Mining Concept-Drifting Data Streams using Ensemble Classifiers

Haixun Wang Wei Fan Philip S. Yu [1]Jiawei Han
IBM T. J. Watson Research, Hawthorne, NY 10532, {haixun,weifan,psyu}@us.ibm.com
[1]Dept. of Computer Science, Univ. of Illinois, Urbana, IL 61801, hanj@cs.uiuc.edu

ABSTRACT

Recently, mining data streams with concept drifts for actionable insights has become an important and challenging task for a wide range of applications including credit card fraud protection, target marketing, network intrusion detection, etc. Conventional knowledge discovery tools are facing two challenges, the overwhelming volume of the streaming data, and the concept drifts. In this paper, we propose a general framework for mining concept-drifting data streams using weighted ensemble classifiers. We train an ensemble of classification models, such as C4.5, RIPPER, naive Bayesian, etc., from sequential chunks of the data stream. The classifiers in the ensemble are judiciously weighted based on their expected classification accuracy on the test data under the time-evolving environment. Thus, the ensemble approach improves both the efficiency in learning the model and the accuracy in performing classification. Our empirical study shows that the proposed methods have substantial advantage over single-classifier approaches in prediction accuracy, and the ensemble framework is effective for a variety of classification models.

1. INTRODUCTION

The scalability of data mining methods is constantly being challenged by real-time production systems that generate tremendous amount of data at unprecedented rates. Examples of such data streams include network event logs, telephone call records, credit card transactional flows, sensoring and surveillance video streams, etc. Other than the huge data volume, streaming data are also characterized by their drifting concepts. In other words, the underlying data generating mechanism, or the concept that we try to learn from the data, is constantly evolving. Knowledge discovery on streaming data is a research topic of growing interest [1, 5, 7, 21]. The fundamental problem we need to solve is the following: given an infinite amount of continuous measurements, how do we model them in order to capture time-evolving trends and patterns in the stream, and make time-critical predictions?

Huge data volume and drifting concepts are not unfamiliar to the data mining community. One of the goals of traditional data mining algorithms is to mine models from large databases with bounded-memory. This goal has been achieved by a number of classification methods, including Sprint [25], RainForest [16], BOAT [15], etc. Nevertheless, the fact that these algorithms require multiple scans of the training data makes them inappropriate in the streaming data environment where the examples are coming in at a higher rate than they can be repeatedly analyzed.

Incremental or online data mining methods [29, 15] are another option for mining data streams. These methods continuously revise and refine a model by incorporating new data as they arrive. However, in order to guarantee that the model trained incrementally is identical to the model trained in the batch mode, most online algorithms rely on a costly model updating procedure, which sometimes makes the learning even slower than it is in batch mode. Recently, an efficient incremental decision tree algorithm called VFDT is introduced by Domingos et al [7]. For streams made up of discrete type of data, Hoeffding bounds guarantee that the output model of VFDT is asymptotically nearly identical to that of a batch learner.

The above mentioned algorithms, including incremental and online methods such as VFDT, all produce a single model that represents the entire data stream. It suffers in prediction accuracy in the presence of concept drifts. This is because the streaming data are not generated by a stationary stochastic process, indeed, the future examples we need to classify may have a very different distribution from the historical data.

In order to make time-critical predictions, the model learned from the streaming data must be able to capture up-to-date trends and transient patterns in the stream. To do this, as we revise the model by incorporating new examples, we must also eliminate the effects of examples representing outdated concepts. This is a non-trivial task. The challenge of maintaining an accurate and up-to-date classifier for infinite data streams with concept drifts including the following:

- ACCURACY. It is difficult to decide what are the examples that represent outdated concepts, and hence their effects should be excluded from the current model. A commonly used approach is to 'forget' old examples at a constant rate. However, a higher rate would lower the accuracy of the 'up-to-date' model as it is supported by a less amount of training data and a lower rate would make the model less sensitive to the current trend and prevent it from discovering transient patterns.

- EFFICIENCY. Decision trees are constructed in a greedy divide-and-conquer manner, and they are non-stable. Even a slight drift of the underlying concepts may trigger substantial changes (e.g., replacing old branches with new branches, re-growing or building alternative subbranches) in the tree, and severely compromise learning efficiency.

- EASE OF USE. Substantial implementation efforts are required to adapt classification methods such as decision trees to handle data streams with drifting concepts in an incremental manner [21]. The usability of this approach is limited as state-of-the-art learning methods cannot be applied directly.

In light of these challenges, we propose using *weighted classifier ensembles* to mine streaming data with concept drifts. Instead of continuously revising a single model, we train an ensemble of classifiers from sequential data chunks in the stream. Maintaining a most up-to-date classifier is not necessarily the ideal choice, because potentially valuable information may be wasted by discarding results of previously-trained less-accurate classifiers. We show that, in order to avoid overfitting and the problems of conflicting concepts, the expiration of old data must rely on data's distribution instead of only their arrival time. The ensemble approach offers this capability by giving each classifier a weight based on its expected prediction accuracy on the current test examples. Another benefit of the ensemble approach is its efficiency and ease-of-use. In this paper, we also consider issues in cost-sensitive learning, and present an *instance-based ensemble pruning* method that shows in a cost-sensitive scenario a pruned ensemble delivers the same level of benefits as the entire set of classifiers.

Paper Organization. The rest of the paper is organized as follows. In Section 2 we discuss the data expiration problem in mining concept-drifting data streams. In Section 3, we prove the error reduction property of the classifier ensemble in the presence of concept drifts. Section 4 outlines an algorithm framework for solving the problem. In Section 5, we present a method that allows us to greatly reduce the number of classifiers in an ensemble with little loss. Experiments and related work are shown in Section 6 and Section 7.

2. THE DATA EXPIRATION PROBLEM

The fundamental problem in learning drifting concepts is how to identify in a timely manner those data in the training set that are no longer consistent with the current concepts. These data must be discarded. A straightforward solution, which is used in many current approaches, discards data indiscriminately after they become old, that is, after a fixed period of time T has passed since their arrival. Although this solution is conceptually simple, it tends to complicate the logic of the learning algorithm. More importantly, it creates the following dilemma which makes it vulnerable to unpredictable conceptual changes in the data: if T is large, the training set is likely to contain outdated concepts, which reduces classification accuracy; if T is small, the training set may not have enough data, and as a result, the learned model will likely carry a large variance due to overfitting.

We use a simple example to illustrate the problem. Assume a stream of 2-dimensional data is partitioned into sequential chunks based on their arrival time. Let S_i be the data that came in between time t_i and t_{i+1}. Figure 1 shows the distribution of the data and the optimum decision boundary during each time interval.

The problem is: after the arrival of S_2 at time t_3, what part of the training data should still remain influential in the current model so that the data arriving right after t_3 can be most accurately classified?

