RC23711 (W0508-165) August 31, 2005
Computer Science

IBM Research Report

On Reducing Classifier Granularity in Mining Concept-Drifting Data Streams

Peng Wang[1], Haixun Wang[2], Xiaochen Wu[1], Wei Wang[1], Baile Shi[1]

[1]Fudan University
China

[2]IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

On Reducing Classifier Granularity in Mining Concept-Drifting Data Streams

Peng Wang[†] Haixun Wang[‡] Xiaochen Wu[†] Wei Wang[†] Baile Shi[†]

[†]Fudan University, China {041021057,0124078,weiwang1,bshi}@fudan.edu.cn

[‡]IBM T. J. Watson Research Center, U.S.A. haixun@us.ibm.com

August 26, 2005

Abstract

Using classification models, applications track streaming data to detect actionable alerts, which may include, for example, network intrusions, transaction frauds, biosurveilence abnormality, etc. Due to concept drifts, maintaining a model's uptodateness has become one of the most challenging tasks in mining data streams. State of the art approaches, including both the incrementally updated classifiers and the ensemble classifiers, have proved that model update is a very costly process. In this paper, we introduce the concept of model granularity, and show that reducing model granularity will reduce update cost. Indeed, models of fine granularity enable us to efficiently pinpoint local components in the model that are affected by the concept drift. It also enables us to derive new components that can easily integrate with the model to reflect the current data distribution, thus avoiding avoid expensive updates on a global scale. Experiments on real and synthetic data show that our approach is able to maintain good prediction accuracy at a fraction of model updating cost of state of the art approaches.

1 Introduction

Traditional classification methods work on static data, and they usually require multiple scans of the training data in order to build a model [14, 8, 7]. The advent of new application areas such as ubiquitous computing, e-commerce, and sensor networks leads to intensive research on data streams. In particular, mining data streams for actionable insights has become an important and challenging task for a wide range of applications [5, 10, 15, 3, 9].

State of the art For many applications, the major challenge in mining data streams lies not in the tremendous data volume, but rather, in the time changing nature of the data, a.k.a. the concept drifts [10, 16, 6]. If the data distribution is static, we can always use a subset of the data to learn a fixed model and use it for all future data. Unfortunately, the data distribution is constantly changing, which means the models have to be constantly revised to reflect the current data feature.

It is not difficult to see why model update incurs a major cost. Stream classifiers that handle concept drifts can be roughly divided into two categories. The first category is known as the *incrementally updated classifiers*. The CVFDT approach [10], for example, uses a single decision tree to model streams with concept drifts. However, even a slight drift of the concept may trigger substantial changes in the tree (e.g., replacing old branches with new branches, re-growing or building alternative sub-trees), which severely compromise learning efficiency. Aside from this undesirable aspect, incremental methods are also hindered by their prediction accuracy. This is so because they discard old examples at a fixed rate (no matter if they represent the changed concept or not). Thus, the learned model is supported only by the data in the current window – a snapshot that contains relatively small amount of data. This causes large variances in prediction.

The second category is known as the *ensemble classifiers*. Instead of maintaining a single model, the ensemble approach divides the stream into data chunks of fixed size, and learns a classifier from each of the chunk [16, 6]. To make a prediction, all valid classifiers have to be consulted, which is an expensive process. Besides, the ensemble approach has high model update cost: i) it keeps learning new models on new data, whether it contains concept drifts or not; ii) it keeps checking the accuracy of old models by applying each of them on the new training data. This apparently introduces considerable cost in modeling high speed data streams.

If models are not updated timely because of high update cost, their prediction accuracy will drop eventually. This causes a severe problem, especially to applications that handle large volume streaming data at very high speed.

Model Granularity In this paper, we introduce the concept of model granularity. We argue that state of the art stream classifiers incur a significant model update cost because they use monolithic models that do not allow a semantic decomposition.

Current approaches for classifying stream data are adapted from algorithms designed for static data, for which monolithic models are not a problem. For incrementally updated classifiers, the fact that even a small disturbance from the data may bring a complete change to the model indicates that monolithic models are not appropriate for data streams. The ensemble approach lowered model granularity by dividing the model into components each making independent predictions. However, this division is not semantic-aware: in face of concept drifts, it is still very costly to tell which components are affected and hence must be replaced, and what new components must be brought in to represent the new concept. In this sense, the ensemble model is still monolithic.

In this paper, we are concerned with the problem of reducing model update cost in classifying data streams. We observe that in most real life applications, concepts evolve slowly, which means we shall be able to avoid making global changes to the model all the time. We achieve this by building a model consisting of semantic-aware components of fine granularity. In face of concept drifts, it enables us to figure out, in an efficient way, i) which components are affected by the current concept drift, and ii) what new components shall be introduced to model the new concept without affecting the rest of the model. We show that our approach give accurate predictions at a fraction of cost required by state of the art stream classifiers.

Our Contribution In summary, our paper makes the following contributions:

- We propose the concept of model granularity, and show that granularity plays an important role in determining model update cost.
- We introduce a model consisting of semantic-aware components of fine granularity. It enables us to immediately pinpoint components in the model that become obsolete when concept drifts occur.
- We introduce a low cost method to revise the model. Instead of learning new model components from raw data, we propose a heuristic that derives new components efficiently from a novel synopsis data structure.
- Experiments show that our model update cost is reduced without compromising classification accuracy.

Paper Organization The rest of the paper is organized as follows. Section 2 describes related work. In section 3, we use decision tree as an example to introduce the impact of model granularity. Section 4 describe our approach of building a model of fine granularity in detail. Section 5 reports experimental results, and we conclude in Section 6.

2 Related Work

Traditional classification methods, including, for example, C4.5 and the Bayesian network, are designed for static data. To the same goal, rule-based approaches called associative classification have also been proposed [12, 11]. A rule-based classifier is composed of high quality association rules learned from training data sets using user-specified support and confidence thresholds. Since association rules explore highly confident associations among multiple variables, the rule-based approach overcomes the constraint of the decision-tree induction method which examines one variable at a time. As a result, they usually have higher accuracy than the traditional classification methods. However, the techniques used in these methods focus on mining rules in static data, and do not apply to infinite data streams, nor concept drifts.

