Classification and Novel Class Detection in Data Streams with Active Mining

Mohammad M. Masud[1], Jing Gao[2], Latifur Khan[1], Jiawei Han[2], and Bhavani Thuraisingham[1]

[1] Department of Computer Science, University of Texas at Dallas
[2] Department of Computer Science, University of Illinois at Urbana-Champaign

Abstract. We present ActMiner, which addresses four major challenges to data stream classification, namely, infinite length, concept-drift, concept-evolution, and limited labeled data. Most of the existing data stream classification techniques address only the infinite length and concept-drift problems. Our previous work, MineClass, addresses the concept-evolution problem in addition to addressing the infinite length and concept-drift problems. Concept-evolution occurs in the stream when novel classes arrive. However, most of the existing data stream classification techniques, including MineClass, require that all the instances in a data stream be labeled by human experts and become available for training. This assumption is impractical, since data labeling is both time consuming and costly. Therefore, it is impossible to label a majority of the data points in a high-speed data stream. This scarcity of labeled data naturally leads to poorly trained classifiers. ActMiner actively selects only those data points for labeling for which the expected classification error is high. Therefore, ActMiner extends MineClass, and addresses the limited labeled data problem in addition to addressing the other three problems. It outperforms the state-of-the-art data stream classification techniques that use ten times or more labeled data than ActMiner.

1 Introduction

Data stream classification is more challenging than classifying static data because of several unique properties of data streams. First, data streams are assumed to have *infinite length*, which makes it impractical to store and use all the historical data for training. Therefore, traditional multi-pass learning algorithms are not directly applicable to data streams. Second, data streams observe *concept-drift*, which occurs when the underlying concept of the data changes over time. In order to address concept-drift, a classification model must continuously adapt itself to the most recent concept. Third, data streams also observe *concept-evolution*, which occurs when a novel class appears in the stream. In order to cope with concept-evolution, a classification model must be able to automatically detect novel classes when they appear, before being trained with the labeled instances of the novel class. Finally, high speed data streams suffer from *insufficient labeled data*. This is because, manual labeling is both costly and time consuming. Therefore, the speed at which the data points are labeled lags far behind the speed at which data points arrive in the stream, leaving most of the data points in the stream as unlabeled. So, supervised classification techniques suffer from the scarcity of labeled data for learning, resulting in a poorly built classifier. Most existing data stream classification techniques address only the infinite length, and concept-drift problems [1–3]. Our previous work MineClass [4] addresses the concept-evolution problem in addition to the infinite length and concept-drift problems. However, it did not address the limited labeled data problem. Our current work, ActMiner, extends MineClass by addressing all the four problems and providing a more realistic data stream classification framework than the state-of-the-art.

A solution to the infinite length problem is incremental learning, which requires a single pass over the training data. In order to cope with concept-drift, a classifier must be continuously updated to be consistent with the most recent concept. ActMiner applies a hybrid batch-incremental process [2, 5] to solve the infinite length and concept-drift problems. It divides the data stream into equal sized chunks and trains a classification model from each chunk. An ensemble of M such models is used to classify the unlabeled data. When a new data chunk becomes available for training, a new model is trained, and an old model from the ensemble is replaced with the new model. The victim for the replacement is chosen by evaluating the accuracy of each model on the latest labeled chunk. In this way, the ensemble is kept up-to-date. ActMiner also solves the concept-evolution problem by automatically detecting novel classes in the data stream. In order to detect novel class, it first identifies the test instances that are *well-separated* from the training data, and tag them as *Raw outlier*. Then raw outliers that possibly appear as a result of concept-drift or noise are filtered out. If a sufficient number of such strongly cohesive filtered outliers (called *F-outlier*s) are observed, a novel class is assumed to have appeared, and the *F-outlier*s are classified as novel class instances. Finally, ActMiner solves the limited labeled data problem by requiring only a few selected instances to be labeled. It identifies the instances for which the classification model has the highest expected error. This selection is done without knowing the true labels of those instances. By selecting only a few instances for labeling, it saves 90% or more labeling time and cost, than traditional approaches that require all instances to be labeled.

We have several contributions. First, we propose a framework that addresses four major challenges in data stream classification. To the best of our knowledge, no other existing data stream classification technique addresses all these four problems in a single framework. Second, we show how to select only a few instances in the stream for labeling, and justify this selection process both theoretically and empirically. Finally, our technique outperforms state-of-the-art data stream classification techniques using ten times or even less amount of labeled data for training. The rest of the paper is organized as follows. Section 2 discusses the related works in data stream classification. Section 3 describes the proposed approach. Section 4 then presents the experiments and analyzes the results. Section 5 concludes with directions to future work.