On one hand, in order to reduce the influence of old data that *may* represent a different concept, we shall use nothing but the most recent data in the stream as the training set. For instance, use the training set consisting of S_2 only (i.e., $T = t_3 - t_2$, data S_1, S_0 are discarded). However, as shown in Figure 1(c), the learned model

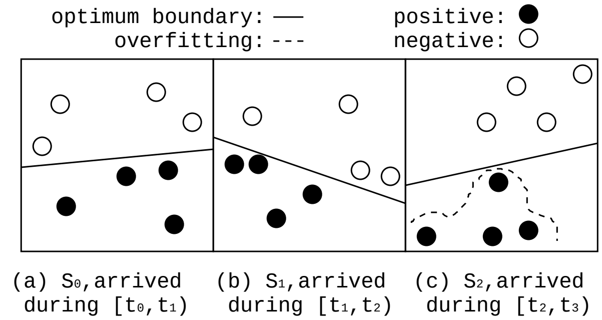


Figure 1: data distributions and optimum boundaries

may carry a significant variance since S_2's insufficient amount of data are very likely to be overfitted.

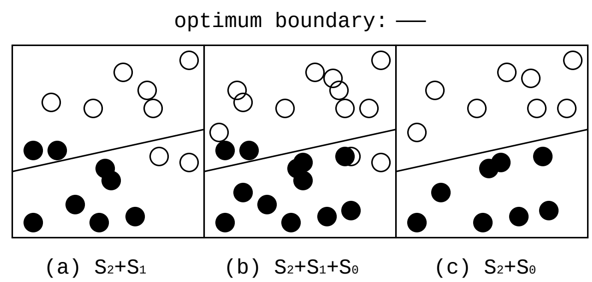


Figure 2: Which training dataset to use?

The inclusion of more historical data in training, on the other hand, may also reduce classification accuracy. In Figure 2(a), where $\mathsf{S}_2 \cup \mathsf{S}_1$ (i.e., $T = t_3 - t_1$) is used as the training set, we can see that the discrepancy between the underlying concepts of S_1 and S_2 becomes the cause of the problem. Using a training set consisting of $\mathsf{S}_2 \cup \mathsf{S}_1 \cup \mathsf{S}_0$ (i.e., $T = t_3 - t_0$) will not solve the problem either. Thus, there may not exists an optimum T to avoid problems arising from overfitting and conflicting concepts.

We should not discard data that may still provide useful information to classify the current test examples. Figure 2(c) shows that the combination of S_2 and S_0 creates a classifier with less overfitting or conflicting-concept concerns. The reason is that S_2 and S_0 have similar class distribution. Thus, instead of discarding data using the criteria based solely on their arrival time, we shall make decisions based on their class distribution. Historical data whose class distributions are similar to that of current data can reduce the variance of the current model and increase classification accuracy.

However, it is a non-trivial task to select training examples based on their class distribution. In this paper, we show that a weighted classifier ensemble enables us to achieve this goal. We first prove, in the next section, a carefully weighted classifier ensemble built on a set of data partitions $\mathsf{S}_1, \mathsf{S}_2, \cdots, \mathsf{S}_n$ is more accurate than a single classifier built on $\mathsf{S}_1 \cup \mathsf{S}_2 \cup \cdots \cup \mathsf{S}_n$. Then, we discuss how the classifiers are weighted.

3. ERROR REDUCTION ANALYSIS

Given a test example y, a classifier outputs $f_c(y)$, the probability of y being an instance of class c. A classifier ensemble pools the outputs of several classifiers before a decision is made. The most popular way of combining multiple classifiers is via averaging [28], in which case the probability output of the ensemble is given by:

$$f_c^E(y) = \frac{1}{k} \sum_{i=1}^{k} f_c^i(y)$$

where $f_c^i(y)$ is the probability output of the i-th classifier in the ensemble.

The outputs of a well trained classifier are expected to approximate the *a posterior* class distribution. In addition to the Bayes error, the remaining error of the classifier can be decomposed into two parts, bias and variance [17, 22, 8]. More specifically, given a test example y, the probability output of classifier $\mathcal{C}_i$ can be expressed as:

$$f_c^i(y) = p(c|y) + \underbrace{\beta_c^i + \eta_c^i(y)}_{\text{added error for } y} \tag{1}$$

where $p(c|y)$ is the *a posterior* probability distribution of class c given input y, β_c^i is the bias of $\mathcal{C}_i$, and $\eta_c^i(y)$ is the variance of $\mathcal{C}_i$ given input y. In the following discussion, we assume the error consists of variance only, as our major goal is to reduce the error caused by the discrepancies among the classifiers trained on different data chunks.

Assume an incoming data stream is partitioned into sequential chunks of fixed size, $\mathsf{S}_1, \mathsf{S}_2, \cdots, \mathsf{S}_n$, with S_n being the most recent chunk. Let $\mathcal{C}_i$, G_k, and E_k denote the following models.

- $\mathcal{C}_i$: classifier learned from training set S_i;
- G_k : classifier learned from the training set consisting of the last k chunks $\mathsf{S}_{n-k+1} \cup \cdots \cup \mathsf{S}_n$;
- E_k : classifier ensemble consisting of the last k classifiers $\mathcal{C}_{n-k+1}, \cdots, \mathcal{C}_n$.

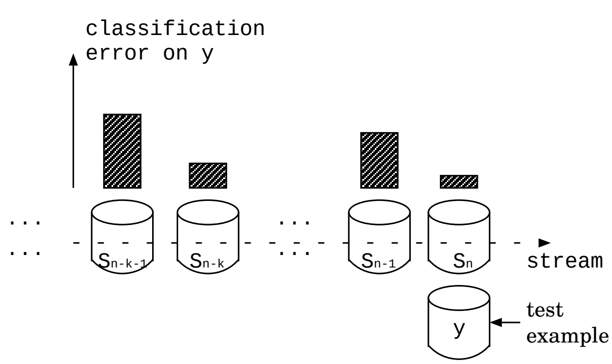


Figure 3: Models' classification error on test example y.

In the concept-drifting environment, models learned up-stream may carry significant variances when they are applied to the current test cases (Figure 3). Thus, instead of averaging the outputs of classifiers in the ensemble, we use the weighted approach. We assign each classifier $\mathcal{C}_i$ a weight w_i, such that w_i is reversely proportional to $\mathcal{C}_i$'s expected error (when applied to the current test cases). In Section 4, we introduce a method of generating such weights based on estimated classification errors. Here, assuming each classifier is so weighted, we prove the following property.

E_k produces a smaller classification error than G_k, if classifiers in E_k are weighted by their expected classification accuracy on the test data.

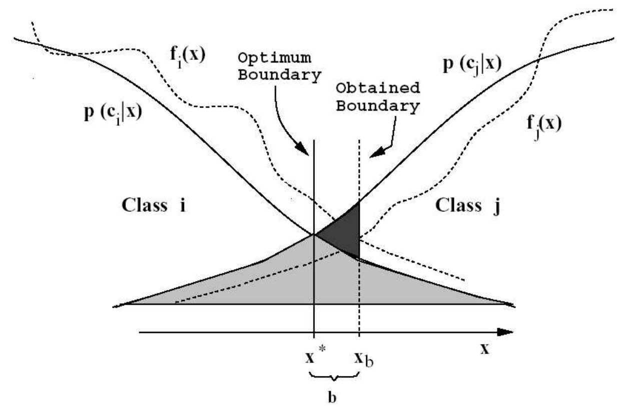


Figure 4: Error regions associated with approximating the *a posteriori* probabilities [28].