In wake of recent interest in data stream applications, several classification algorithms have been introduced for streaming data [10, 16]. The CVFDT [10] is an incremental approach. It refines a decision tree by continuously incorporating new data from the data stream. In order to handle concept drifts, it retires old examples at a predetermined "fixed rate" and discards or re-grows sub-trees. However, because decision trees are unstable structures, even slight changes in the underlying data distribution may trigger substantial changes in the tree, which may severely compromise learning efficiency. The ensemble approach [16], on the other hand, constructs a weighted ensemble of classifiers. Classifiers in the ensemble are learned from data chunks of fixed size. However, no matter whether concept drifts occur or not, it keeps training new classifiers and recomputing weights of existing classifiers in the ensemble, which introduces considerable cost, and becomes vulnerable in dealing high speed data streams.

Our work introduces a rule-based stream classifier. It aims at maintaining a model made up of tiny components that are individually revisable. To access these components efficiently, we use tree based index structures. Using trees as summary structures of data streams has been studied mostly in the field of finding frequent patterns in data stream. Manku *et al.* [13] proposed an approximate algorithm that mines frequent patterns over data in the entire stream up to now. The estDec method [2] finds recent frequent patterns, and it defines frequency using an aging function. The Moment algorithm [4] uses index trees to mine closed frequent patterns in a sliding window. However, although similar data structures are used, they are for different purposes. Their goal is to find all patterns whose occurrence is above a threshold, and the problem of prediction is not considered.

3 Motivation

For streams with concept drifts, model updates are not avoidable. Our task is to design a model that can be updated easily and efficiently. To achieve this goal, we need to ensure the following:

1. The model is decomposable into smaller components, and each component can be revised independently of other components;

2. The decomposition is semantic-aware in the sense that when concept drift occurs, there is

an efficient way to pinpoint which component is obsolete, and what new component shall be built.

Clearly, the incrementally updated classifiers meet neither of the two requirements, while the ensemble classifiers satisfy only the first one. Our motivation is to reduce update cost by reducing model granularity. We first use decision tree classifiers to illustrate the impact of model granularity on update cost. Then, we introduce a rule-based classifier, and discuss the possibility it offers to reduce update cost.

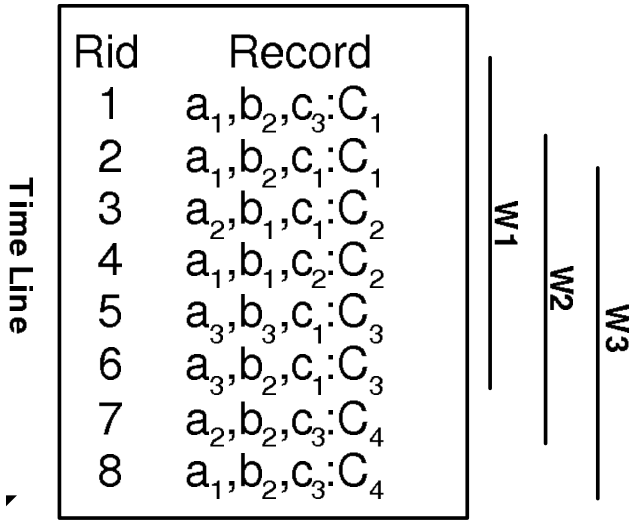
3.1 Monolithic Models

We consider a stream as a sequence of records $r_1, \cdots, r_k, \cdots$, where each record has *d* attributes $A_1, \cdots, A_d$. In addition, each training record is associated with a class label C_i. We place a moving window on the stream. Let W_i denote the window over records $r_i, ..., r_{i+w-1}$, where w is the window size. From window W_i, we learn a model $\mathcal{C}_i$.

We use decision tree to illustrate the cost of model update. The reason of using decision tree is because decision trees are considered "interpretable" models, that is, unlike a black box, their semantics allow them to be updated incrementally. Figure 1 shows a data stream with moving windows of size 6. Each record has 3 attributes and a class label. The decision tree for W_1 is shown in Figure 2(a). After the arrival of record r_7 and r_8, the window moves to W_3. The decision tree of W_3 is shown in Figure 2(b).

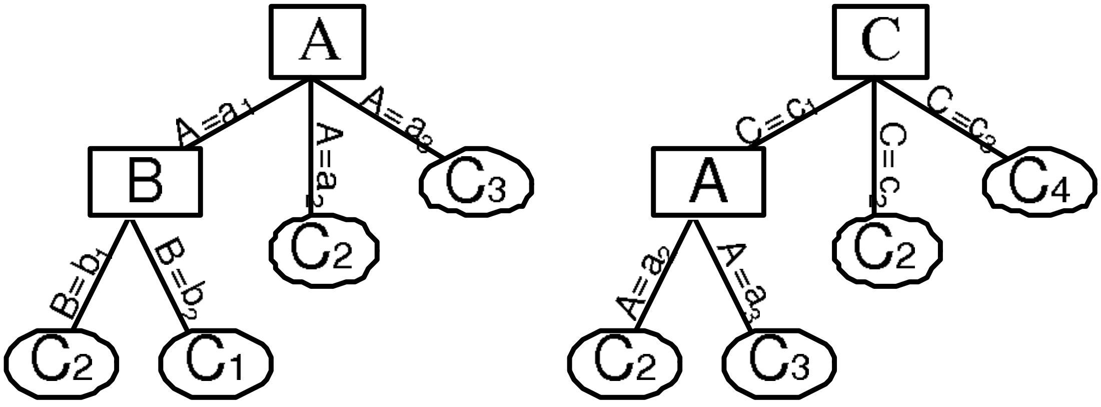
Although majority of the data and the concept they embody keep the same, we find that the two decision trees are completely different. This illustrates that small disturbances in the data stream may cause global changes in the model. Thus, even for an interpretable model, in many cases, incrementally maintaining the model is as costly as rebuilding one from scratch.

The ensemble approach uses multiple models. However, the models are usually homogeneous, each of which tries to capture the global data feature as accurate as possible. In this sense, each model is still monolithic, and it is replaced as a whole when accuracy drops [16].