2 Related Work

Related works in data stream classification can be divided into three groups: i) approaches that address the infinite length and concept-drift problems, ii) approaches that address the infinite length, concept-drift, and limited labeled data problems, and iii) approaches that address the infinite length, concept-drift, and concept-evolution problems. Groups i) and ii) again can be subdivided into two subgroups: single model and ensemble classification approach.

Most of the existing techniques fall into group i). The single-model approaches in group i) apply incremental learning and adapt themselves to the most recent concept by continuously updating the current model to accommodate concept drift [1, 3, 6]. Ensemble techniques [2, 5] maintain an ensemble of models, and use ensemble voting to classify unlabeled instances. These techniques address the infinite length problem by keeping a fixed-size ensemble, and address the concept-drift problem by updating the ensemble with newer models. ActMiner also applies an ensemble classification technique. Techniques in group ii) goes one step ahead of group i) by addressing the limited labeled data problem. Some of them apply active learning [7, 8] to select the instances to be labeled, and some [9] apply random sampling along with semi-supervised clustering. ActMiner also applies active learning, but its data selection process is different from the others. Unlike other active mining techniques such as [7] that requires extra computational overhead to select the data, ActMiner does the selection on the fly during classification. Moreover, none of these approaches address the concept-evolution problem, but ActMiner does.

Techniques in group iii) are the most rare. An unsupervised novel concept detection technique for data streams is proposed in [10], but it is not applicable to multi-class classification. Our previous work MineClass [4] addresses the concept-evolution problem on a multi-class classification framework. It can detect the arrival of a novel class automatically, without being trained with any labeled instances of that class. However, it does not address the limited labeled data problem, and requires that all instances in the stream be labeled and available for training. ActMiner extends MineClass by requiring only a few chosen instances to be labeled, thereby reducing the labeling cost by 90% or more.

3 ActMiner: Active Classification and Novel Class Detection

In this section we discuss ActMiner in details. Before describing ActMiner, we briefly introduce MineClass, and present some definitions.

3.1 Background: Novel Class Detection with MineClass

ActMiner is based on our previous work MineClass [4], which also does data stream classification and novel class detection. MineClass is an ensemble classification approach, which keeps an ensemble L of M classification models, i.e., L=$\{L_1$,...,$L_M\}$. First, we define the concept of *novel class* and *existing class.*

Definition 1 (Existing class and Novel class). *Let L be the current ensemble of classification models. A class c is an existing class if at least one of the models $L_i \in L$ has been trained with class c. Otherwise, c is a novel class.*

The basic assumption in novel class detection lies in the following property.

Property 1. *Let x be an arbitrary instance belonging to a class c', and c be any class other than c'. Also, let $\lambda_{c',q}(x)$ be the q-nearest neighbors of x within class c', and $\lambda_{c,q}(x)$ be the q-nearest neighbors of x within class c. Then the mean distance from x to $\lambda_{c',q}(x)$ is less than the mean distance from x to $\lambda_{c,q}(x)$, for any class $c \neq c'$.*

In other words, property 1 states that an instance is closer to other same class instances and farther from the instances of any other class. Therefore, if a novel class arrives, the instances belonging to that class must be closer to other novel class instances and far from any existing class instances. This is the basic idea in detecting novel class with MineClass. MineClass detects novel classes in three steps: i) creating decision boundary for a classifier during its training, ii) detecting and filtering outliers, and iii) computing cohesion among the outliers, and separation of the outliers from the training data.

The decision boundaries are created by clustering the training data, and saving the cluster centroids and radii as pseudopoints. Each pseudopoint represents a hypersphere in the feature space. Union of all the hyperspheres in a classification model constitutes the decision boundary for that model. The decision boundary for the ensemble of models is the union of the decision boundaries of each model in the ensemble. Any test instance falling outside the decision boundary of the ensemble of models is considered an outlier, called *F-outlier*.

Definition 2 (F$-$outlier). *A test instance is an F$-$outlier (i.e., filtered outlier) if it is outside the decision boundary of* all *classifiers $L_i \in L$.*

If any test instance x is inside the decision boundary, then it can be shown that there is at least one existing class instance x', such that the mean distance from x to the existing class instances is less than the mean distance from x' to the existing class instances. Therefore, according to property 1, x must be an existing class instance. Any *F-outlier* is a potential novel class instance, because it is outside the decision boundary of the ensemble of models, and therefore, we its membership in the existing classes cannot be guaranteed. However, only one *F-outlier* does not imply a novel class. We need to know whether there are enough *F-outlier*s that are sufficiently close to each other and far from the existing class instances. This is done by computing the cohesion among *F-outlier*s and separation of *F-outlier*s from existing class instances. This is done using the following equation: $q\text{-}NSC(x) = \frac{b_{min}(x) - a(x)}{max(b_{min}(x), a(x))}$, where x is an *F-outlier*, $b_{min}(x)$ is the mean distance from x to its q-nearest existing class instances and $a(x)$ is the mean distance from x to its q-nearest *F-outlier* instances. A positive value indicates that x is closer to other *F-outlier* instances than the existing class instances. If q-NSC(x) is positive for at least q *F-outlier* instances, then a novel class is assumed to have arrived. This is the basic working principle of the DetectNovelClass() function in algorithm 1.