We prove this property through bias-variance decomposition based on Tumer's work [28]. The Bayes optimum decision assigns x to class i if $p(c_i|x) > p(c_k|x), \forall k \neq i$. Therefore, as shown in Figure 4, the Bayes optimum boundary is the loci of all points x^* such that $p(c_i|x^*) = p(c_j|x^*)$, where $j = \text{argmax}_k \, p(c_k|x^*)$ [28]. The decision boundary of our classifier may vary from the optimum boundary. In Figure 4, $b = x_b - x^*$ denotes the amount by which the boundary of the classifier differs from the optimum boundary. In other words, patterns corresponding to the darkly shaded region are erroneously classified by the classifier. The classifier thus introduces an expected error Err in addition to the error of the Bayes optimum decision:

$$Err = \int_{-\infty}^{\infty} A(b) f_b(b) db$$

where $A(b)$ is the area of the darkly shaded region, and f_b is the density function for b. Tumer et al [28] proves that the expected added error can be expressed by:

$$Err = \frac{\sigma_{\eta_c}^2}{s} \tag{2}$$

where $s = p'(c_j|x^*) - p'(c_i|x^*)$ is independent of the trained model[1], and $\sigma_{\eta_c}^2$ denotes the variances of $\eta_c(x)$.

Thus, given a test example y, the probability output of the single classifier G_k can be expressed as:

$$f_c^g(y) = p(c|y) + \eta_c^g(y)$$

Assuming each partition S_i is of the same size, we study $\sigma_{\eta_c^g}^2$, the variance of $\eta_c^g(y)$. If each S_i has identical class distribution, that is, there is no conceptual drift, then the single classifier G_k, which is learned from k partitions, can reduce the average variance by a factor of k. With the presence of conceptual drifts, we have:

$$\sigma_{\eta_c^g}^2 \geq \frac{1}{k^2} \sum_{i=n-k+1}^{n} \sigma_{\eta_c^i}^2 \tag{3}$$

For the ensemble approach, we use weighted averaging to combine outputs of the classifiers in the ensemble. The probability output of the ensemble is given by:

$$f_c^E(y) = \sum_{i=n-k+1}^{n} w_i f_c^i(y) / \sum_{i=n-k+1}^{n} w_i \tag{4}$$

[1] Here, $p'(c_j|\cdot)$ denotes the derivative of $p(c_j|\cdot)$.

where w_i is the weight of the i-th classifier, which is assumed to be reversely proportional to Err_i (c is a constant):

$$w_i = \frac{c}{\sigma^2_{\eta^i_c}} \tag{5}$$

The probability output E_k (4) can also be expressed as:

$$f^E_c(y) = p(c|y) + \eta^E_c(y)$$

where

$$\eta^E_c(y) = \sum_{i=n-k+1}^{n} w_i \eta^i_c(y) / \sum_{i=n-k+1}^{n} w_i$$

Assuming the variances of different classifiers are independent, we derive the variance of $\eta^E_c(y)$:

$$\sigma^2_{\eta^E_c} = \sum_{i=n-k+1}^{n} w_i^2 \sigma^2_{\eta^i_c} / (\sum_{i=n-k+1}^{n} w_i)^2 \tag{6}$$

We use the reverse proportional assumption of (5) to simplify (6) to the following:

$$\sigma^2_{\eta^E_c} = 1 / \sum_{i=n-k+1}^{n} \frac{1}{\sigma^2_{\eta^i_c}} \tag{7}$$

It is easy to prove:

$$\sum_{i=n-k+1}^{n} \sigma^2_{\eta^i_c} \sum_{i=n-k+1}^{n} \frac{1}{\sigma^2_{\eta^i_c}} \geq k^2$$

or equivalently:

$$1 / \sum_{i=n-k+1}^{n} \frac{1}{\sigma^2_{\eta^i_c}} \leq \frac{1}{k^2} \sum_{i=n-k+1}^{n} \sigma^2_{\eta^i_c}$$

which based on (3) and (7) means:

$$\sigma^2_{\eta^E_c} \leq \sigma^2_{\eta^g_c}$$

and thus, we have proved:

$$Err^E \leq Err^G$$

This means, compared with the single classifier G_k, which is learned from the examples in the entire window of k chunks, the classifier ensemble approach is capable of reducing classification error through a weighting scheme where a classifier's weight is reversely proportional to its expected error.

Note that the above property does not guarantee that E_k has higher accuracy than classifier G_j if $j < k$. For instance, if the concept drifts between the partitions are so dramatic that S_{n-1} and S_n represent totally conflicting concepts, then adding $\mathcal{C}_{n-1}$ in decision making will only raise classification error. A weighting scheme should assign classifiers representing totally conflicting concepts near-zero weights. We discuss how to tune weights in detail in Section 4.

4. CLASSIFIER ENSEMBLE FOR DRIFTING CONCEPTS

The proof of the error reduction property in Section 3 showed that a classifier ensemble can outperform a single classifier in the presence of concept drifts. To apply it to real-world problems we need to assign an actual weight to each classifier that reflects its predictive accuracy on the current testing data.

4.1 Accuracy-Weighted Ensembles

The incoming data stream is partitioned into sequential chunks, $\mathsf{S}_1, \mathsf{S}_2, \cdots, \mathsf{S}_n$, with S_n being the most up-to-date chunk, and each chunk is of the same size, or `ChunkSize`. We learn a classifier $\mathcal{C}_i$ for each S_i, $i \geq 1$.

According to the error reduction property, given test examples T, we should give each classifier $\mathcal{C}_i$ a weight reversely proportional to the expected error of $\mathcal{C}_i$ in classifying T. To do this, we need to know the actual function being learned, which is unavailable.

We derive the weight of classifier $\mathcal{C}_i$ by *estimating* its expected prediction error on the test examples. We assume the class distribution of S_n, the most recent training data, is closest to the class distribution of the current test data. Thus, the weights of the classifiers can be approximated by computing their classification error on S_n.

More specifically, assume that S_n consists of records in the form of (x, c), where c is the true label of the record. $\mathcal{C}_i$'s classification error of example (x, c) is $1 - f^i_c(x)$, where $f^i_c(x)$ is the probability given by $\mathcal{C}_i$ that x is an instance of class c. Thus, the mean square error of classifier $\mathcal{C}_i$ can be expressed by:

$$\text{MSE}_i = \frac{1}{|\mathsf{S}_n|} \sum_{(x,c)\in \mathsf{S}_n} (1 - f^i_c(x))^2$$

The weight of classifier $\mathcal{C}_i$ should be reversely proportional to MSE_i. On the other hand, a classifier predicts randomly (that is, the probability of x being classified as class c equals to c's class distributions $p(c)$) will have mean square error:

$$\text{MSE}_r = \sum_c p(c)(1 - p(c))^2$$

For instance, if $c \in \{0, 1\}$ and the class distribution is uniform, we have $\text{MSE}_r = .25$. Since a random model does not contain useful knowledge about the data, we use MSE_r, the error rate of the random classifier as a threshold in weighting the classifiers. That is, we discard classifiers whose error is equal to or larger than MSE_r. Furthermore, to make computation easy, we use the following weight w_i for classifier $\mathcal{C}_i$:

$$w_i = \text{MSE}_r - \text{MSE}_i \tag{8}$$

For cost-sensitive applications such as credit card fraud detection, we use the benefits (e.g., total fraud amount detected) achieved by classifier C_i on the most recent training data S_n as its weight.

	predict $fraud$	predict $\neg fraud$
actual $fraud$	$t(x) - cost$	0
actual $\neg fraud$	$-cost$	0

Table 1: Benefit matrix $b_{c,c'}$

Assume the benefit of classifying transaction x of actual class c as a case of class c' is $b_{c,c'}(x)$. Based on the benefit matrix shown in Table 1 (where $t(x)$ is the transaction amount, and $cost$ is the fraud investigation cost), the total benefits achieved by $\mathcal{C}_i$ is:

$$b_i = \sum_{(x,c)\in \mathsf{S}_n} \sum_{c'} b_{c,c'}(x) \cdot f^i_{c'}(x)$$

and we assign the following weight to $\mathcal{C}_i$:

$$w_i = b_i - b_r \tag{9}$$

where b_r is the benefits achieved by a classifier that predicts randomly. Also, we discard classifiers with 0 or negative weights.