Rid	Record
1	$a_1,b_2,c_3:C_1$
2	$a_1,b_2,c_1:C_1$
3	$a_2,b_1,c_1:C_2$
4	$a_1,b_1,c_2:C_2$
5	$a_3,b_3,c_1:C_3$
6	$a_3,b_2,c_1:C_3$
7	$a_2,b_2,c_3:C_4$
8	$a_1,b_2,c_3:C_4$

Figure 1: Moving windows on streaming data



(a) W_1

(b) W_3

Figure 2: Decision tree models

3.2 Rule-based Models

A rule has the form of $p_1 \wedge p_2 \wedge ... \wedge p_k \rightarrow C_j$, where C_j is a class label, and each p_i is a predicate in the form of $A_i = v$. We also denote $p_1 \wedge p_2 \wedge ... \wedge p_k$ as a pattern.

We learn rules from records in each window W_i. If a rule's support and confidence are above the predefined threshold *minsup* and *minconf*, we call it a *valid rule*. All valid rules learned from window W_i form a classifier $\mathcal{C}_i$.

We use rules to model the data shown in Figure 1. Assume the *minsup* and *minconf* are set at 0.3 and 0.8 respectively. The valid rules of W_1 are[1]

$$a_1, b_2 \rightarrow C_1, \quad b_1 \rightarrow C_2, \quad a_3, c_1 \rightarrow C_3, \quad a_3 \rightarrow C_3 \tag{1}$$

After the window moves to W_3, the valid rules become

$$c_3, b_2 \rightarrow C_4, \quad b_1 \rightarrow C_2, \quad a_3, c_1 \rightarrow C_3, \quad a_3 \rightarrow C_3 \tag{2}$$

From (1) and (2) above, we make the following observations: i) only the first rule has changed, which shows, indeed, small disturbance in the data does not always introduce overall model update; ii) rule-based models have very low granularity, because each component, whether a rule, a pattern, or a predicate, is interpretable and replaceable.

Thus, our goal is that, as long as the majority of the data remains the same between two windows, we shall be able to slightly change certain components of the model to maintain its uptodateness. In the rest of the paper, we develop algorithms that allow us to i) efficiently pinpoint components that become outdated, and ii) efficiently derive new components to represent emerging concepts.

[1] When no confusion can arise, we use a_1 to denote predicate $A = a_1$.

4 A Low Granularity Stream Classifier

4.1 Overview

We maintain a classifier that consist of a set of rules. Let W_i be the most recent window, and let $\mathcal{C}_i$ be the classifier for W_i. When window W_i moves to W_{i+1}, we update the support and the confidence of the rules in $\mathcal{C}_i$. The new classifier $\mathcal{C}_{i+1}$ contains the old rules of $\mathcal{C}_i$ that are still valid (support and confidence above threshold) as well as new rules we find in W_{i+1}. To classify an unlabelled record, we use the rule that has the highest confidence among all the rules that match the record. If the record does not match any valid rule, we classify it to be the majority class of the current window.

The main technique of our approach lies in its handling of concept drifts.

- If the concept drifts are not too dramatic and the window of the stream has appropriate size[2], most rules do not change their status from valid to invalid or from invalid to valid. In this case, our approach incurs minimal learning cost.

- Our approach detects concept drifts by tracking misclassified records. The tracking is performed by designating a historical window for each class as a reference for accuracy comparison. This allows us to efficiently pinpoint the components in the model that are outdated.

- We introduce a heuristic to derive new model components from the distribution of misclassified records, thus avoiding learning new models from the raw data again.

[2] Windows that are too big will build up conflicting concepts, and windows that are too small will give rise to the overfitting problem. Both have an adverse effect on prediction accuracy.

4.2 Dealing with concept drifts

We describe methods that detect concept drifts, pinpoint obsolete model components, and derive new model components efficiently.

4.2.1 Detecting Concept Drifts

First, we group rules by their class label. Second, for each class, we designate a historical window as its reference window.

To detect concept drifts related to class $\mathcal{C}_i$, we compare the predictive accuracy of rules corresponding to $\mathcal{C}_i$ in the reference window and in the current window. The rationale is that when the data distribution is stable and the window size is appropriate, the classifier has stable predictive accuracy. If at certain point, the accuracy drops considerably, it usually means some concept drifts have occurred and some new rules that represent the new concept may have emerged.

Predictive accuracy of a model is usually measured by the percentage of records that the model misclassified. For instance, in the ensemble approach [16], a previous classifier is considered obsolete if it has a high percentage of misclassified records, in which case, the entire classifier will be discarded. Clearly, this approach does not tell us *which part* of the classifier gives rise to the inaccuracy so that only this particular part needs to be updated.

In our approach, instead of using the percentage, we study the *distribution* of the misclassified records. More formally, we say a record is misclassified if i) the classifier assigns a wrong label to the record, or ii) there is no matching rule for the record. Furthermore, we say record r is a misclassified record that belong to C_j if r's true label is C_j.

Definition 1. *Number of misclassified records:* N_{ij}

Let W_i be a window. N_{ij} is the number of records in W_i whose true class is C_j but are misclassified by the last classifier.

Theorem 2. *If the stream is stable, N_{ij} keeps stable with a high probability.*

Proof: Let R: $P \to C$ is a valid rule in a window. The true support(stable support) of P is s and the observed support in this window is s'. Similarly, the true confidence of R is c and the observed confidence is c'. We regard s' and c' as random variables. Chernoff bound states, for any $0 \leq \epsilon \leq 1$:

$$Pr\{|s' - s| \geq \epsilon\} \leq 2e^{\frac{-w\epsilon^2}{2s}} \tag{3}$$

Let the right side of Equation be δ. We see that, with probability $\geq \delta$, the difference of s' and s is beyond ϵ, when $w = \frac{2sln(2/\delta)}{\epsilon^2}$. Let $s_0 = max\{sup(P)|P$ is the pattern of a valid rule$\}$. We can get when $w \geq \frac{2s_0 ln(2/\delta)}{\epsilon^2}$, the observed support of any pattern of a valid rule will be within $\pm\epsilon$ of its true support with probability $1 - \delta$.