3.2 ActMiner Algorithm

ActMiner, which stands for Active Classifier for Data Streams with novel class Miner, performs classification and novel class detection in data streams while requiring very small amount of labeled data for training. The top level algorithm is sketched in algorithm 1.

Algorithm 1. ActMiner

```
1: $L \leftarrow$ Build-initial-ensemble(), $\mathcal{L} \leftarrow$ empty //training data
2: while true do
3:   $D_n \leftarrow$ the latest data chunk in the stream
4:   //Classification, outlier detection, novel class detection
5:   $buf \leftarrow$ empty //temporary buffer
6:   for each $x_k \in D_n$ do
7:     $< fout, \hat{y}_k > \leftarrow$ Classify($x_k$,$L$) //$\hat{y}_k$ is the predicted class label of $x_k$
8:     if $fout$ = true then $buf \Leftarrow x_k$ //enqueue into buffer
9:     else output prediction $< x_k, \hat{y}_k >$ end if
10:  end for
11:  $found \leftarrow$ DetectNovelClass($L$,$buf$) //(see section 3.1)
12:  if $found$=true then
13:    for each novel class instance $x_k \in buf$ do $\hat{y}_k \leftarrow$ "novel class" end for
14:  end if
15:  for each instance $x_k \in buf$ output prediction $< x_k, \hat{y}_k >$ end for
16:  //Label the chunk
17:  for each $x_k \in D_n$ do
18:    if $x_k$ is an weakly classified instance (WCI)
19:      then $\mathcal{L} \Leftarrow < x_k, y_k >$ //label it and save ($y_k$ is the true class label of $x_k$)
20:      else $\mathcal{L} \Leftarrow < x_k, \hat{y}_k >$ //save in training buffer with the predicted class label
21:    end if
22:  end for
23:  //Training
24:  $L' \leftarrow$ Train-and-save-decision-boundary ($\mathcal{L}$) //(see section 3.1)
25:  $L \leftarrow$ Update($L$,$L'$,$\mathcal{L}$)
26:  $\mathcal{L} \leftarrow$ empty
27: end while
```

The algorithm starts with building the initial ensemble $L = \{L_1, ..., L_M\}$ with the first few data chunks of the stream (line 1), and initializing the training buffer. Then a **while** loop (line 2) runs indefinitely until the stream is finished. Within the while loop, the latest data chunk D_n is examined. Each instance x_k in D_n is first passed to the Classify() function, which uses the existing ensemble to get its predicted class $\hat{y}_k$ and its F-*outlier* status (line 7). If it is identified as an F-*outlier*, then it is temporarily saved in a buffer buf for further inspection (line 8), otherwise, we output its predicted class (line 9). Then we call the DetectNovelClass() function to inspect buf to detect whether any novel class has arrived (line 11). If a novel class has arrived then the novel class instances are classified as "novel class" (line 13). Then the class predictions of all instances in *buf* are sent to the output (line 15). We then select the instances that need to be labeled (lines 17-22). Only the instances identified as *Weakly Classified Instance* (WCI) are required to be labeled by human experts, and they are saved in the training buffer with their *true* class labels (line 19). We will explain WCI shortly. All other instances remain as unlabeled, and they are saved in the training buffer with their *predicted* class labels (line 20). A new model L' is trained with the training buffer (line 24), and this model is used to update the existing ensemble L (line 25). Updating is done by first evaluating each model $L_i \in L$ on $\mathcal{L}$, and replacing the worst (based on accuracy) of them with L'. ActMiner can be applied to any base learning algorithm in general. The only operation that needs to be specific to a learning algorithm is *train and save decision boundary.*

3.3 Data Selection for Labeling

Unlike MineClass, ActMiner does not need all the instances in the training data to have true labels. Only those instances need to be labeled about whose class labels MineClass is the most uncertain. We call these instances as "weakly classified instances" or WCIs. ActMiner finds the WCIs and presents them to the user for labeling, because the ensemble has the highest uncertainty in classifying the WCIs. In order to perform ensemble voting on an instance x_j, first we initialize a vector $V = \{v[1], ..., v[C]\}$ to zeros, where C is the total number of classes, and each $v[k]$ represents a real value. Let classifier L_i predicts the class label of x_j to be c, where $c \in \{1, ..., C\}$. Then we increment $v[c]$ by 1. Let $v[max]$ represent the maximum among all $v[i]$. Then the predicted class of x_j is max. An instance x_j is a WCI if either i) The instance has been identified as an *F-outlier* (see definition 2), or ii) The *ratio* of its majority vote to its total vote is less than the *Minimum Majority Threshold* (MMT), a user-defined parameter.