Since we are handling infinite incoming data flows, we will learn an infinite number of classifiers over the time. It is impossible and unnecessary to keep and use all the classifiers for prediction. Instead, we only keep the top K classifiers with the highest prediction accuracy on the current training data. In Section 5, we discuss ensemble pruning in more detail and present a technique for instance-based pruning.

Algorithm 1 gives an outline of the classifier ensemble approach for mining concept-drifting data streams. Whenever a new chunk of data has arrived, we build a classifier from the data, and use the data to tune the weights of the previous classifiers. Usually, `ChunkSize` is small (our experiments use chunks of size ranging from 1,000 to 25,000 records), and the entire chunk can be held in memory with ease.

The algorithm for classification is straightforward, and it is omitted here. Basically, given a test case y, each of the K classifiers is applied on y, and their outputs are combined through weighted averaging.

Input: S: a dataset of `ChunkSize` from the incoming stream
K: the total number of classifiers
$\mathcal{C}$: a set of K previously trained classifiers
Output: $\mathcal{C}$: a set of K classifiers with updated weights

train classifier $\mathcal{C}'$ from S;
compute error rate / benefits of $\mathcal{C}'$ via cross validation on S;
derive weight w' for $\mathcal{C}'$ using (8) or (9);
for *each classifier* $\mathcal{C}_i \in \mathcal{C}$ **do**
  apply $\mathcal{C}_i$ on S to derive MSE_i or b_i;
  compute w_i based on (8) and (9);
$\mathcal{C} \leftarrow K$ of the top weighted classifiers in $C \cup \{C'\}$;
return $\mathcal{C}$;

Algorithm 1: A classifier ensemble approach for mining concept-drifting data streams

4.2 Complexity

Assume the complexity for building a classifier on a data set of size s is $f(s)$. The complexity to classify a test data set in order to tune its weight is linear in the size of the test data set. Suppose the entire data stream is divided into a set of n partitions, then the complexity of Algorithm 1 is $O(n \cdot f(s/n) + Ks)$, where $n \gg K$. On the other hand, building a single classifier on s requires $O(f(s))$. For most classifier algorithms, $f(\cdot)$ is super-linear, which means the ensemble approach is more efficient.

5. ENSEMBLE PRUNING

A classifier ensemble combines the probability or the benefit output of a set of classifiers. Given a test example y, we need to consult every classifier in the ensemble, which is often time consuming in an online streaming environment.

5.1 Overview

In many applications, the combined result of the classifiers usually converges to the final value well before all classifiers are consulted. The goal of pruning is to identify a subset of classifiers that achieves the same level of total benefits as the entire ensemble.

Traditional pruning is based on classifiers' overall performances (e.g., average error rate, average benefits, etc.). Several criteria can be used in pruning. The first criterion is mean square error. The goal is to find a set of n classifiers that has the minimum mean square error. The second approach favors classifier diversity, as diversity is the major factor in error reduction. KL-distance, or relative entropy, is a widely used measure for difference between two distributions. The KL-distance between two distributions p and q is defined as $D(p||q) = \sum_x p(x) \log p(x)/q(x)$. In our case, p and q are the class distributions given by two classifiers. The goal is then to find the set of classifiers S that maximizes mutual KL-distances.

It is, however, impractical to search for the optimal set of classifiers based on the MSE criterion or the KL-distance criterion. Even greedy methods are time consuming: the complexities of the greedy methods of the two approaches are $O(|T| \cdot N \cdot K)$ and $O(|T| \cdot N \cdot K^2)$ respectively, where N is the total number of available classifiers.

Besides the complexity issue, the above two approaches do not apply to cost-sensitive applications. Moreover, the applicability of the KL-distance criterion is limited to streams with no or mild concept drifting only, since concept drifts also enlarge the KL-distance.

In this paper, we apply the instance-based pruning technique [9] to data streams with conceptual drifts.

5.2 Instance Based Pruning

Cost-sensitive applications usually provide higher error tolerance. For instance, in credit card fraud detection, the decision threshold of whether to launch an investigation or not is:

$$p(\text{fraud}|y) \cdot t(y) > cost$$

where $t(y)$ is the amount of transaction y. In other words, as long as $p(\text{fraud}|y) > cost/t(y)$, transaction y will be classified as *fraud* no matter what the exact value of $p(\text{fraud}|y)$ is. For example, assuming $t(y) = \$900$, $cost = \$90$, both $p(\text{fraud}|y) = 0.2$ and $p(\text{fraud}|y) = 0.4$ result in the same prediction. This property helps reduce the "expected" number of classifiers needed in prediction.

We use the following approach for instance based ensemble pruning [9]. For a given ensemble S consisting of K classifiers, we first order the K classifiers by their decreasing weight into a "pipeline". (The weights are tuned for the most-recent training set.) To classify a test example y, the classifier with the highest weight is consulted first, followed by other classifiers in the pipeline. This pipeline procedure stops as soon as a "confident prediction" can be made or there are no more classifiers in the pipeline.

More specifically, assume that $\mathcal{C}_1, \cdots, \mathcal{C}_K$ are the classifiers in the pipeline, with $\mathcal{C}_1$ having the highest weight. After consulting the first k classifiers $\mathcal{C}_1, \cdots, \mathcal{C}_k$, we derive the current weighted probability as:

$$F_k(x) = \frac{\sum_{i=1}^{k} w_i \cdot p_i(\text{fraud}|x)}{\sum_{i=1}^{k} w_i}$$

The final weighted probability, derived after all K classifiers are consulted, is $F_K(x)$. Let $\epsilon_k(x) = F_k(x) - F_K(x)$ be the error at stage k. The question is, if we ignore $\epsilon_k(x)$ and use $F_k(x)$ to decide whether to launch a fraud investigation or not, how much confidence do we have that using $F_K(x)$ would have reached the same decision?

Algorithm 2 estimates the confidence. We compute the mean and the variance of $\epsilon_k(x)$, assuming that $\epsilon_k(x)$ follows normal distribution. The mean and variance statistics can be obtained by evaluating $F_k(.)$ on the current training set for every classifier $\mathcal{C}_k$. To reduce the possible error range, we study the distribution under a finer grain. We divide $[0, 1]$, the range of $F_k(\cdot)$, into ξ bins. An example x is put into bin i if $F_k(x) \in [\frac{i}{\xi}, \frac{i+1}{\xi})$. We then compute $\mu_{k,i}$ and $\sigma^2_{k,i}$, the mean and the variance of the error of those training examples in bin i at stage k.