Similarly, we have:

$$Pr\{|c' - c| \geq \epsilon'\} \leq 2e^{\frac{-w\epsilon'^2}{2c}} \tag{4}$$

Let $c_0 = max\{conf(R)|R$ is a valid rule$\}$.Similarly, we can get when $w \geq \frac{2c_0 ln(2/\delta)}{\epsilon'^2}$, the observed confidence of any valid rule will be within $\pm\epsilon'$ of its true confidence with probability $1 - \delta$.

From above we can get that if stream is stable, the support and the confidence of valid rules will be stable. With these rules we can get a stable classifier named standard classifier. Although the support and the confidence may fluctuate slightly in different window, the fluctuation will not affect the prediction results. That is, for a record, the prediction result by the classifier in certain window will be same with that of the standard classifier. Further, we can split the data space into two parts: P_r and P_w where P_r contains the records that can be classified correctly and P_w contains the records that cannot be classified correctly by the standard classifier. Using chernoff bound again, and taking whether a record belongs to P_r as a random variable, we can state that the number of misclassified records will stable with a high probability. So, when the stream is stable, N_{ij} will be stable with a high probability. End.

With this theorem, we can get that when N_{ij} increases dramatically, a concept drift related to C_j happens in a high probability. Moreover, The misclassified records can enable us to pinpoint the exact subset of rules that are conflicting with the new data distribution, and further more, through a careful analysis of them, we can derive the emerging new rules.

To measure whether an increase of N_{ij} amounts to a concept drift in window W_i, we choose a historical window W_k as a reference. Note that this reference window cannot be a fixed window. For example, if we always use the window immediately before W_i (that is, $k = i - 1$) as the reference window, we may not be able to detect any concept drift, because concept drifts usually build up slowly, and only become apparent over a certain period of time.

Definition 3. *Reference Window*

Let W_i be the current window. We say window W_k is the reference window for class C_j if $N_{kj} = \min_{l \le i} N_{lj}$.

Clearly, for different classes, the reference windows may be different. The reference window enables us to tell how far the concepts (with regard to a particular class) have drifted away from the state in which they are accurately modelled. With this knowledge, we can decide whether we need to mine new rules that model this concept. Formally, if the difference between N_{ij} and N_{kj} reaches a user-defined threshold `minWR`, i.e., $N_{ij} - N_{kj} \ge \texttt{minWR}$, it may indicate that we need new rules for class C_j in W_i.

N_{ij} is computed by the following equation:

$$N_{ij} = N_{i-1,j} + g(r_{i+w-1}, i, j) - g(r_{i-1}, i, j) \qquad (5)$$

where $g(r, i, j) = 1$ if r's true label is C_j and is misclassified by $\mathcal{C}_i$ to some other class, and 0 otherwise.

4.2.2 Finding new rules

Assume in window W_i, we find $N_{ij} - N_{kj} \geq \texttt{minWR}$. We need to find new rules to deal with the drop of the accuracy. To avoid learning the new rules from scratch, we analyze the misclassified records being tracked to find clues about the patterns of the new rules.

Assume all misclassified records whose true class label is C_j satisfy two predicates $A_1 = v_1$ and $A_2 = v_2$. Then, it is very likely that a new rule in the form of $P \rightarrow C_j$ has emerged where P contains one or both of the two predicates. On the other hand, if a predicate is satisfied by few misclassified records, probably the new rules do not contain the predicate. We use this heuristics to estimate the form of the new rules based on the information in the misclassified records.

Formally, we use L_{ij} to denote the set of predicates each of which is satisfied by no less than c misclassified records that belong to class C_j. We represent L_{ij} in the form of $\{p_i : c_i\}$ where p_i is a predicate, and $c_i \geq c$ is the number of misclassified records belonging to C_j that satisfy p_i. We use L_{ij} to generate candidate patterns of the new rules.

Let us return to the data stream in Figure 1 as an example. Assume minWR is 2. For window W_1, classifier $\mathcal{C}_1$ (shown in Eq 1) classifies every record correctly, so we have $N_{1,i} = 0$ for $1 \leq i \leq 4$. For window W_3, both r_7 and r_8 are misclassified, so $N_{3,4}$ becomes 2. Since the increase of $N_{3,4}$ is

$\geq$ `minWR`, we decide to mine new rules in W_3. For class C_4, the misclassified predicates and their misclassified frequency are: $\{b_2 : 2, c_3 : 2, a_1 : 1, a_2 : 1\}$. We then decide the new rule is very likely to have a pattern that includes predicate b_2 and/or predicate c_3, and we use these predicates to generate the pattern of the new rule, and ignore other patterns. It turns out that $c_3, b_2 \rightarrow C_4$ is exactly the new rule we are looking for (Eq 2).

We describe our method of mining new rules more formally. We use a table *T* to store L_{ij} for each class C_j. Table *T* is updated as the window moves so it always contain the misclassified predicates and their frequencies in the most recent window. We update T as follows. When W_i becomes the new window and record r_{i-1} (the record that moves out) is a previously misclassified record whose true label is C_j, then for each attribute A_d, we decrease the count of $A_d = v_d$ by 1, where v_d is the value of r_{i-1} for attribute A_d. We do the same for r_{i+w-1} (the record that moves in), but increase the count instead of deceasing it.

```
Algorithm MINERULE
1.   sort all candidate predicates by their frequency
2.   choose the top-K predicates $p_1, ..., p_K$ to construct a Candidate Rule Set (CRS)
3.   scan the window to compute the support and confidence of rules in CRS
4.   add valid rules to the current classifier
```

For every w records (w is the window size), we compare N_{ij} with N_{kj} for each j. We invoke procedure MINERULE to mine new rules if the difference exceeds `minWR`. First, we construct a set of candidate patterns. We sort all predicates by their occurrence frequencies in descending order. Then, we use the top-K predicates to construct the patterns. We restrict the patterns to be within certain length N. The reason is: i) a rule with many predicates has low support and cannot form a valid rule, ii) complex rules tent to overfit the data, and iii) evaluating rules with a long pattern is time consuming. Then, we construct a candidate rule set (CRS) for class C_j using patterns we have obtained. We compute the support and the confidence of the rules in CRS. If a certain candidate rule is valid, that is, its support and confidence exceeds $minsup$ and $minconf$, we add this new rule into the current classifier.