For condition i), consider that *F-outlier*s are outside the decision boundary of all the models in the ensemble. So the ensemble has the highest uncertainty in classifying them. Therefore, *F-outlier*s are considered as WCIs and need to be labeled. For condition ii), let us denote the ratio with *Majority to Sum* (M2S) ratio. Let $v[max]$ be maximum in the vector V, and let $s = \sum_{i=1}^{C} v[i]$. Therefore, the M2S ratio of x_j is given by: $M2S(x_j) = \frac{v[max]}{s}$. The data point x_j is considered to be a WCI if M2S(x_j) < MMT. A lower value of M2S(x_j) indicates higher uncertainty in classifying that instance, and vice versa.

Next we justify the reason for labeling the WCIs of the second type, i.e., instances that have M2S(x_j) < MMT. We show that the ensemble classification error is higher for the instances having lower M2S.

Lemma 1. *Let $\mathcal{A}$ and $\mathcal{B}$ be two sets of disjoint datapoints such that for any $x_a \in \mathcal{A}$, and $x_b \in \mathcal{B}$, M2S(x_a) < M2S(x_b). Then the ensemble error on $\mathcal{A}$ is higher than the ensemble error on $\mathcal{B}$.*

Proof. Given an instance x, the posterior probability distribution of class c is $p(c|x)$. Let C be the total number of classes, and $c \in \{1, ..., C\}$. According to

Tumer and Ghosh [11], a classifier is trained to learn a function $f_c(.)$ that approximates this posterior probability (i.e., probability of classifying x into class c): $f_c(x) = p(c|x) + \eta_c(x)$ where $\eta_c(x)$ is the error of $f_c(x)$ relative to $p(c|x)$. This is the error in addition to Bayes error and usually referred to as the *added error*. This error occurs either due to the bias of the learning algorithm, and/or the variance of the learned model. According to [11], the expected added error can be obtained from the following formula: $Error = \frac{\sigma^2_{\eta_c(x)}}{s}$ where $\sigma^2_{\eta_c(x)}$ is the variance of $\eta_c(x)$, and s is the difference between the derivatives of $p(c|x)$ and $p(\neg c|x)$, which is independent of the learned classifier.

Let $L = \{L_1, ..., L_M\}$ be an ensemble of M classifiers, where each classifier L_i is trained from a data chunk. If we average the outputs of the classifiers in a M-classifier ensemble, then according to [11], the probability of the ensemble in classifying x into class c is: $f_c^{avg}(x) = \frac{1}{M}\sum_{m=1}^{M} f_c^m(x) = p(c|x) + \eta_c^{avg}(x)$, where $f_c^{avg}(x)$ is the output of the ensemble L, $f_c^m(x)$ is the output of the m-th classifier L_m, and $\eta_c^{avg}(x)$ is the added error of the ensemble, given by: $\eta_c^{avg}(x) = \frac{1}{M}\sum_{m=1}^{M} \eta_c^m(x)$, where $\eta_c^m(x)$ is the added error of the m-th classifier in the ensemble. Assuming the error variances are independent, the variance of $\eta_c^{avg}(x)$, i.e., the error variance of the ensemble, $\sigma^2_{\eta_c^{avg}(x)}$, is given by:

$$\sigma^2_{\eta_c^{avg}(x)} = \frac{1}{M^2}\sum_{m=1}^{M} \sigma^2_{\eta_c^m(x)}, \tag{1}$$

where $\sigma^2_{\eta_c^m(x)}$ is the variance of $\eta_c^m(x)$.

