is executed to assign all elements of ***Acc*** to 0 when the algorithm identifies any concept drift. This procedure benefits from the automatic concept drift detecting mechanism, and it does not exist in CVFDT.

When a new example $(\boldsymbol{X}, y)$ arrives, it should be used to grow the current ***AT***, and accordingly the *aCVFDTGrow*$(\cdot)$ procedure is used. This procedure is different from that of CVFDT since each node has multiple options. If ***aCVFDTnode***$(\boldsymbol{X}')$ is not an empty set, we conclude that this node has been previously split on one or more attributes. To measure the quality of each option, each subtree rooted at one element of ***aCVFDTnode***$(\boldsymbol{X}')$ is used to predict the class label of $\boldsymbol{X}$, and the outcomes are used to update ***aCVFDTnode***(***Acc***) and ***aCVFDTnode***(***LT***). Afterwards, $(\boldsymbol{X}, y)$ is used to update the sufficient statistics of each option, and the procedure is executed iteratively on the option according to $\boldsymbol{X}$'s attribute values. If ***aCVFDTnode***$(\boldsymbol{X}')$ is still an empty set, we conclude that this node has not been split on any attribute, and the Hoeffding bound should be computed to check whether or not to undertake a splitting operation. When this node is split, each of its child nodes is



Initialization before the data steams come
Set the window of examples, W, to empty
Set the ambiguous tree, AT, to a single node l1
Set the accuracy of AT to 0 and initialize l1
S
(X, y)
Predict the class label of X using AT, and update the accuracy of AT
Did the accuracy of AT indidate that concept drift occurred?
Yes
No
Set the accuracy of each node of AT to 0
Update W with (X, y)
Please see aCVFDTGrow(.) procedure
Update AT with (X, y)
Is the condition of checking other options' Hoeffding bounds satisfied?
Yes
No
Please see CheckSplitValidity() procedure
Check other options' Hoeffding bound
Is the condition of deleting old options satisfied?
Yes
No
Delete old options from each node of AT
AT

Fig. 2. Flowchart for the aCVFDT algorithm.

still an ***aCVFDTnode***, and each needs to be initialized according to the information recorded at the appropriate ***aCVFDTnode***.

When the algorithm detects concept drift, it needs more examples to see whether other options would pass the Hoeffding bound test. The *CheckSplitValidity*$(\cdot)$ procedure is executed at every 5000-example point in the subsequently 20,000 examples. Since each node has multiple options, and because each option has a sufficient statistic, we should first determine which sufficient statistic is suitable for computing new options. Therefore, the most recent option with the highest prediction accuracy is assumed to be the best. Only those attributes not in ***aCVFDTnode***$(\boldsymbol{X}')$ are taken into account.

To prevent the set of options from growing too large, the *DeleteOldOptions*$(\cdot)$ procedure is executed every *oc* examples. The parameter *op* is used to control the frequency with which obsolete options are deleted. The greater the value of *op*, the older the options that can be preserved in the learned model. The implementation details of all procedures, namely *AmbiguousCVFDT*$(\cdot)$, *ClearATnodeAccuracy*$(\cdot)$, *aCVFDTGrow*$(\cdot)$, *CheckSplitValidity*$(\cdot)$, and *DeleteOldOptions*$(\cdot)$, are listed as pseudo-code in Appendix A.

3.5. Class predictions using aCVFDT

In traditional decision trees, an internal node only splits on a single attribute. Thus, when making class predictions, it only needs to pass the example from the root down to a leaf, test the appropri-