One remaining problem is, are the new rules discovered by procedure MINERULE enough to represent the latest data distribution? After all, the constraints we use might have prevented us from discovering some subtle concepts. Indeed, if the constraints are relaxed, we may find more rules, but some just reflect noises in the data. The new rules will be evaluated as the window moves ahead. At some window $W_{i'}$ down the stream when we are required to compare $N_{i'j}$ and N_{kj}, we will have accumulated more statistics. In most cases, the introduction of the new rules in window W_i will have reduced $N_{i'j}$ so that its difference with N_{kj} is smaller than `minWR`, which means the new rules are sufficient. In case the difference between $N_{i'j}$ and N_{kj} is still larger than `minWR`, which means either we have missed some subtle rules, or there is another concept drift, we invoke procedure MINERULE again. The experiments show that in most cases, we only need to apply procedure MINERULE once to get enough new rules for one concept drift.

4.3 The algorithm

To build a stream classifier that is efficiently adaptable to new concepts, one major issue is to how to access the records and update the rules efficiently. In this section, we describe data structures and algorithms for this purpose.

4.3.1 Data Structure

We use two tree structures to maintain the rules and the records for the most recent window.

The RS-tree We assume there is a total order among attributes $A_1 \prec \cdots \prec A_d$. We can sort predicates and patterns based on this order. We store current rules in a prefix tree called the RS-tree. Each node N represents a unique rule $R : P \rightarrow C_i$. A node N' that represents rule $P' \rightarrow C_j$ is a child node of N, iff:

1. $P \subset P'$
2. $P \prec P'$
3. no other rule $P'' \rightarrow C_k$ exists so that $P \subset P'' \subset P'$ and $P \prec P'' \prec P'$

A node stores $sup(R)$ and $conf(R)$ for the rule R it represents. An example RS-tree is shown in Figure 3(b). Node N_1 represents rule $(a_1, b_2) \rightarrow C_1$ whose support and confidence are 0.33 and 1 respectively. Node N_3 is the child of N_2 since $\{a_3\} \subset \{a_3, c_1\}$ and $\{a_3\} \prec \{a_3, c_1\}$.

The REC-tree We think of each record r as a sequence, $\langle v_d, \cdots, v_1, C\rangle$, where v_i is r's value for attribute A_i, and C is r's class label. We insert record r in its sequence representation $\langle v_d, \cdots, v_1, C_i\rangle$ into a tree structure, which we call the REC-tree. A path from any internal node N to the root node represents a unique postfix $\{A_i = v_i, A_{i+1} = v_{i+1}, \cdots, A_d = v_d\}$.

Each internal node keeps a counter, which denotes how many records of the current window contain the postfix represented by the node. A node in the REC-tree may point to nodes in the RS-tree. Assume $p_1 \prec \cdots \prec p_k$. Node N points to rule $p_1 \wedge p_2 \wedge \cdots \wedge p_k \rightarrow C_i$ in the RS-tree if:

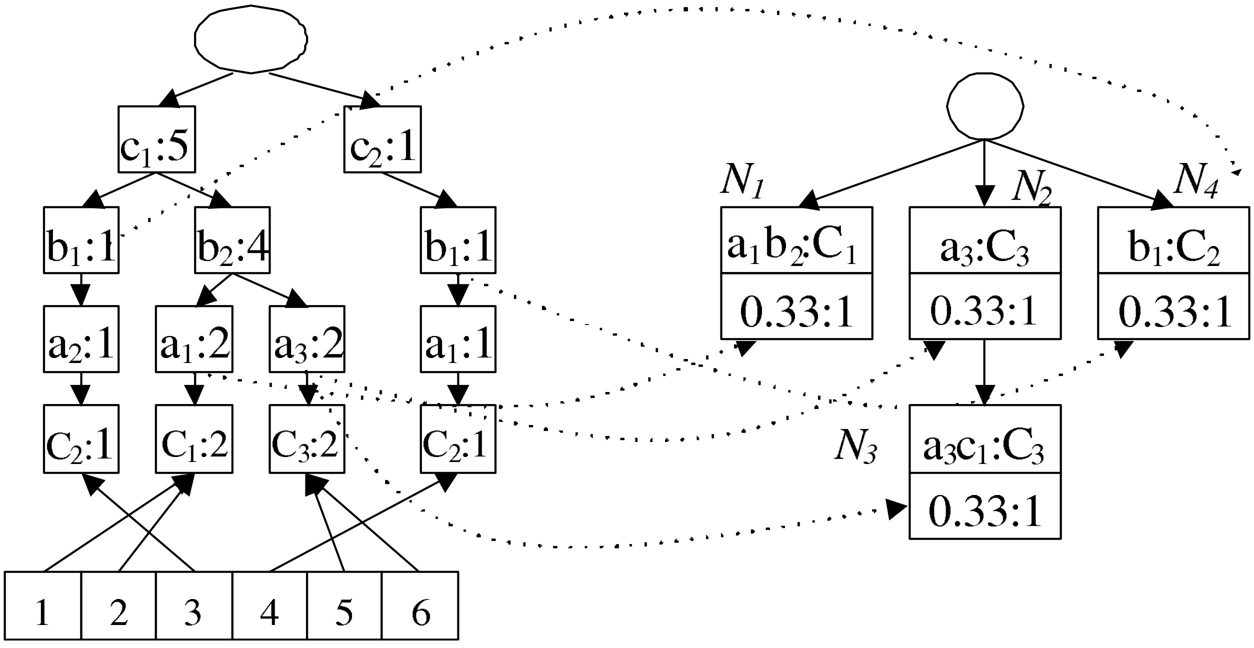
1. node N satisfies p_1

2. the postfix that starts at N contains the pattern $p_1 \wedge p_2 \wedge \cdots \wedge p_k$

Intuitively, node N represents a projection of a record r, and it points to all rules whose pattern r satisfies. For each record that moves into the window, we update the support and the confidence of the rules it matches. The rule pointers speed up this process.

An example REC-tree is shown in Figure 3(a). Record $\{a_2, b_1, c_1 : C_1\}$ is stored on the left-most path. Node $(b_1 : 1)$ in the path points to rule $b_1 \rightarrow C_2$ in the RS-tree.