Also, let $\sigma^2_{\eta_c^{avg}(x_a)}(\mathcal{A})$, and $\sigma^2_{\eta_c^{avg}(x_b)}(\mathcal{B})$ be the variances of the ensemble error on $\mathcal{A}$, and $\mathcal{B}$, respectively. Let $z_c(x)$ be 1 if the true class label of x is c, and $z_c(x)$ be 0, otherwise. Also, let $f_c^m(x)$ be either 0 or 1. The error variance of classifier L_m on $\mathcal{A}$ is given by [7]:

$$\sigma^2_{\eta_c^m(x_a)}(\mathcal{A}) = \frac{1}{|\mathcal{A}|}\sum_{x_a \in \mathcal{A}} (z_c(x_a) - f_c^m(x_a))^2, \tag{2}$$

where $(z_c(x_a) - f_c^m(x_a))^2$ is the squared error of classifier L_m on instance x_a. Since we assume that $f_c^m(x_a)$ is either 0 or 1, it follows that $(z_c(x_a) - f_c^m(x_a))^2 = 0$ if the prediction of L_m is correct, and $= 1$, otherwise. Let x_a be an arbitrary instance in $\mathcal{A}$, and let $r(x_a)$ be the majority vote count of x_a. Also, let us divide the classifiers into two groups. Let *group 1* be $\{L_{m_j}\}_{j=1}^{r(x_a)}$, i.e., the classifiers that contributed to the majority vote, and *group 2* be $\{L_{m_j}\}_{j=r(x_a)+1}^{M}$, i.e., all other classifiers. Since we consider that the errors of the classifiers are independent, it is highly unlikely that majority of the classifiers will make the same mistake. Therefore, we may consider the votes in favor of the majority class to be correct. So, all classifiers in group 1 has correct prediction, and all other classifiers have incorrect predictions. The *combined squared error* (CSE) of the *individual* classifiers in classifying x_a into class c is:

$$\sum_{m=1}^{M}(z_c(x_a) - f_c^m(x_a))^2 = \sum_{j=1}^{r(x_a)}(z_c(x_a) - f_c^{m_j}(x_a))^2 + \sum_{j=r(x_a)+1}^{M}(z_c(x_a) - f_c^{m_j}(x_a))^2$$

$$= 0 + \sum_{j=r(x_a)+1}^{M}(z_c(x_a) - f_c^{m_j}(x_a))^2 \tag{3}$$

Note that CSE is the sum of the squared errors of individual classifiers in the ensemble, not the error of the ensemble itself. Also, note that each component of group 2 in the CSE, i,e,. each $(z_c(x_a) - f_c^{m_j}(x_a))^2$, $j > r(x_a)$ contributes 1 to the sum (since the prediction is wrong). Now we may proceed as follows:

$$M2S(x_a) < M2S(x_b) \Rightarrow r(x_a) < r(x_b) \quad \text{(since the total vote} = M) \tag{4}$$

This implies that the size of group 2 for x_a is larger than that for x_b. Therefore, the CSE in classifying x_a is greater than that of x_b, since each component of group 2 in CSE contributes 1 to the sum. Continuing from eqn (4),

$$\Rightarrow \sum_{j=r(x_a)+1}^{M}(z_c(x_a) - f_c^{m_j}(x_a))^2 > \sum_{j=r(x_b)+1}^{M}(z_c(x_b) - f_c^{m_j}(x_b))^2$$

$$\Rightarrow \sum_{m=1}^{M}(z_c(x_a) - f_c^m(x_a))^2 > \sum_{m=1}^{M}(z_c(x_b) - f_c^m(x_b))^2 \quad \text{(using eqn 3)} \tag{5}$$

Now, according to the Lemma statement, for any pair $(x_a \in \mathcal{A}, x_b \in \mathcal{B})$, $M2S(x_a) < M2S(x_b)$ holds, and hence, inequality (5) holds. Therefore, the mean CSE of set $\mathcal{A}$ must be less than the mean CSE of set $\mathcal{B}$, i.e.,

$$\Rightarrow \frac{1}{|\mathcal{A}|}\sum_{x_a\in\mathcal{A}}\sum_{m=1}^{M}(z_c(x_a) - f_c^m(x_a))^2 > \frac{1}{|\mathcal{B}|}\sum_{x_b\in\mathcal{B}}\sum_{m=1}^{M}(z_c(x_b) - f_c^m(x_b))^2$$

$$\Rightarrow \sum_{m=1}^{M}(\frac{1}{|\mathcal{A}|}\sum_{x_a\in\mathcal{A}}(z_c(x_a) - f_c^m(x_a))^2) > \sum_{m=1}^{M}(\frac{1}{|\mathcal{B}|}\sum_{x_b\in\mathcal{B}}(z_c(x_b) - f_c^m(x_b))^2)$$

$$\Rightarrow \sum_{m=1}^{M}\sigma^2_{\eta_c^m(x_a)}(\mathcal{A}) > \sum_{m=1}^{M}\sigma^2_{\eta_c^m(x_b)}(\mathcal{B}) \quad \text{(using eqn 2)}$$

$$\Rightarrow \sigma^2_{\eta_c^{avg}(x_a)}(\mathcal{A}) > \sigma^2_{\eta_c^{avg}(x_b)}(\mathcal{B}) \quad \text{(using eqn 1)}$$

That is, the ensemble error variance, and hence, the ensemble error (since error variance is proportional to error) on $\mathcal{A}$ is higher than that of $\mathcal{B}$. □

4 Experiments

In this section we describe the datasets, experimental environment, and discuss and analyze the results.