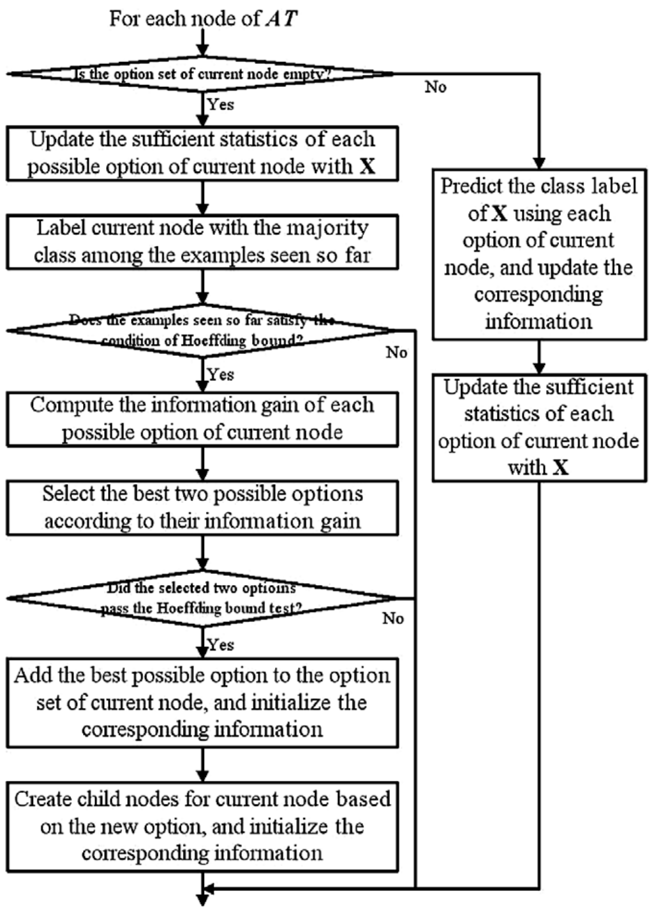

For each node of AT
Is the option set of current node empty?
Yes
No
Update the sufficient statistics of each possible option of current node with X
Label current node with the majority class among the examples seen so far
Does the examples seen so far satisfy the condition of Hoeffding bound?
Yes
No
Compute the information gain of each possible option of current node
Select the best two possible options according to their information gain
Did the selected two optioins pass the Hoeffding bound test?
Yes
No
Add the best possible option to the option set of current node, and initialize the corresponding information
Create child nodes for current node based on the new option, and initialize the corresponding information
Predict the class label of X using each option of current node, and update the corresponding information
Update the sufficient statistics of each option of current node with X

Fig. 3. Flowchart for the *aCVFDTGrow*($\cdot$) procedure.

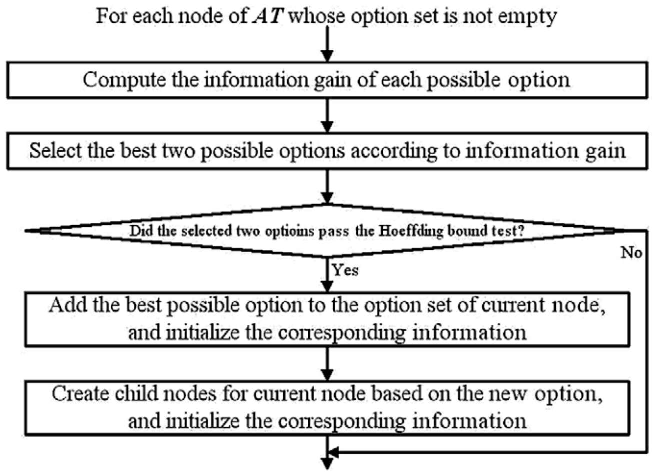

For each node of AT whose option set is not empty
Compute the information gain of each possible option
Select the best two possible options according to information gain
Did the selected two optioins pass the Hoeffding bound test?
Yes
No
Add the best possible option to the option set of current node, and initialize the corresponding information
Create child nodes for current node based on the new option, and initialize the corresponding information

Fig. 4. Flowchart for the *CheckSplitValidity*($\cdot$) procedure.

ate attributes at each node, and follow the branch corresponding to the attribute's value in the example. However, in an ambiguous decision tree, each internal node can split on one or more attributes, that is, it can split on each element of $\boldsymbol{X}'$. Thus, when using an ambiguous decision tree to make class predictions, at each internal node, the element of $\boldsymbol{X}'$ to be used should be determined first. Usually, the decisions can be made based on the information recorded at the ***aCVFDTnode***, such as starting times, prediction accuracies, etc. As indicated in Section 3.2, the algorithm can use one of these pieces of information or it can combine them with different weights, depending on the types of concepts that users wish to specify. Here, we use the most recent element that offers the highest prediction accuracy. The details are summarized in Algorithm 1. The prediction can start at any internal node similar to that in *aCVFDTGrow*($\cdot$).

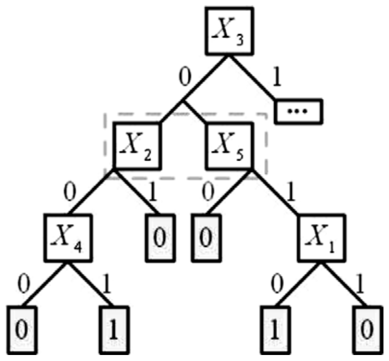


Fig. 5. An example of aCVFDT for class predictions.

To clarify the special prediction method of aCVFDT, we offer a simple example. Suppose that $\boldsymbol{X} = \{X_1, X_2, X_3, X_4, X_5\}$, and the value of each attribute is 0 or 1. The current aCVFDT is shown in Fig. 5, where the grey nodes are leaf nodes with predicted class labels, and the right branch of the root node, namely "X_3" is omitted due to limited space. We use this ambiguous decision tree to predict the class label of the example $\boldsymbol{E} = \{X_1 = 1, X_2 = 0, X_3 = 0, X_4 = 1, X_5 = 1\}$. Here, the left child node of "X_3" has two options, namely "X_2" and "X_5". Thus, we need to select one based on the most recent prediction accuracies. If the most recent prediction accuracy of "X_2" is better than that of "X_5", then "X_2" is selected, and the following node is "X_4" which results in a class label of "1". Similarly, if the most recent prediction accuracy of "X_5" is better than that of "X_2", then "X_5" is selected, and the subsequent node is "X_1" which results in a class label of "0". Clearly, these are two different prediction results that depend entirely on the option selected.