The REC-tree is associated with an array of record ids $[i, \cdots, i+w-1]$. Each record id points to a leaf node that represents that record. When a new record arrives, we insert it into REC-tree and also insert an entry in the rid array. The record id array enables us to access any record in the window efficiently.



(a) REC-tree (b) RS-tree

Figure 3: REC-tree and RS-tree

4.3.2 Update the classifier using REC-tree

Assume the current window is W_i. When a new record r_{i+w} arrives, the window becomes W_{i+1} and we derive a new classifier $\mathcal{C}_{i+1}$. First we insert r_{i+w} to the REC-tree. We update the support and the confidence of the rules pointed to by the nodes involved in the insertion as follows.

$$\begin{aligned} sup_{i+1}(R) &= \frac{sup_i(R) * w + 1}{w} \\ conf_{i+1}(R) &= \begin{cases} \frac{conf_i(R) * sup_i(R) + 1}{sup_i(R) + 1} & : \quad C_i = C_j \\ \frac{conf_i(R) * sup_i(R)}{sup_i(R) + 1} & : \quad C_i \neq C_j \end{cases} \end{aligned}$$

where $sup_i(R)$ and $conf_i(R)$ are the old support and the old confidence of R, and $sup_{i+1}(R)$ and $conf_{i+1}(R)$ are the new ones; C_i is the class label of r_{i+w} and C_j is the class label of R.

The insertion of r_{i+w} can create new nodes, in which case the counter is set to 1. Moreover, new rule pointers, if necessary, are added to this node. To find which rules are matched a postfix, we need to scan RS-tree. Assume a new node represents $A_i = v$. Since the RS-tree is a prefix tree, we only need to scan the subtrees whose root's rule has $A_i = v$ as its pattern's first predicate.

We delete r_i from REC-tree and update the rules matched by it. We decrease the counters of the nodes involved. When the counter of a node becomes 0, we do not delete it from the REC-tree immediately. Since it contains the information of the rules it points, the information can be used later when a record with the same postfix arrives. However, when the number of nodes in REC-tree exceeds a threshold, we delete the nodes whose counters are 0.

4.3.3 The main algorithm

We now describe our algorithm as a whole. It contains two phases. The first phase is the initial phase. We use the first w records to train all valid rules for window W_1. Based on them, we construct the RS-tree and the REC-tree. The second phase is the update phase. When record r_{i+w} arrives, we insert it into the REC-tree and update the support and the confidence of the rules matched by it. Then we delete the oldest record and also update the rules matched according to it. For every w records, we compare $N_{i+1,j}$ and N_{kj} for each class label. If for some j, their difference exceeds `minWR`, we apply procedure MINERULE to find the new rules. Algorithm UPDATE describes the update phase.

Algorithm UPDATE

```
Input: $r_i$: record that moves out of the window;
Input: $r_{i+w}$: record that moves into the window;
let $N$ be the node that represents $r_i$ in the REC-tree;
for each node $n$ from $N$ to the root node
    decrement $n$'s counter by 1;
    update the rules pointed by $n$;
for $m \leftarrow d$ to $1$
    if $A_m = v_m$ already exists in REC-tree
      then increment its counter by 1;
           update the rules' support and the confidence;
      else create a new node with counter =1;
           scan RS-tree and add rule pointers if necessary;
add a new entry in record id array;
update $N_{i+1,j}$ and $L_{i+1,j}$;
if $(i+1)$ mod $w = 0$ and $(N_{i+1,j} - N_{kj}) \geq$ minWR
```

```
14.    then apply MINERULE;
```

5 Experiments

We conducted extensive experiments on both synthetic and real life data streams. The results show the low granularity classifier proposed in this paper has significant advantage over state of the art approaches. Tests are conducted on a PC with a 1.7GHz CPU and 256 MB main memory.

Datasets We create synthetic data with drifting concepts using a moving hyperplane. A hyperplane in d-dimensional space is represented by: $\sum_{i=1}^{d} a_i x_i = a_0$. Records satisfying $\sum_{i=1}^{d} a_i x_i < a_0$ are labelled positive, otherwise negative. Hyperplane has been used to simulate time-changing concepts because the orientation and the position of the hyperplane can be changed in a smooth manner by changing the magnitude of the weights [10, 16].

We generate random records uniformly distributed in $[0, 1]^d$. Weights $a_i (1 \leq i \leq d)$ are initialized by random values in [0, 1]. We set $a_0 = \frac{1}{2} \sum_{i=1}^{d} a_i$ so that the hyperplane cuts the multi-dimensional space in two parts of the same volume. Thus, roughly half of the examples are positive, and the other half are negative.

We simulate concept drifts using several parameters. Parameter k specifies the number of dimensions whose weights are changing. Parameter $t_i \in \mathcal{R}, 1 \leq i \leq k$ specifies the magnitude of the change for weights a_i, and $s_i \in \{-1, 1\}$ specifies the direction of change for each weight a_i.

We also used real life dataset ‘nursery’ in the UCI ML Repository [1]. The dataset has 10,344 records and 8 dimensions. we randomly sample records from the dataset to generate a stream. To simulate concept drifts, for every 50,000 records sampled, we randomly select some attributes of the data set and change their values in a consistent way. One method we use is to shuffle the values, for instance, we change values $a_1 \rightarrow a_2 \rightarrow \cdots \rightarrow a_n \rightarrow a_1$ for all records and keep the class label intact.

5.1 Effect of model updating

We show the effectiveness of updating a rule-based classifier model in handling concept drifts. We compare three approaches: i) training an initial rule-based classifier and using it without change to classifier streaming data, ii) continuously revising the initial classifier by updating the support and confidence of its rules, and iii) continuously revising the initial classifier by updating the confidence/support of the existing rules and discovering new rules.

In the synthetic dataset we use, each record has 10 dimensions. The window size is 5,000. We introduce a concept drift by randomly choosing 4 dimensions and changing their weights for every 50,000 records. The results in Figure 4 are obtained for a run with parameters $minsup = .03$, $minconf = .7$, $\texttt{minWR} = 150$, $K = 7$ (the top-K parameter) and $N = 5$ (the maximum pattern length).