Algorithm 3 outlines the procedure of classifying an unknown instance y. We use the following decision rules after we have applied the first k classifiers in the pipeline on instance y:

$$\begin{cases} F_k(y) - \mu_{k,i} - \mathbf{t} \cdot \sigma_{k,i} > cost/t(y), & \text{fraud} \\ F_k(y) + \mu_{k,i} + \mathbf{t} \cdot \sigma_{k,i} \leq cost/t(y), & \text{non-fraud} \\ \text{otherwise}, & \text{uncertain} \end{cases} \tag{10}$$

where i is the bin y belongs to, and $\mathbf{t}$ is a confidence interval parameter. Under the assumption of normal distribution, $\mathbf{t} = 3$ delivers a confidence of 99.7%, and $\mathbf{t} = 2$ of confidence 95%. When the prediction is uncertain, that is, the instance falls out of the $\mathbf{t}$ sigma region, the next classifier in the pipeline, $\mathcal{C}_{k+1}$, is employed, and the rules are applied again. If there are no classifiers left in the pipeline, the current prediction is returned. As a result, an example does not need to use all classifiers in the pipeline to compute a confident prediction. The "expected" number of classifiers can be reduced.

Input: S: a dataset of `ChunkSize` from the incoming stream
K: the total number of classifiers
ξ: number of bins
$\mathcal{C}$: a set of K previously trained classifiers
Output: $\mathcal{C}$: a set of K classifiers with updated weights
μ, σ: mean and variance for each stage and each bin

train classifier $\mathcal{C}'$ from S;
compute error rate / benefits of $\mathcal{C}'$ via cross validation on S;
derive weight w' for $\mathcal{C}'$ using (8) or (9);
for *each classifier* $\mathcal{C}_k \in \mathcal{C}$ **do**
  apply $\mathcal{C}_k$ on S to derive MSE_k or b_k;
  compute w_k based on (8) and (9);
$\mathcal{C} \leftarrow K$ of the top weighted classifiers in $\mathcal{C} \cup \{\mathcal{C}'\}$;
for *each* $y \in S$ **do**
  compute $F_k(y)$ for $k = 1, \cdots, K$;
  y belongs to bin (i, k) if $F_k(y) \in [\frac{i}{\xi}, \frac{i+1}{\xi})$;
  incrementally updates $\mu_{i,k}$ and $\sigma_{i,k}$ for bin (i, k);

Algorithm 2: Obtaining $\mu_{k,i}$ and $\sigma_{k,i}$ during ensemble construction

5.3 Complexity

Algorithm 3 outlines the procedure of instance based pruning. To classify a dataset of size s, the worst case complexity is $O(Ks)$. In the experiments, we show that the actual number of classifiers can be reduced dramatically without affecting the classification performance.

The cost of instance based pruning mainly comes from updating $\mu_{k,i}$ and $\sigma^2_{k,i}$ for each k and i. These statistics are obtained during training time. The procedure shown in Algorithm 2 is an improved version of Algorithm 1. The complexity of Algorithm 2 remains $O(n \cdot f(s/n) + Ks)$ (updating of the statistics costs $O(Ks)$), where s is the size of the data stream, and n is the number of partitions of the data stream.

6. EXPERIMENTS

We conducted extensive experiments on both synthetic and real life data streams. Our goals are to demonstrate the error reduction effects of weighted classifier ensembles, to evaluate the impact of the frequency and magnitude of the concept drifts on prediction accuracy, and to analyze the advantage of our approach over alternative methods such as incremental learning.

Input: y: a test example
$\mathbf{t}$: confidence level
$\mathcal{C}$: a set of K previously trained classifiers
Output: prediction of y's class

Let $\mathcal{C} = \{\mathcal{C}_1, \cdots, \mathcal{C}_n\}$ with $w_i \geq w_j$ for $i < j$;
$F_0(y) \leftarrow 0$;
$w \leftarrow 0$;
for $k = \{1, \cdots, K\}$ **do**
  $F_k(y) \leftarrow \frac{F_{k-1} \cdot w + w_k \cdot p_k(fraud|x)}{w + w_k}$;
  $w \leftarrow w + w_k$;
  let i be the bin y belongs to;
  apply rules in (10) to check if y is in $\mathtt{t}$-σ region;
  return fraud/non-fraud if $\mathtt{t}$-σ confidence is reached;
if $F_K(y) > cost/t(y)$ **then**
  return fraud;
else
  return non-fraud;

Algorithm 3: Classification with Instance Based Pruning

The base models used in our tests are C4.5 [24], the RIPPER rule learner [6], and the Naive Bayesian method [23]. The tests are conducted on a Linux machine with a 770 MHz CPU and 256 MB main memory.

6.1 Algorithms used in Comparison

We denote a classifier ensemble with a capacity of K classifiers as E_K. Each classifier is trained by a data set of size `ChunkSize`. We compare with algorithms that rely on a single classifier for mining streaming data. We assume the classifier is continuously being revised by the data that have just arrived and the data being faded out. We call it a window classifier, since only the data in the most recent window have influence on the model. We denote such a classifier by G_K, where K is the number of data chunks in the window, and the total number of the records in the window is $K \cdot$`ChunkSize`. Thus, ensemble E_K and G_K are trained from the same amount of data. Particularly, we have $E_1 = G_1$. We also use G_0 to denote the classifier built on the entire historical data starting from the beginning of the data stream up to now. For instance, BOAT [15] and VFDT [7] are G_0 classifiers, while CVFDT [21] is a G_K classifier.

6.2 Streaming Data

We use both synthetic and real life data streams.

Synthetic Data. We create synthetic data with drifting concepts based on a moving hyperplane. A hyperplane in d-dimensional space is denoted by equation:

$$\sum_{i=1}^{d} a_i x_i = a_0 \tag{11}$$

We label examples satisfying $\sum_{i=1}^{d} a_i x_i \geq a_0$ as positive, and examples satisfying $\sum_{i=1}^{d} a_i x_i < a_0$ as negative. Hyperplanes have 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 [21].

We generate random examples uniformly distributed in multi-dimensional space $[0,1]^d$. Weights a_i ($1 \leq i \leq d$) in (11) are initialized by random values in the range of $[0,1]$. We choose the value of a_0 so that the hyperplane cuts the multi-dimensional space in two parts of the same volume, that is, $a_0 = \frac{1}{2}\sum_{i=1}^{d} a_i$. Thus, roughly half of the examples are positive, and the other half are negative. Noise is introduced by randomly switching the labels of $p\%$ of the examples. In our experiments, the noise level $p\%$ is set to 5%.

We simulate concept drifts through a series of parameters. Parameter k specifies the total number of dimensions whose weights are involved in changing. Parameter $t \in \mathcal{R}$ specifies the magnitude of the change (every N examples) for weights $a_1, \cdots, a_k$, and $s_i \in \{-1, 1\}$ specifies the direction of change for each weight a_i, $1 \leq i \leq k$. Weights change continuously, i.e., a_i is adjusted by $s_i \cdot t/N$ after each example is generated. Furthermore, there is a possibility of 10% that the change would reverse direction after every N examples are generated, that is, s_i is replaced by $-s_i$ with probability 10%. Also, each time the weights are updated, we recompute $a_0 = \frac{1}{2}\sum_{i=1}^{d} a_i$ so that the class distribution is not disturbed.