4.1 Data Sets and Experimental Setup

We use two synthetic and two real datasets for evaluation. These are: Synthetic data with only concept-drift **(SynC)**, Synthetic data with concept-drift and novel-class **(SynCN)**, Real data - KDDCup 99 network intrusion detection **(KDD)**, and Real data - Forest cover dataset from UCI repository **(Forest)**. Due to space limitation, we omit the details of the datasets. Details can be found in [4]. We use the following parameter settings, unless mentioned otherwise: i) K (number of pseudopoints per classifier) = 50, ii) q (minimum number of instances required to declare novel class) = 50, iii) L (ensemble size) = 6, iv) S (chunk size) = 2,000. v) MMT (minimum majority threshold) = 0.5.

4.2 Baseline Approach

We use the same baseline techniques that were used to compare with MineClass [4]. Since to the best of our knowledge, there is no technique that can both classify and detect novel class in data streams, a combination of two baseline techniques are used in MineClass: $OLINDDA$ [10], and Weighted Classifier Ensemble (WCE) [2], where the former works as novel class detector, and the latter performs classification. For each chunk, we first detect the novel class instances using $OLINDDA$. All other instances in the chunk are assumed to be in the existing classes, and they are classified using WCE. We use $OLINDDA$ as the novelty detector, since it is a recently proposed algorithm that is shown to have outperformed other novelty detection techniques in data streams [10].

However, $OLINDDA$ assumes that there is only one "normal" class, and all other classes are "novel". So, it is not directly applicable to the multi-class novelty detection problem, where any combination of classes can be considered as the "existing" classes. We propose two alternative solutions. First, we build parallel $OLINDDA$ models, one for each class, which evolve simultaneously. Whenever the instances of a novel class appear, we create a new $OLINDDA$ model for that class. A test instance is declared as novel, if *all the existing class models* identify this instance as novel. We will refer to this baseline method as WCE-OLINDDA_PARALLEL. Second, we initially build an $OLINDDA$ model with all the available classes. Whenever a novel class is found, the class is absorbed into the existing $OLINDDA$ model. Thus, only one "normal" model is maintained throughout the stream. This will be referred to as WCE-OLINDDA_SINGLE. In all experiments, the ensemble size and chunk-size are kept the same for both these techniques. Besides, the same base learner is used for WCE and ActMiner. The parameter settings for OLINDDA are the same as in [4].

In this experiment, we also use WCE-OLINDDA_Parallel and
WCE-OLINDDA_Single for comparison, with some minor changes. In order to see the effects of limited labeled data on WCE-OLINDDA models, we run two different settings for WCE-OLINDDA_Parallel and WCE-OLINDDA_Single. First, we run WCE-OLINDDA_Parallel (WCE-OLINDDA_Single) with *all instances in each chunk labeled*. We denote this setting as WCE-OLINDDA_Parallel-Full (WCE-OLINDDA_Single-Full). Second, we run WCE-OLINDDA_Parallel

(WCE-OLINDDA_Single) with exactly the same instances labeled as were labeled by ActMiner, *plus* any instance identified as novel class by WCE-OLINDDA_Parallel (WCE-OLINDDA_Single). We denote this setting as WCE-OLINDDA_Parallel-Partial (WCE-OLINDDA_Single-Partial). We will henceforth use the acronyms **AM** for ActMiner, $\mathbf{WOP}_f$ for WCE-OLINDDA_Parallel-Full, $\mathbf{WOS}_f$ for WCE-OLINDDA_Single-Full, $\mathbf{WOP}_p$ for WCE-OLINDDA_Parallel-Partial, and $\mathbf{WOS}_p$ for WCE-OLINDDA_Single-Partial.

4.3 Evaluation

Evaluation approach: Let F_n = total novel class instances misclassified as existing class, F_p = total existing class instances misclassified as novel class, F_e = total existing class instances misclassified (other than F_p), N_c = total novel class instances in the stream, N = total instances the stream. We use the following performance metrics to evaluate our technique: $\boldsymbol{M_{new}}$ = % of novel class instances Misclassified as existing class = $\frac{F_n*100}{N_c}$, $\boldsymbol{F_{new}}$ = % of existing class instances Falsely identified as novel class = $\frac{F_p*100}{N-N_c}$, **ERR** = Total misclassification error (%)(including M_{new} and F_{new}) = $\frac{(F_p+F_n+F_e)*100}{N}$. From the definition of the error metrics, it is clear that ERR is not necessarily equal to the sum of M_{new} and F_{new}. Also, let L_p be the percentage of instances in the data stream required to have labels for training.