Algorithm 1: Prediction method using ambiguous CVFDT

```
Inputs:   StartNode: The root
  X: An unlabeled example
Output: y: The predicted class label of X

1:  Assign StartNode to CurrentNode
2:  While CurrentNode is not empty
3:  If CurrentNode($X'$) is an empty set
4:     Let $y$ be the majority class of CurrentNode
5:     Let CurrentNode be empty
6:  Else
7:     Let CurrentNode($X'^{best}$) be the most recent
       element with the highest prediction accuracy
       among CurrentNode($X'$)
8:     Assign the child node corresponding to the
       value of CurrentNode($X'^{best}$) in X to
       CurrentNode
```

4. Experimental studies

The algorithm aCVFDT was implemented using Visual C++ 6.0[1]. A series of experiments comparing aCVFDT with CVFDT was conducted, and three well-known benchmark datasets, namely

[1] The source code can be downloaded at http://see.xidian.edu.cn/faculty/liujing/.

hyperplane, Network Intrusion Hettich and Bay, xxxx, and Forest CoverType Hettich and Bay, xxxx, were used. The parameters of aCVFDT were set as follows: information gain was used as the G function, $\delta = 0.0001, \tau = 0.05, w = 100,000, n_{min} = 300, oc = 50,000$, and $op = 20,000$. The parameters of CVFDT were the same as those of (Hulten et al., 2001).

4.1. Hyperplane

We used the method in (Wang et al., 2003) to generate synthetic data: synthetic data with drifting concepts are generated based on a moving hyperplane. A hyperplane in d-dimensional space is denoted by:

$$\sum_{i=1}^{d} a_i x_i = a_0 \tag{3}$$

The examples satisfying $\sum_{i=1}^{d} a_i x_i \geqslant a_0$ are labeled as positive, and those satisfying $\sum_{i=1}^{d} a_i x_i < a_0$ are labeled as negative. Since the orientation and the position of the hyperplane can be changed in a smooth manner by altering the magnitude of weights, hyperplanes are suitable for simulating concept drift in data streams (Hulten et al., 2001).

Examples are generated uniformly in a d-dimensional unit hypercube, that is, $[0,1]^d$. They are then labeled using the concept, and their continuous attributes are uniformly discretized into 5 bins. Weights a_i $(1 \leqslant i \leqslant d)$ in (3) are initialized with random values in the range [0,1]. The value of a_0 is chosen so that the hyperplane cuts the multi-dimensional space into two parts of the same volume, namely, $a_0 = \frac{1}{2}\sum_{i=1}^{d} a_i$.

Concept drift is added to the datasets using a series of parameters. Parameter k specifies the total number of dimensions whose weights are involved in changing. Parameter $t \in \Re$ and $s_i \in \{-1, 1\}$ specify the magnitude and direction of the change for weights $a_1, a_2, \cdots, a_k$, respectively. The probability of assigning s_i to -1 is p_s%. Weights change every N_c examples, i.e., $(s_i \cdot t)$ is added to a_i after every additional N_c examples. In our experiments, five million training examples and a test set of 50,000 examples are generated independently. Also, each time the weights are updated, we re-compute $a_0 = \frac{1}{2}\sum_{i=1}^{d} a_i$ so that the class distribution is not disturbed, and the test set is relabeled.

We examined how aCVFDT responds to changing levels of concept drift on four datasets with $d = 10$. Changing levels of concept drift are adjusted by parameter k, and $k = 2, 4, 6$, and 8. Our experimental results are illustrated in Fig. 6. The reported prediction accuracies are obtained by testing the accuracy of the learned model every 10,000 examples throughout the run and averaging these results. For $k = 2$ and 4, the prediction accuracies of CVFDT and aCVFDT are similar. But for $k = 6$ and 8, aCVFDT clearly outperforms CVFDT. The average percent accuracies of aCVFDT and CVFDT are 73.5% and 70.0% for $k = 6$, 79.2% and 71.6% for $k = 8$, respectively. Since the greater the value of k, the more serious is the concept drift, these results demonstrate that the mechanism used to handle concept drift in aCVFDT is more effective than that in CVFDT. To further illustrate the performance of aCVFDT in handling serious concept drift, a detailed view of the run when $k = 8$ is given in Fig. 7. This result shows that aCVFDT nearly always outperforms CVFDT, especially when the number of examples is greater than 500,000.

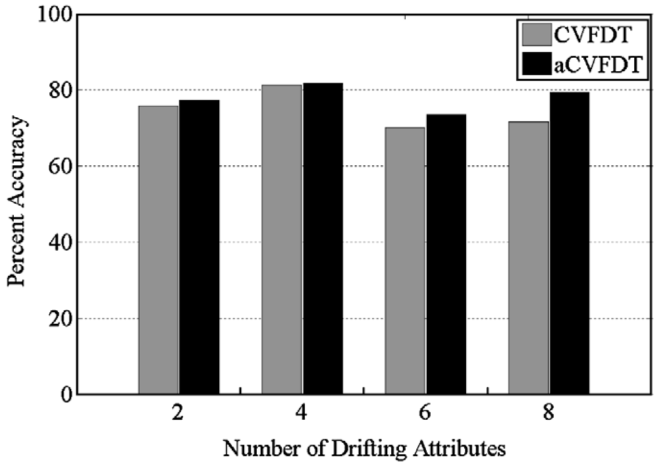


Fig. 6. Percent accuracies for different changing levels of concept drift.