Credit Card Fraud Data. We use real life credit card transaction flows for cost-sensitive mining. The data set is sampled from credit card transaction records within a one year period and contains a total of 5 million transactions. Features of the data include the time of the transaction, the merchant type, the merchant location, past payments, the summary of transaction history, etc. A detailed description of this data set can be found in [26]. We use the benefit matrix shown in Table 1 with the cost of disputing and investigating a fraud transaction fixed at $cost = \$90$.

The total benefit is the sum of recovered amount of fraudulent transactions less the investigation cost. To study the impact of concept drifts on the benefits, we derive two streams from the dataset. Records in the 1st stream are ordered by transaction time, and records in the 2nd stream by transaction amount.

6.3 Experimental Results

Time Analysis. We first study the time complexity of the ensemble approach. We generate synthetic data streams and train single decision tree classifiers and decision tree ensembles with varied `ChunkSize`. Consider a window of $K = 100$ chunks in the data stream. Figure 5 shows that the ensemble approach E_K is much more efficient than the corresponding single-classifier G_K in training.

Smaller `ChunkSize` offers better training performance. However, `ChunkSize` also affects classification error. Figure 5 shows the relationship between error rate (of E_{10}, e.g.) and `ChunkSize`. The dataset is generated with certain concept drifts (weights of 20% of the dimensions change $t = 0.1$ per $N = 1000$ records), large chunks produce higher error rates because the ensemble cannot detect the concept drifts occurring inside the chunk. Small chunks can also drive up error rates if the number of classifiers in an ensemble is not large enough. This is because when `ChunkSize` is small, each individual classifier in the ensemble is not supported by enough amount of training data.

Pruning Effects. Pruning improves classification efficiency. We examine the effects of instance based pruning using the credit card fraud data. In Figure 6(a), we show the total benefits achieved by ensemble classifiers with and without instance-based pruning. The X-axis represents the number of the classifiers in the ensemble, K,

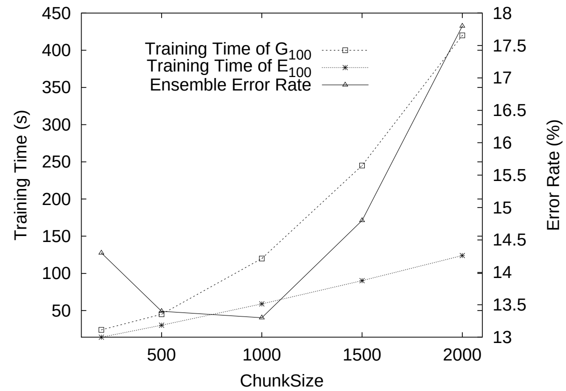


Figure 5: Training Time, `ChunkSize`, and Error Rate

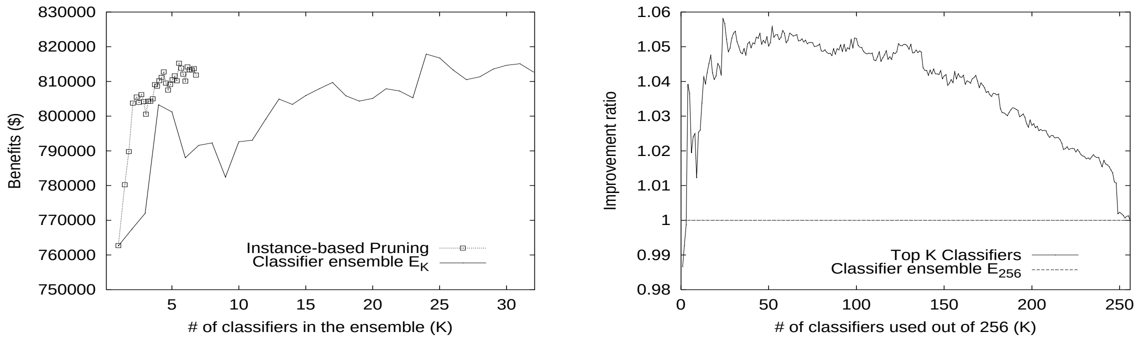
which ranges from 1 to 32. When instance-based pruning is in effect, the actual number of classifiers to be consulted is reduced. In the figure, we overload the meaning of the X-axis to represent the average number of classifiers used under instance-based pruning. For E_{32}, pruning reduces the average number of classifiers to 6.79, a reduction of 79%. Still, it achieves a benefit of \$811,838, which is just a 0.1% drop from \$812,732 – the benefit achieved by E_{32} which uses all 32 classifiers.

Figure 6(b) studies the same phenomena using 256 classifiers ($K = 256$). Instead of dynamic pruning, we use the top K classifiers, and the Y-axis shows the benefits improvement ratio. The top ranked classifiers in the pipeline outperform E_{256} in almost all the cases except if only the 1st classifier in the pipeline is used.

Error Analysis. We use decision tree classifier C4.5 as our base model, and compare the error rates of the single classifier approach and the ensemble approach. The results are shown in Figure 7 and Table 2. The synthetic datasets used in this study have 10 dimensions ($d = 10$). Figure 7 shows the averaged outcome of tests on data streams generated with varied concept drifts (the number of dimensions with changing weights ranges from 2 to 8, and the magnitude of the change t ranges from 0.10 to 1.00 for every 1000 records).

First, we study the impact of ensemble size (total number of classifiers in the ensemble) on classification accuracy. Each classifier is trained from a dataset of size ranging from 250 records to 1000 records, and their averaged error rates are shown in Figure 7(a). Apparently, when the number of classifiers increases, due to the increase of diversity of the ensemble, the error rate of E_k drops significantly. The single classifier, G_k, trained from the same amount of the data, has a much higher error rate due to the changing concepts in the data stream. In Figure 7(b), we vary the chunk size and average the error rates on different K ranging from 2 to 8. It shows that the error rate of the ensemble approach is about 20% lower than the single-classifier approach in all the cases. A detailed comparison between single- and ensemble-classifiers is given in Table 2, where G_0 represents the global classifier trained by the entire history data, and we use **bold** font to indicate the better result of G_k and E_k for $K = 2, 4, 6, 8$.