Evaluation is done as follows: we build the initial models in each method with the first *init_number* labeled chunks with all instances in each chunk labeled. In our experiments, we set *init_number* = 3. From the 4th chunk onward, we evaluate the performances of each method on each data point. We update the models with a new chunk whenever all weakly classified instances (WCIs) in that chunk are labeled.

Results: Figures 1(a_1),1(b_1) show the ERR of each baseline technique and figures 1(a_2),1(b_2) show the percentage of data labeled (L_p) corresponding to each technique on a real (Forest) and a synthetic (SynCN) dataset with decision tree. Corresponding charts for other datasets and k-NN classifier are similar, and omitted due to the space limitation. Figure 1(a_1) shows the ERR of each technique at different stream positions for Forest dataset. The X axis in this chart corresponds to a particular stream position, and the corresponding value at the Y axis represents the ERR upto that position. For example, at X=200, corresponding Y values represent the ERR of a technique on the first 200K instances in the stream. At this position, corresponding Y values (i.e., ERR) of AM, WOP_f, WOP_P, WOS_f and WOS_p are 7.5%, 10.8%, 56.2%, 12.3%, and 63.2%, respectively. The percentage of data required to be labeled (L_P) by each of these techniques for the same dataset (Forest) is shown in figure 1(a_2). For example, at the same X position (X=200), the L_P values for AM, WOP_f, WOP_P, WOS_f and WOS_p are 8.9%, 100%, 12.7%, 100%, and 9%, respectively. Therefore, from the first 200K instances in the stream, AM required only 8.9% instances to have labels, whereas, its nearest competitor (WOP_f) required 100% instances to

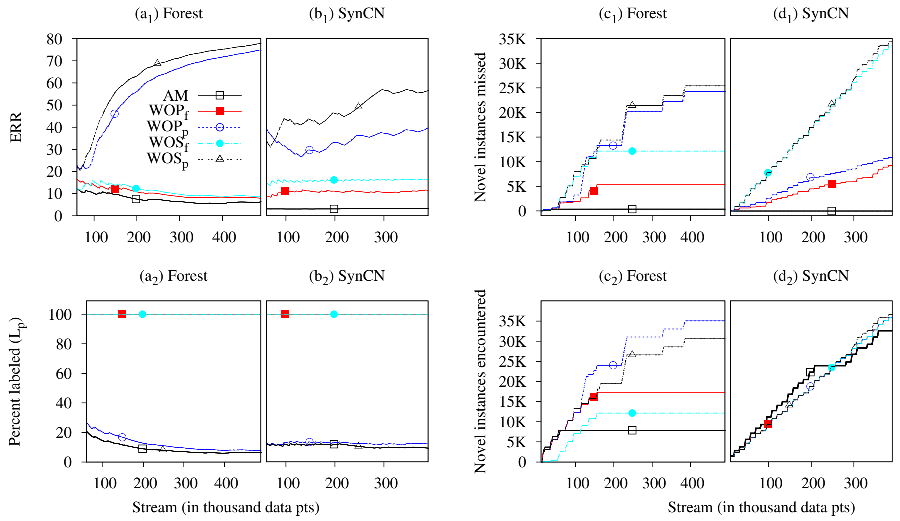

Fig. 1. Overall error (ERR), percentage of data required to be labeled (L_p), total novel instances missed, and encountered by each method

have labels. So, AM, using 11 times less labeled data, achieves lower ERR rates than WOP$_f$. Note that ERR rates of other methods such as WOP$_p$, which uses less than 100% labeled data, are much worse.

Figures 1(c$_1$),1(d$_1$) show the number of novel instances missed (i.e., misclassified as existing class) by each baseline technique, and figures 1(c$_2$),1(d$_2$) report the total number of novel instances encountered by each technique on the same real (Forest) and synthetic (SynCN) datasets with decision tree classifier. For example, in figure 1(c$_1$), for X=200, the Y values represent the total number of novel class instances missed by each technique within the first 200K instances in the stream. The corresponding Y values for AM, WOP$_f$, WOP$_P$, WOS$_f$ and WOS$_p$ are 366, 5,317, 13,269, 12,156 and 14,407, respectively. figure 1(c$_2$) shows the total number of novel instances encountered by each method at different stream positions for the same dataset. Different approaches encounter different amount of novel class instances because the ensemble of classifiers in each approach evolve in different ways. Therefore, a class may be novel for one approach, and may be existing for another approach.