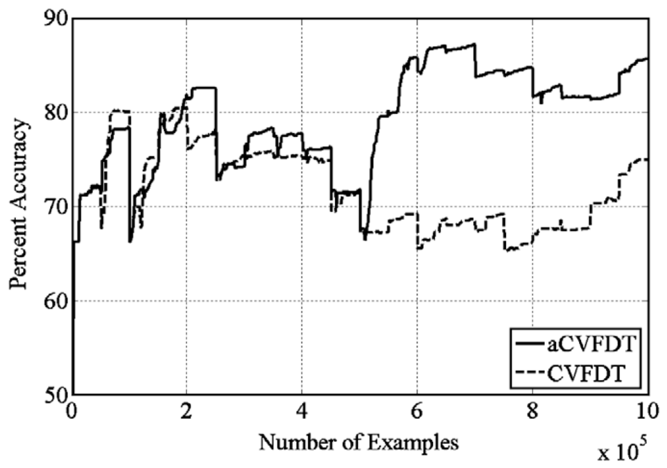


Fig. 7. Incremental results on hyperplane for $k = 8$.

4.2. Network intrusion

Network intrusion is a benchmark problem in the UCI KDD Archive (Hettich and Bay, xxxx), which was also used for the third International KDD Tools Competition. It includes a wide variety of simulated intrusions in a military network environment. The task is to build a prediction model that is capable of distinguishing between normal connections and network intrusions. In this experiment, we let the training examples arrive one by one to form a data stream. Thus, in different periods, different intrusion classes may occur, which is equivalent to the situation of concept drift. Since the original test dataset does not change and cannot reflect concept drift, we use the training data as the test datasets. To let the test datasets be independent of the training data, we test the prediction accuracy of the learned model every 5000 examples throughout the run and we use the subsequent 10,000 examples after that point as the test data. Our experimental results are shown in Fig. 8.

Fig. 8 demonstrates that as data arrive, although the prediction accuracies of both aCVFDT and CVFDT fluctuate, aCVFDT clearly outperforms CVFDT since when it does drop, it always drops less than CVFDT. This demonstrates that aCVFDT can detect concept drift promptly and can respond to the situation much more quickly than CVFDT. The average percent accuracy of aCVFDT is 98.9%, while that of CVFDT is 97.4%, which also confirms that aCVFDT outperforms CVFDT.

To further check whether this improvement is statistically important, we use the paired t-test to compare the difference between 100 pairs of prediction accuracies from aCVFDT and from CVFDT. The null hypothesis states that there is no actual difference between aCVFDT and CVFDT. The obtained t-value with 99 degrees of freedom is 3.283, which is statistically significant at $\alpha = 0.05$

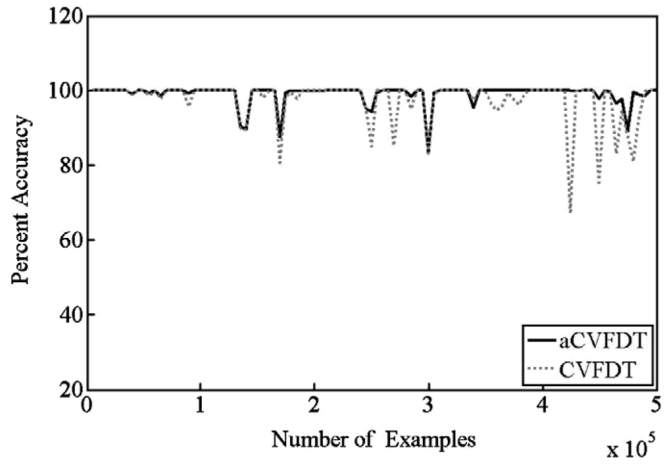


Fig. 8. Incremental results on Network Intrusion.

using a two-tailed test. The null hypothesis is clearly rejected, and the improvement of aCVFDT over CVFDT is statistically meaningful.

4.3. Forest CoverType

Forest CoverType is another benchmark problem in the UCI KDD Archive (Hettich and Bay, xxxx). This problem relates to the actual forest cover type for a given observation (30×30 m cell) that was determined from US Forest Service (USFS) Region to Resource Information System (RIS) data. Our task is to predict forest cover type from cartographic variables alone (no remotely sensed data). In this experiment, we also allow the training examples to arrive one by one to form a data stream and we test the prediction accuracy of the learned model every 5000 examples throughout the run using the subsequent 10,000 examples as the test data. The experimental results are shown in Fig. 9.

Fig. 9 demonstrates that as data arrive, although the prediction accuracies of both aCVFDT and CVFDT fluctuate, aCVFDT outperforms CVFDT. The average percent accuracy of aCVFDT is 65.7%, while that of CVFDT is 62.9%. To further check whether this improvement is statistically important, we also use the paired t-test to compare the difference between 100 pairs of prediction accuracies from aCVFDT and from CVFDT. The obtained t-value with 99 degrees of freedom is 4.057, which is statistically significant at $\alpha = 0.05$ using a two-tailed test. We therefore conclude that the improvement of aCVFDT over CVFDT is statistically meaningful.