From Figure 4, we observe that, 1) after the first concept drift, the accuracy of the initial classifier drops significantly, which shows that concept drifts can be effectively detected by misclassified records, 2) updating rule can adjust the classifier to adapt to the concept drift. 3) compared to adjusting existing rules only, the classifier that mines new rules has higher accuracy.

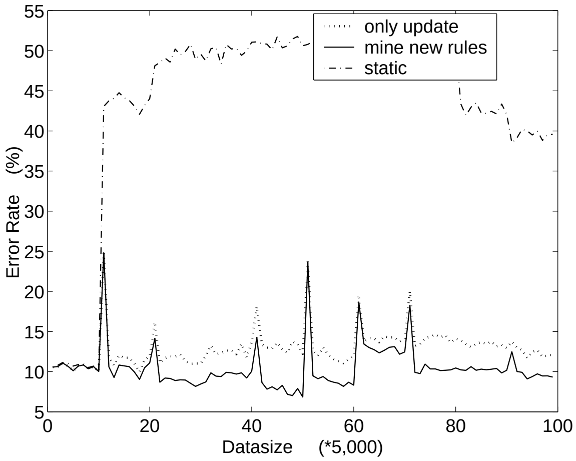


Figure 4: Error Rate Comparison

5.2 The relation of concept drifts and N_{ij}

In this subsection, we verify the approach to detect concept drifts using abnormally increase of N_{ij}. We test it in a hyperplane dataset. Every after 50,000 records, we adjust the hyperplane to simulate a concept drift. These two curves indicate N_{i1} and N_{i0} respectively. The result is shown in figure 5. We can see that each time a concept drift happen, N_{i1} and N_{i0} burst dramatically while they keep stable when the streaming data is stable.

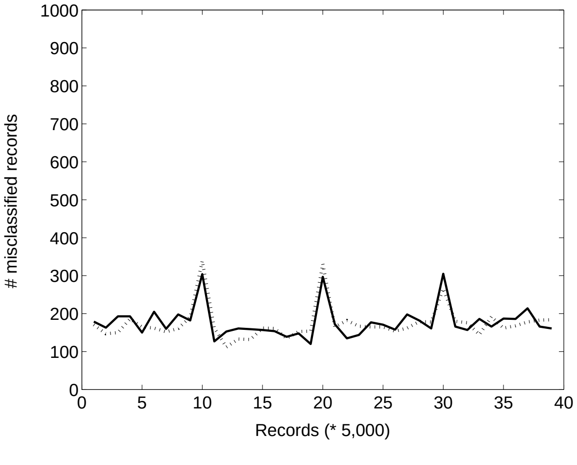


Figure 5: Relation of concept drifts and N_{ij}

5.3 Effect of rule composition

We verify the effectiveness of choosing the top-K candidate predicates in composing new rules. We compare its accuracy against choosing predicates randomly. We used the same parameters as in the previous experiment. The result is shown in Figure 6. It shows that by using most frequently occurring predicates in misclassified records to construct CRS, we can obtain rules that effectively representing the new concept.

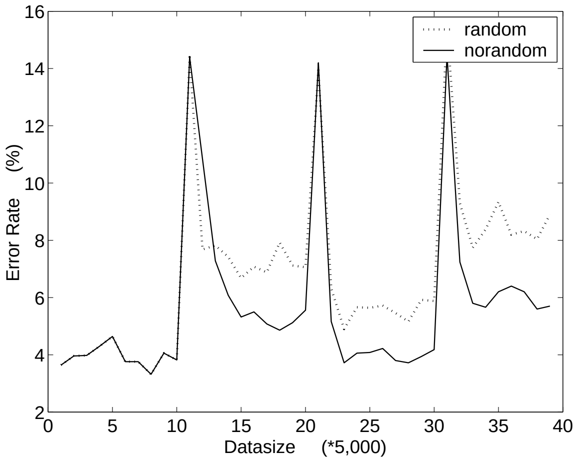
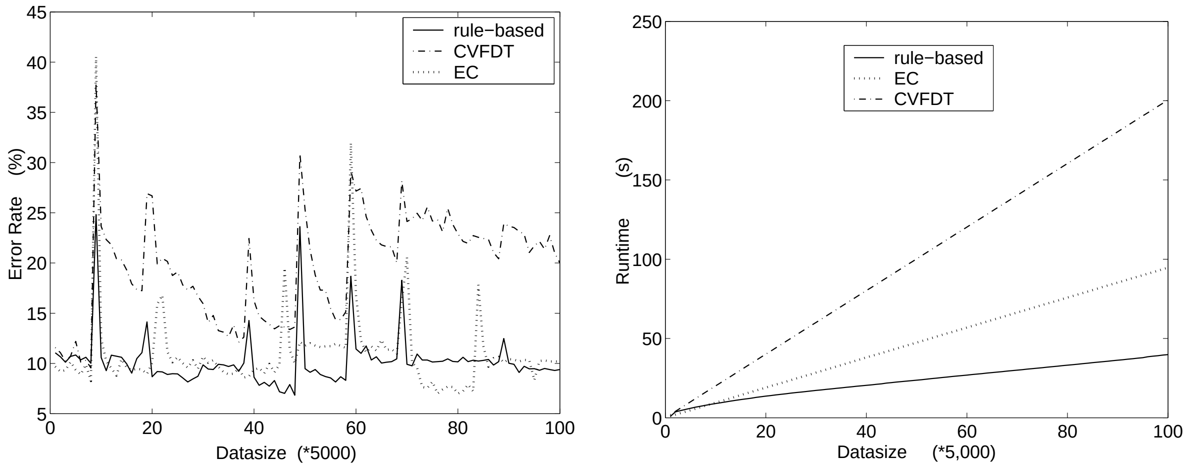


Figure 6: Choosing literals

5.4 Accuracy and time

While having improved or similar classifying accuracy, our rule-based approach is much faster than state of the art stream classifiers. We compare our method with the ensemble classifier [16] and CVFDT [10] in terms of accuracy and run time on both synthetic data and real life data.