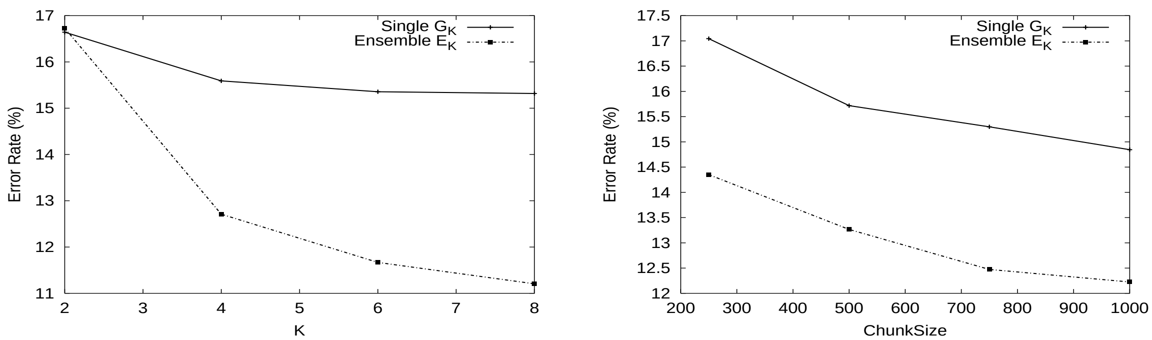
We also tested the Naive Bayesian and the RIPPER classifier under the same setting. The results are shown in Table 3 and Table 4. Although C4.5, Naive Bayesian, and RIPPER deliver different accuracy rates, they confirmed that, with a reasonable amount of classifiers (K) in the ensemble, the ensemble approach outperforms the single classifier approach.



(a) Reduction of ensemble size by instance-based pruning

(b) Benefit improvement of pruned ensemble for credit card dataset

Figure 6: Effects of Instance-Based Pruning



(a) Varying window size/ensemble size

(b) Varying `ChunkSize`

Figure 7: Average Error Rate of Single and Ensemble Decision Tree Classifiers

`ChunkSize`	G_0	$G_1 = E_1$	G_2	E_2	G_4	E_4	G_6	E_6	G_8	E_8
250	18.09	18.76	**18.00**	18.37	16.70	**14.02**	16.72	**12.82**	16.76	**12.19**
500	17.65	17.59	**16.39**	17.16	16.19	**12.91**	15.32	**11.74**	14.97	**11.25**
750	17.18	16.47	16.29	**15.77**	15.07	**12.09**	14.97	**11.19**	14.86	**10.84**
1000	16.49	16.00	15.89	**15.62**	14.40	**11.82**	14.41	**10.92**	14.68	**10.54**

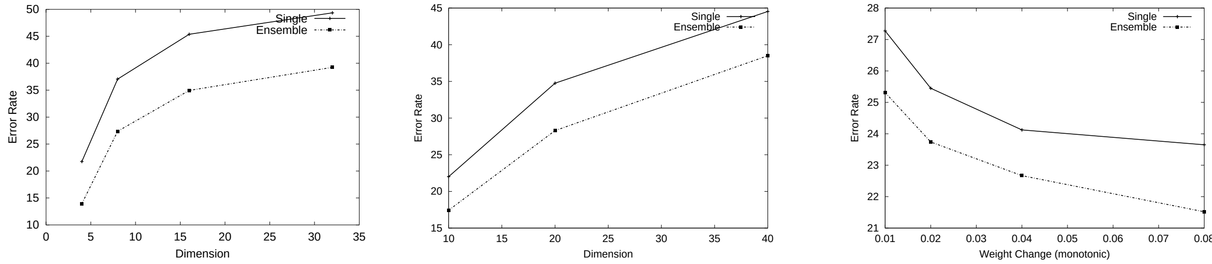
Table 2: Error Rate (%) of Single and Ensemble Decision Tree Classifiers

`ChunkSize`	G_0	G_1=E_1	G_2	E_2	G_4	E_4	G_6	E_6	G_8	E_8
250	11.94	8.09	7.91	**7.48**	8.04	**7.35**	8.42	**7.49**	8.70	**7.55**
500	12.11	7.51	7.61	**7.14**	7.94	**7.17**	8.34	**7.33**	8.69	**7.50**
750	12.07	7.22	7.52	**6.99**	7.87	**7.09**	8.41	**7.28**	8.69	**7.45**
1000	15.26	7.02	7.79	**6.84**	8.62	**6.98**	9.57	**7.16**	10.53	**7.35**

Table 3: Error Rate (%) of Single and Ensemble Naive Bayesian Classifiers

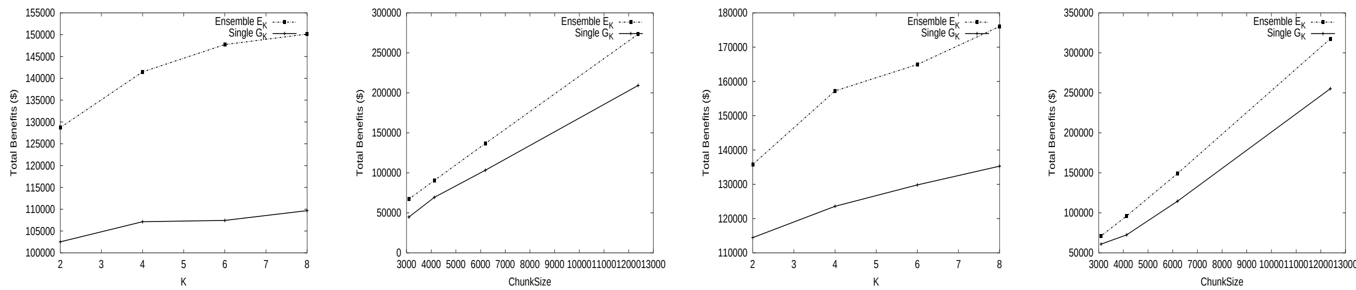
`ChunkSize`	G_0	G_1=E_1	G_2	E_2	G_4	E_4	G_6	E_6	G_8	E_8
50	27.05	24.05	22.85	**22.51**	21.55	**19.34**	24.05	**22.51**	19.34	**17.84**
100	25.09	21.97	**19.85**	20.66	**17.48**	17.50	21.97	**20.66**	17.50	**15.91**
150	24.19	20.39	**18.28**	19.11	17.22	**16.39**	20.39	**19.11**	16.39	**15.03**

Table 4: Error Rate (%) of Single and Ensemble RIPPER Classifiers



(a) # of changing dimensions (b) total dimensionality (c) (monotonic) weight change per dimension

Figure 8: Magnitude of Concept Drifts



(a) Varying K (original stream) (b) Varying `ChunkSize` (original stream) (c) Varying K (simulated stream) (d) Varying `ChunkSize` (simulated stream)

Figure 9: Averaged Benefits using Single Classifiers and Classifier Ensembles

Concept Drifts. Figure 8 studies the impact of the magnitude of the concept drifts on classification error. Concept drifts are controlled by two parameters in the synthetic data: i) the number of dimensions whose weights are changing, and ii) the magnitude of weight change per dimension. Figure 8 shows that the ensemble approach outperform the single-classifier approach under all circumstances. Figure 8(a) shows the classification error of G_k and E_k (averaged over different K) when 4, 8, 16, and 32 dimensions' weights are changing (the change per dimension is fixed at $t = 0.10$). Figure 8(b) shows the increase of classification error when the dimensionality of dataset increases. In the datasets, 40% dimensions' weights are changing at ± 0.10 per 1000 records. An interesting phenomenon arises when the weights change monotonically (weights of some dimensions are constantly increasing, and others constantly decreasing). In Figure 8(c), classification error drops when the change rate increases. This is because of the following. Initially, all the weights are in the range of $[0, 1]$. Monotonic changes cause some attributes to become more and more 'important', which makes the model easier to learn.