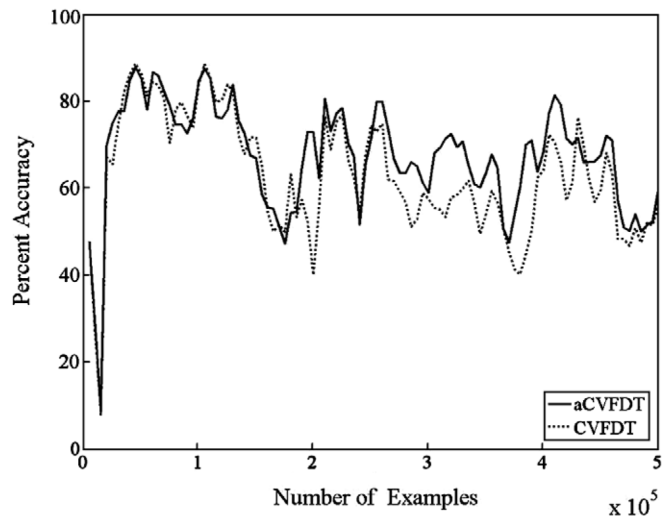


Fig. 9. Incremental results on Forest CoverType.

5. Conclusions

In this paper, we integrate ambiguities and automatic detection of concept drift into CVFDT, leading to a new algorithm, namely aCVFDT, that is more sensitive to concept drift. In contrast to traditional decision tree approaches, an ambiguous decision tree generated by aCVFDT can incorporate one or more options at each inner node, reflecting the best performing and most recent knowledge. These options can reflect knowledge corresponding to different time periods or to relevant contextual domain knowledge that is preferred by users. Benefiting from this structure, aCVFDT can reexamine and reuse old concepts when they recur. When an ambiguous decision tree is used to make class predictions, it can select knowledge according to user preferences. aCVFDT offers a new way to accommodate ambiguities in decision-making processes. Our experimental results demonstrate that aCVFDT offers significant performance advantages over CVFDT.

Acknowledgement

This work was supported by the National Natural Science Foundation of China under Grant 60872135, the Program for New Century Excellent Talents in University of China under Grant NCET-06-0857, and the Australian ARC Discovery Project under Grant DP0558879.

Appendix A. Pseudo-code for ambiguous CVFDT

A.1. AmbiguousCVFDT procedure

AmbiguousCVFDT.

```
Inputs: $\boldsymbol{S}$: A sequence of examples
        $\boldsymbol{X}$: A set of symbolic attributes
        $G(\cdot)$: A split evaluation function
        $\delta$: One minus the desired probability of choosing the correct attribute at a given node
        $\tau$: A user-supplied tie threshold
        $w$: The size of the window
        $n_{min}$: The number of examples between checks for growth
        $oc$: The number of examples between checks for deleting old options
        $op$: Delete the options whose prediction accuracy has not increased over $op$ examples
Output: $\boldsymbol{AT}$: An ambiguous decision tree

1: Let $\boldsymbol{W}$ be the window of examples, initially empty
2: Let $\boldsymbol{AT}$ be an ambiguous tree with a single node $\boldsymbol{l}_1$, the root, and $Acc(\boldsymbol{AT})$ be 0
3: Let $\boldsymbol{l}_1(\boldsymbol{X}'), \boldsymbol{l}_1(\boldsymbol{ST}), \boldsymbol{l}_1(\boldsymbol{Acc}), \boldsymbol{l}_1(\boldsymbol{LT})$ be empty sets, and $\boldsymbol{l}_1(\boldsymbol{n}_{ijk}) = \{\boldsymbol{l}_1(n^1_{ijk})\}$
4: Let $\overline{G}_{l_1}(X_{ptyset})$ be $\overline{G}$ obtained by predicting the most frequent class in $\boldsymbol{S}$
5: Let $\boldsymbol{X}_{l_1} = \boldsymbol{X} \cup \{X_{ptyset}\}$
6: For each class $y_k$ and each value $x_{ij}$ of each attribute $X_i \in \boldsymbol{X}$
7:     Let $\boldsymbol{l}_1(n^1_{ijk}) = 0$
8: For each example $(\boldsymbol{X}, y)$ in $\boldsymbol{S}$
9:     Predict the class label of $\boldsymbol{X}$ using $\boldsymbol{AT}$, and update $Acc(\boldsymbol{AT})$
10:    If $Acc(\boldsymbol{AT})$ indicates that concept drift occurs (Section 3.3)
11:        $ClearATnodeAccuracy(\boldsymbol{l}_1)$
12:    Add $(\boldsymbol{X}, y)$ into $\boldsymbol{W}$ and perform operations that correspond to those in CVFDT
13:    $aCVFDTGrow(\boldsymbol{l}_1, n, G, (\boldsymbol{X}, y), \delta, n_{min}, \tau)$
```