We used different parameters (e.g., window size, number of classifiers in the ensemble, etc.) to tune the classifiers, and a typical result is shown in Figure 7, which is obtained using windows of size 10,000, and the ensemble contains 10 classifiers, each trained on 1,000 records. We report



(a) accuracy

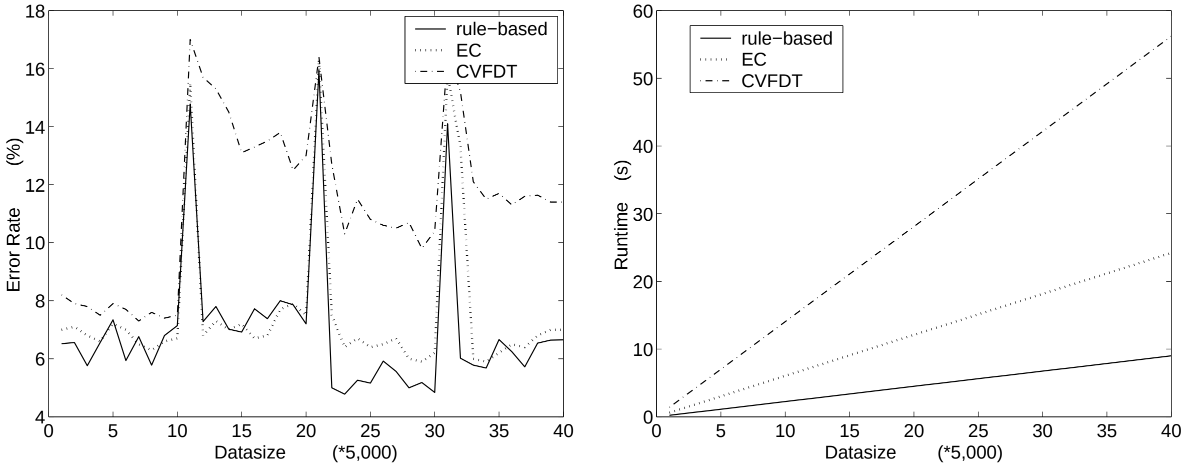
(b) time

Figure 7: Synthetic data

error rate for every 5,000 records.

Figure 7(a) shows that the accuracy of our rule-based approach is higher than that of CVFDT, and is similar to the ensemble classifier. Compared with CVFDT, our rule-based approach can catch concept drifts and adjust the classifier more quickly. This is because the rule-based classifier has low granularity, which means it can quickly determine which component to revise and quickly figure out how to revise it, while the CVFDT has to learn the new concepts by re-growing a decision tree level by level.

Figure 7(b) shows that our rule-based approach is much faster than the ensemble classifier and CVFDT. Unlike the ensemble classifier and CVFDT, which keep learning new models, most of the time all that our approach requires is to adjust the support and confidence of matched rules in the model. Since each record only matches a limited number of rules, this operation is made very efficient using the RS-tree and the REC-tree structure. Even when the new rule detection procedure is triggered, the cost is still very small, because the update has already been limited to a very small space, which is embodied by the top-K predicates. The second experiment is run on the real life



(a) accuracy

(b) time

Figure 8: Real life data

‘nursery’ dataset, and the results shown in Figure 8 are consistent with those on the synthetic data.

6 Conclusion

An important task in mining data stream is to overcome the negative effects of concept drifts. Concept drifts pose a major challenge to stream classification algorithms because of the high cost associated with maintaining the uptodateness of the models. Current stream classifiers are adapted from algorithms designed for static data, and they are hardly incrementally maintainable because it is not easy, if not impossible, to semantically break down the model into smaller pieces. In this paper, we addressed the issue of classifier granularity, and we showed that by reducing this granularity, change detection and model update can be made much more efficient without compromising classification accuracy, as reported by our extensive experiments. Our future study will focus extending this work to enable the reuse of previously learned models by detecting recurring patterns in concept drifts, thus further reducing the model update cost.

References

[1] C. Blake and C. Merz. Uci repository of machine learning databases. In *Univ. of California, Dept. of Information and Computer Science*, 1998.

[2] J. H. Chang and W. S. Lee. Finding recent frequent itemsets adaptively over online data streams. In *SIGKDD*, 2003.

[3] Y. Chen, G. Dong, J. Han, B. W. Wah, and J. Wang. Multi-dimensional regression analysis of time-series data streams. In *VLDB*, Hongkong, China, 2002.

[4] Yun Chi, Haixun Wang, Philip S. Yu, and Richard R. Muntz. Moment: Maintaining closed frequent itemsets over a stream sliding window data streams. In *ICDM*, 2004.

[5] P. Domingos and G. Hulten. Mining high-speed data streams. In *SIGKDD*, pages 71–80, Boston, MA, 2000. ACM Press.

[6] Wei Fan. Systematic data selection to mine concept-drifting data streams. In *SIGKDD*, 2004.

[7] J. Gehrke, V. Ganti, R. Ramakrishnan, and W. Loh. BOAT– optimistic decision tree construction. In *SIGMOD*, 1999.

[8] J. Gehrke, R. Ramakrishnan, and V. Ganti. RainForest: A framework for fast decision tree construction of large datasets. In *VLDB*, 1998.

[9] S. Guha, N. Milshra, R. Motwani, and L. O'Callaghan. Clustering data streams. In *FOCS*, pages 359–366, 2000.

[10] G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *SIGKDD*, pages 97–106, San Francisco, CA, 2001. ACM Press.

[11] Wenmin Li, Jiawei Han, and Jian Pei. CMAR: Accurate and efficient classification based on multiple class-association rules. In *ICDM*, 2001.

[12] Bing Liu, Wynne Hsu, and Yiming Ma. Integrating classification and association rule mining. In *SIGKDD*, 1998.

[13] G. Manku and R. Motwani. Approximate frequency counts over data streams. In *VLDB*, 2002.

[14] C. Shafer, R. Agrawal, and M. Mehta. Sprint: A scalable parallel classifier for data mining. In *VLDB*, 1996.

[15] W. Nick Street and YongSeog Kim. A streaming ensemble algorithm (SEA) for large-scale classification. In *SIGKDD*, 2001.

[16] Haixun Wang, Wei Fan, Philip S. Yu, and Jiawei Han. Mining concept-drifting data streams using ensemble classifiers. In *SIGKDD*, 2003.