Cost-sensitive Learning. For cost-sensitive applications, we aim at maximizing benefits. In Figure 9(a), we compare the single classifier approach with the ensemble approach using the credit card transaction stream. The benefits are averaged from multiple runs with different chunk size (ranging from 3000 to 12000 transactions per chunk). Starting from $K = 2$, the advantage of the ensemble approach becomes obvious.

In Figure 9(b), we average the benefits of E_k and G_k ($K = 2, \cdots, 8$) for each fixed chunk size. The benefits increase as the chunk size does, as more fraudulent transactions are discovered in the chunk. Again, the ensemble approach outperforms the single classifier approach.

To study the impact of concept drifts of different magnitude, we derive another data stream from the credit card transactions. The simulated stream is obtained by sorting the original 5 million transactions by their transaction amount. We perform the same test on the simulated stream, and the results are shown in Figure 9(c) and 9(d).

Detailed results of the above tests are given in Table 6 and 5.

7. DISCUSSION AND RELATED WORK

Data stream processing has recently become a very important research domain. Much work has been done on modeling [1], querying [2, 14, 18], and mining data streams, for instance, several papers have been published on classification [7, 21, 27], regression analysis [5], and clustering [19].

Traditional data mining algorithms are challenged by two characteristic features of data streams: the infinite data flow and the drifting concepts. As methods that require multiple scans of the datasets [25, 16] can not handle infinite data flows, several incremental algorithms [15, 7] that refine models by continuously incorporating new data from the stream have been proposed. In order to handle drifting concepts, these methods are revised again to achieve the goal that effects of old examples are eliminated at a certain rate. In terms of an incremental decision tree classifier, this means we have to discard, re-grow sub trees, or build alternative subtrees under a node [21]. The resulting algorithm is often complicated, which indicates substantial efforts are required to adapt state-of-the-art learning methods to the infinite, concept-drifting streaming environment. Aside from this undesirable aspect, incremental methods are also hindered by their prediction accuracy. Since old examples are discarded at a fixed rate (no matter if they represent the changed concept or not), the learned model is supported only by the current snapshot – a relatively small amount of data. This usually results in larger prediction variances.

Classifier ensembles are increasingly gaining acceptance in the

ChunkSize	G_0	G_1=E_1	G_2	E_2	G_4	E_4	G_6	E_6	G_8	E_8
12000	296144	207392	233098	**268838**	248783	**313936**	263400	**327331**	275707	**360486**
6000	146848	102099	102330	**129917**	113810	**148818**	118915	**155814**	123170	**162381**
4000	96879	62181	66581	**82663**	72402	**95792**	74589	**101930**	76079	**103501**
3000	65470	51943	55788	**61793**	59344	**70403**	62344	**74661**	66184	**77735**

Table 5: Benefits (US $) using Single Classifiers and Classifier Ensembles (simulated stream)

ChunkSize	G_0	G_1=E_1	G_2	E_2	G_4	E_4	G_6	E_6	G_8	E_8
12000	201717	203211	197946	**253473**	211768	**269290**	211644	**282070**	215692	**289129**
6000	103763	98777	101176	**121057**	102447	**138565**	103011	**143644**	106576	**143620**
4000	69447	65024	68081	**80996**	69346	**90815**	69984	**94400**	70325	**96153**
3000	43312	41212	42917	**59293**	44977	**67222**	45130	**70802**	46139	**71660**

Table 6: Benefits (US $) using Single Classifiers and Classifier Ensembles (original stream)

data mining community. The popular approaches to creating ensembles include changing the instances used for training through techniques such as Bagging [3], Boosting [13], and pasting [4]. The classifier ensembles have several advantages over single model classifiers. First, classifier ensembles offer a significant improvement in prediction accuracy [13, 28]. Second, building a classifier ensemble is more efficient than building a single model, since most model construction algorithms have super-linear complexity. Third, the nature of classifier ensembles lend themselves to scalable parallelization [20] and on-line classification of large databases [4]. Previously, we used averaging ensemble for scalable learning over very-large datasets [12]. We show that a model's performance can be estimated before it is completely learned [10, 11]. In this work, we use weighted ensemble classifiers on concept-drifting data streams. It combines multiple classifiers weighted by their expected prediction accuracy on the current test data. Compared with incremental models trained by data in the most recent window, our approach combines talents of set of experts based on their credibility and adjusts much nicely to the underlying concept drifts. Also, we introduced the dynamic classification technique [9] to the concept-drifting streaming environment, and our results show that it enables us to dynamically select a subset of classifiers in the ensemble for prediction without loss in accuracy.

8. REFERENCES

[1] B. Babcock, S. Babu, M. Datar, R. Motawani, and J. Widom. Models and issues in data stream systems. In *PODS*, 2002.

[2] S. Babu and J. Widom. Continuous queries over data streams. *SIGMOD Record*, 30:109–120, 2001.

[3] Eric Bauer and Ron Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, 36(1-2):105–139, 1999.

[4] L. Breiman. Pasting bites together for prediction in large data sets and on-line. Technical report, Statistics Dept., UC Berkeley, 1996.

[5] 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.

[6] William Cohen. Fast effective rule induction. In *ICML*, pages 115–123, 1995.

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

[8] P. Domingos. A unified bias-variance decomposition and its applications. In *ICML*, pages 231–238, 2000.

[9] Wei Fan, Fang Chu, Haixun Wang, and Philip S. Yu. Pruning and dynamic scheduling of cost-sensitive ensembles. In *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI)*, 2002.

[10] W. Fan, H. Wang, P. Yu, S. Lo, and S. Stolfo. Progressive modeling. In *ICDM*, 2002.

[11] W. Fan, H. Wang, P. Yu, S. Lo, and S. Stolfo. Inductive learning in less than one sequential scan. In *IJCAI*, 2003.

[12] W. Fan, H. Wang, P. Yu, and S. Stolfo. A framework for scalable cost-sensitive learning based on combining probabilities and benefits. In *SIAM Data Mining*, 2002.

[13] Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *ICML*, pages 148–156, 1996.

[14] L. Gao and X. Wang. Continually evaluating similarity-based pattern queries on a streaming time series. In *SIGMOD*, Madison, Wisconsin, June 2002.

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

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

[17] S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the bias/variance dilemma. *Neural Computation*, 4(1):1–58, 1992.

[18] M. Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. In *SIGMOD*, pages 58–66, Santa Barbara, CA, May 2001.

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

[20] L. Hall, K. Bowyer, W. Kegelmeyer, T. Moore, and C. Chao. Distributed learning on very large data sets. In *Workshop on Distributed and Parallel Knowledge Discover*, 2000.

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

[22] Ron Kohavi and David H. Wolpert. Bias plus variance decomposition for zero-one loss functions. In *ICML*, pages 275–283, 1996.

[23] Tom M. Mitchell. *Machine Learning*. McGraw Hill, 1997.

[24] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

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

[26] S. Stolfo, W. Fan, W. Lee, A. Prodromidis, and P. Chan. Credit card fraud detection using meta-learning: Issues and initial results. In *AAAI-97 Workshop on Fraud Detection and Risk Management*, 1997.

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

[28] Kagan Tumer and Joydeep Ghosh. Error correlation and error reduction in ensemble classifiers. *Connection Science*, 8(3-4):385–403, 1996.

[29] P. E. Utgoff. Incremental induction of decision trees. *Machine Learning*, 4:161–186, 1989.