(*continued on next page*)

```
14: If $Acc(\boldsymbol{AT})$ indicates that concept drift has occurred within the past 20,000 examples, and if 5000 examples have been assessed since the last check
15:   $CheckSplitValidity(\boldsymbol{l}_1, n, \delta)$
16: If there have been $oc$ examples since the last checking
17:   $DeleteOldOptions(\boldsymbol{l}_1, op)$;
18: Return $\boldsymbol{AT}$.
```

A.2. ClearATnodeAccuracy procedure

ClearATnodeAccuracy(aCVFDTnode).

```
1: Let each element of $\boldsymbol{aCVFDTnode}(\boldsymbol{Acc})$ be 0
2: For each $\boldsymbol{aCVFDTnode}(X'^{i})$
3:   For each child node $\boldsymbol{cnode}$ of $\boldsymbol{aCVFDTnode}(X'^{i})$
4:     $ClearATnodeAccuracy(\boldsymbol{cnode})$
```

A.3. aCVFDTGrow procedure

$aCVFDTGrow(\mathbf{aCVFDTnode}, n, G, (X, y), \delta, n_{min}, \tau)$.

```
1: If $\boldsymbol{aCVFDTnode}(\boldsymbol{X}')$ is not an empty set
2:   Predict the class label of $\boldsymbol{X}$ using trees whose roots are at each element of $\boldsymbol{aCVFDTnode}(\boldsymbol{X}')$, and update the corresponding elements of $\boldsymbol{aCVFDTnode}(\boldsymbol{Acc})$ and $\boldsymbol{aCVFDTnode}(\boldsymbol{LT})$
3:   For each element of $\boldsymbol{aCVFDTnode}(\boldsymbol{X}')$
4:     For each $x_{ij}$ in $\boldsymbol{X}$ such that $X_i \in \boldsymbol{X}_{aCVFDTnode}$
5:       Increment $\boldsymbol{aCVFDTnode}(n^{i}_{ijk})$
6:   For the child node, $\boldsymbol{cnode}$, of $\boldsymbol{aCVFDTnode}(X'^{i})$ for the value of attribute $X_i$ in $\boldsymbol{X}$
7:     $aCVFDTGrow(\boldsymbol{cnode}, n, G, (\boldsymbol{X}, y), \delta, n_{min}, \tau)$
8: Else if $\boldsymbol{aCVFDTnode}(\boldsymbol{X}')$ is an empty set
9:   For each $x_{ij}$ in $\boldsymbol{X}$ such that $X_i \in \boldsymbol{X}_{aCVFDTnode}$
10:     Increment $\boldsymbol{aCVFDTnode}(n^{1}_{ijk})$
11:   Label $\boldsymbol{aCVFDTnode}$ with the majority class among the examples seen so far
12:   Let $n_{aCVFDTnode}$ be the number of examples seen at $\boldsymbol{aCVFDTnode}$
13:   If the examples seen so far at $\boldsymbol{aCVFDTnode}$ are not all of the same class and if $(n_{aCVFDTnode}$ mod $n_{min})$ is 0, then
14:     Compute $\triangle\overline{G}_{aCVFDTnode}(X_i)$ for each $X_i \in \boldsymbol{X}_{aCVFDTnode} - \{X_{ptyset}\}$ using the counts $\boldsymbol{aCVFDTnode} n^{1}_{ijk})$
15:     Let $X_a$ be the attribute with highest $\overline{G}_{aCVFDTnode}$
16:     Let $X_b$ be the attribute with second-highest $\overline{G}_{aCVFDTnode}$
17:     Compute $\varepsilon$ using Eq. (1) and $\delta$
18:     Let $\triangle\overline{G}_{aCVFDTnode} = \triangle\overline{G}_{aCVFDTnode}(X_a) - \triangle\overline{G}_{aCVFDTnode}(X_b)$
19:     If $((\triangle\overline{G}_{aCVFDTnode} > \varepsilon)$ or $(\triangle\overline{G}_{aCVFDTnode} \leqslant \varepsilon < \tau))$ and $(X_a \neq X_{ptyset})$, then
20:       Let $\boldsymbol{aCVFDTnode}(\boldsymbol{X}') = \boldsymbol{aCVFDTnode}(\boldsymbol{X}') \cup \{X_a\}$
21:       Let $\boldsymbol{aCVFDTnode}(\boldsymbol{ST}) = \boldsymbol{aCVFDTnode}(\boldsymbol{ST}) \cup$ {Current Time}
22:       Let $\boldsymbol{aCVFDTnode}(\boldsymbol{ACC}) = \boldsymbol{aCVFDTnode}(\boldsymbol{ACC}) \cup \{0\}$
23:       Let $\boldsymbol{aCVFDTnode}(\boldsymbol{LT}) = \boldsymbol{aCVFDTnode}(\boldsymbol{LT}) \cup \{-1\}$
24:       For each value of $X_a$
25:         Create a child node $\boldsymbol{l}_m$ and add it to $\boldsymbol{aCVFDTnode}$
26:         Let $\boldsymbol{l}_m(\boldsymbol{X}'), \boldsymbol{l}_m(\boldsymbol{ST}), \boldsymbol{l}_m(\boldsymbol{Acc}), \boldsymbol{l}_m(\boldsymbol{LT})$ be empty sets, and $\boldsymbol{l}_m(\boldsymbol{n}_{ijk}) = \{\boldsymbol{l}_m(n^{1}_{ijk})\}$
27:         Let $\boldsymbol{X}_{l_m} = \boldsymbol{X}_{aCVFDTnode} - \{X_a\}$
28:         Let $\overline{G}_{l_m}(X_{ptyset})$ be $\overline{G}$ obtained by predicting the most frequent class at $\boldsymbol{l}_m$
29:         For each class $y_k$ and each value $x_{ij}$ of each attribute $X_i \in \boldsymbol{X}_{l_m} - \{X_{ptyset}\}$ Let $\boldsymbol{l}_m(n^{1}_{ijk}) = 0$
```

A.4. CheckSplitValidity procedure

$CheckSplitValidity(\boldsymbol{aCVFDTnode}$, n, $\delta)$.

```
1: If $\boldsymbol{aCVFDTnode}(\boldsymbol{X}')$ is not an empty set
2:   For each element of $\boldsymbol{aCVFDTnode}(\boldsymbol{X}')$
3:     For the each child node $\boldsymbol{cnode}$ of $\boldsymbol{aCVFDTnode}(X'^{i})$
4:       $CheckSplitValidity(\boldsymbol{cnode}, n, \delta)$
5:   Find the best element of $\boldsymbol{aCVFDTnode}(\boldsymbol{X}')$
6:   Compute $\triangle\overline{G}_{aCVFDTnode}(X_i)$ for each $X_i \in \boldsymbol{X}_{aCVFDTnode} - \{X_{ptyset}\} \cup \boldsymbol{aCVFDTnode}(\boldsymbol{X}')$ using the counts $\boldsymbol{aCVFDTnode}(n^{best}_{ijk})$
7:   Let $X_a$ be the attribute with highest $\overline{G}_{aCVFDTnode}$ in the above computation
8:   Let $X_b$ be the attribute with second-highest $\overline{G}_{aCVFDTnode}$ in the above computation
9:   Let $\triangle\overline{G}_{aCVFDTnode} = \triangle\overline{G}_{aCVFDTnode}(X_a) - \triangle\overline{G}_{aCVFDTnode}(X_b)$
10:   If $(\triangle\overline{G}_{aCVFDTnode} \geqslant 0)$
11:     Compute $\varepsilon$ using Eq. (1) and $\delta$;
12:     If $((\triangle\overline{G}_{aCVFDTnode} > \varepsilon)$ or $(\varepsilon < \tau$ and $\triangle\overline{G}_{aCVFDTnode} \geqslant \tau/2))$, then
13:       Let $\boldsymbol{aCVFDTnode}(\boldsymbol{X}') = \boldsymbol{aCVFDTnode}(\boldsymbol{X}') \cup \{X_a\}$
14:       Let $\boldsymbol{aCVFDTnode}(\boldsymbol{ST}) = \boldsymbol{aCVFDTnode}(\boldsymbol{ST}) \cup$ {Current Time}
15:       Let $\boldsymbol{aCVFDTnode}(\boldsymbol{ACC}) = \boldsymbol{aCVFDTnode}(\boldsymbol{ACC}) \cup \{0\}$
16:       Let $\boldsymbol{aCVFDTnode}(\boldsymbol{LT}) = \boldsymbol{aCVFDTnode}(\boldsymbol{LT}) \cup \{-1\}$
17:       For each value of $X_a$
18:         Create a child node $\boldsymbol{l}_m$ and add it to $\boldsymbol{aCVFDTnode}$
19:         Let $\boldsymbol{l}_m(\boldsymbol{X}'), \boldsymbol{l}_m(\boldsymbol{ST}), \boldsymbol{l}_m(\boldsymbol{Acc}), \boldsymbol{l}_m(\boldsymbol{LT})$ be empty sets, and $\boldsymbol{l}_m(\boldsymbol{n}_{ijk}) = \{\boldsymbol{l}_m(n^{1}_{ijk})\}$
20:         Let$\boldsymbol{X}_{l_m} = \boldsymbol{X}_{aCVFDTnode} - \{X_a\}$
21:         Let $\overline{G}_{l_m}(X_{ptyset})$ be $\overline{G}$ obtained by predicting the most frequent class at $\boldsymbol{l}_m$
22:         For each class $y_k$ and each value $x_{ij}$ of each attribute $X_i \in \boldsymbol{X}_{l_m} - \{X_{ptyset}\}$
23:           Let $\boldsymbol{l}_m(n^{1}_{ijk}) = 0$
```

A.5. DeleteOldOptions procedure

DeleteOldOptions(aCVFDTnode, op).

```
1: For each element of $\boldsymbol{aCVFDTnode}(\boldsymbol{X}')$
2:   If $(CurrentTime - \boldsymbol{aCVFDTnode}(LT^{l}) > op)$
3:     Delete $\boldsymbol{aCVFDTnode}(X'^{l}), \boldsymbol{aCVFDTnode}(n^{l}_{ijk}), \boldsymbol{aCVFDTnode}(ST^{l}), \boldsymbol{aCVFDTnode}(Acc^{l}), \boldsymbol{aCVFDTnode}(LT^{i})$ from $\boldsymbol{aCVFDTnode}(\boldsymbol{X}'), \boldsymbol{aCVFDTnode}(\boldsymbol{n}_{ijk})$, $\boldsymbol{aCVFDTnode}(\boldsymbol{ST})$, $\boldsymbol{aCVFDTnode}(\boldsymbol{Acc})$, $\boldsymbol{aCVFDTnode}(\boldsymbol{LT})$, respectively;
```

(continued on next page)

```
4:  Else
5:      For each child node cnode of aCVFDTnode($X^{\prime l}$)
6:          DeleteOldOptions(cnode, op);
```

References

Domingos, P., Hulten, G., 2000. Mining high-speed data streams. In: Proc. 6th ACM SIGKDD Internat. Conf. on Knowledge Discovery and Data Mining, pp. 71–80.

Fan, W., Huang, Y., Yu, P.S., 2004. Decision tree evolution using limited number of labeled data items from drifting data streams. In: Proc. 4th IEEE Internat. Conf. on Data Mining, pp. 379–382.

Gaber, M.M., Zaslavsky, A., Krishnaswamy, S., 2005. Mining data streams: A review. Sigmod Record 34 (2), 18–26.

Gama, J., Rocha, R., Medas, P., 2003. Accurate decision trees for mining high-speed data streams. In: Proc. 9th ACM SIGKDD Internat. Conf. on Knowledge Discovery and Data Mining, pp. 523–528.

Gama, J., Medas, P., Rocha, R., 2004. Forest trees for on-line data. In: Proc. 2004 ACM Symposium on Applied Computing, pp. 632–636.

Gama, J., Medas, P., Rodrigues, P., 2005. Learning decision trees from dynamic data streams. In: Proc. 20th Annual ACM Symposium on Applied Computing, pp. 573–577.

Gama, J., Fernandes, R., Rocha, R., 2006. Decision trees for mining data streams. Intell. Data Anal. 10 (1), 23–45.

Hettich, S., Bay, S.D. The UCI KDD Archive. University of California, Department of Information and Computer Science, Irvine, CA. <http://www.kdd.ics.uci.edu>.

Hoeffding, W., 1963. Probability inequalities for sums of bounded random variables. J. Amer. Stat. Assoc. 58, 13–30.

Hulten, G., Spencer, L., Domingos, P., 2001. Mining time-changing data streams. In: Proc. 7th ACM SIGKDD Internat. Conf. on Knowledge Discovery and Data Mining, pp. 97–106.

Keogh, E., Kasetty, S., 2002. On the need for time series data mining benchmarks: A survey and empirical demonstration. In: Proc. 8th ACM SIGKDD Internat. Conf. on Knowledge Discovery and Data Mining, pp. 102–111.

Klinkenberg, R., 2004. Learning drifting concepts: Example selection vs. example weighting. Intell. Data Anal. 8 (3), 281–300. Special Issue on Incremental Learning Systems Capable of Dealing with Concept Drift.

Kolter, J.Z., Maloof, M.A., 2003. Dynamic weighted majority: A new ensemble method for tracking concept drift. In: Proc. 3rd IEEE Internat. Conf. on Data Mining, pp. 123–130.

Li, X., Barajas, J.M., Ding, Y., 2007. Collaborative filtering on streaming data with interest-drifting. Internat. J. Intell. Data Anal. 11 (1), 75–87.

Maron, O., Moore, A.W., 1994. Hoeffding races: Accelerating model selection search for classification and function approximation. In: Proc. Advances in Neural Information.

Natwichai, J., Li, X., 2004. Knowledge maintenance on data streams with concept drifting. In: Proc. of the 1st Internat. Symposium on Computational and Information Science, pp. 705–710.

Orrego, S., 2004. SAWTOOTH: Learning from Huge Amounts of Data. West Virginia University, USA.

Schlimmer, J.C., RichardII, G.J., 1986. Beyond incremental processing: Tracking concept drift. In: Proc. the AAAI National Conf. on Artifical Intelligence, pp. 502–507.

Street, W., Kim, Y., 2001. A streaming ensemble algorithm (SEA) for large-scale classification. In: Proc. 7th ACM SIGKDD Internat. Conf. on Knowledge Discovery and Data Mining, pp. 377–382.

Tsymbalb, A., 2004. The Problem of Concept Drift: Definitions and Related Work. TCD-CS-2004-15. Computer Science Department, Trinity College Dublin, Ireland.

Wang, H., Fan, W., Yu, P.S., Han, J., 2003. Mining concept-drifting data streams using ensemble classifiers. In: Proc. 9th ACM SIGKDD Internat. Conf. on Knowledge Discovery and Data Mining, pp. 226–235.

Wang, Y., Martonosi, M., Peh, L.S., 2006. Predicting link quality using supervised learning in wireless sensor networks. Mobile Comput. Commun. Rev. 11 (3), 71–83.

Yang, Y., Wu, X., Zhu, X., 2005. Combining proactive and reactive predictions for data streams. In: Proc. 11th ACM SIGKDD Internat. Conf. on Knowledge Discovery and Data Mining, pp. 710–715.