Table 1 shows the summary of the evaluation. The table is split into two parts: the upper part shows the ERR and M_{new} values, and the lower part shows the F_{new} and L_p values. For example, consider the upper part of the table corresponding to the row *KDD* under *Decision tree*. This row shows the ERR and M_{new} rates for each of the baseline techniques on KDD dataset for decision tree classifier. Here AM has the lowest ERR rate, which is 1.2%, compared to 5.8%, 64.0%, 6.7%, and 74.8% ERR rates of WOP$_f$, WOP$_p$, WOS$_f$ and WOS$_p$, respectively. Also, the M_{new} rate of AM is much lower (1.4%) compared to any other baselines. Although WOS$_f$ and WOP$_f$ have lower ERR rates in SynC

Table 1. Performance comparison

Classifier	Dataset	ERR					M_{new}				
		AM	WOP$_f$	WOP$_p$	WOS$_f$	WOS$_p$	AM	WOP$_f$	WOP$_p$	WOS$_f$	WOS$_p$
Decision tree	SynC	13.4	14.1	42.5	**12.8**	42.3	-	-	-	-	-
	SynCN	**0.3**	8.9	38.4	13.9	55.7	**0.0**	26.5	31.0	96.2	96.3
	KDD	**1.2**	5.8	64.0	6.7	74.8	**1.4**	13.2	22.4	96.9	96.1
	Forest	**6.3**	7.9	74.5	8.5	77.4	**4.6**	30.7	69.3	70.1	83.1
k-NN	SynC	**0.0**	2.4	2.4	1.1	1.1	-	-	-	-	-
	SynCN	**0.0**	8.9	17.2	13.9	36.0	**0.0**	26.5	26.3	96.2	98.9
	KDD	**1.1**	4.9	15.3	5.2	63.2	**6.2**	12.9	76.1	96.5	99.1
	Forest	7.1	**4.1**	16.9	4.6	37.8	**15.4**	32.0	28.6	70.1	82.2
Classifier	Dataset	F_{new}					L_p				
		AM	WOP$_f$	WOP$_p$	WOS$_f$	WOS$_p$	AM	WOP$_f$	WOP$_p$	WOS$_f$	WOS$_p$
Decision tree	SynC	**0.0**	2.4	2.4	1.1	1.0	**1.04**	100	3.41	100	2.05
	SynCN	**0.0**	1.6	1.5	0.1	0.1	**9.31**	100	12.10	100	9.31
	KDD	1.1	4.3	4.5	**0.03**	0.03	**3.33**	100	8.82	100	3.34
	Forest	3.0	1.1	1.1	**0.2**	0.2	**6.51**	100	8.08	100	6.56
k-NN	SynC	**0.0**	2.4	2.4	1.1	1.1	**0.0**	100	2.46	100	1.09
	SynCN	**0.0**	1.6	1.7	0.1	0.1	**8.35**	100	12.73	100	8.35
	KDD	0.9	4.4	4.8	**0.03**	0.03	**1.73**	100	7.94	100	1.73
	Forest	1.9	1.1	1.0	**0.2**	0.2	**5.05**	100	6.82	100	5.20

(decision tree), and Forest (k-NN), respectively, they use at least 20 times more labeled data than AM in those datasets, which is reported in the lower right part of the table (under L_p), and their M_{new} rates are much higher than AM. Note that L_p is determined from the WCIs, i.e., what percentage of instances are weakly classified by the ensemble. Therefore, it is different for different datasets. Some readers might find it surprising that active learning outperforms learning with full labels. However, it should be noted that ActMiner outperforms other proposed techniques, not MineClass itself. MineClass, using 100% labeled instances for training, still outperforms ActMiner because ActMiner uses less labeled instance. However, other proposed techniques (like WOP) have too strong requirement about class properties. For example, OLINDDA requires the classes to have convex shape, and assumes similar density of each class of data. On the other hand, ActMiner does not have any such requirement. Therefore, in most real world scenarios, where classes have non-convex shape, and different classes have different data densities, ActMiner performs much better than OLINDDA in detecting novel class, even with much less label information.

Figure 2(left) shows the effect of increasing the minimum majority threshold (MMT) on ERR rate, and figure 2(right) shows the percentage instances labeled for different values of MMT on SynC. For AM, the ERR rate starts decreasing after MMT=0.5. This is because there is no instance for which the M2S (majority to sum) ratio is less than 0.5. So, L_p remains the same (1%) for MMT=0.1 to 0.5 (see figure 2(b)), since the only instances needed to be labeled for these values of MMT are the *F-outlier* instances. However, when MMT=0.6, more instances needed to be labeled (L_p=3.2%) as the M2S ratio for these (3.2-1.0=) 2.2% instances are within the range [0.5,0.6). The overall ERR also reduces since more labeled instances are used for training. Sensitivity of AM to other parameters are similar to MineClass [4], and omitted here due to space limitations.

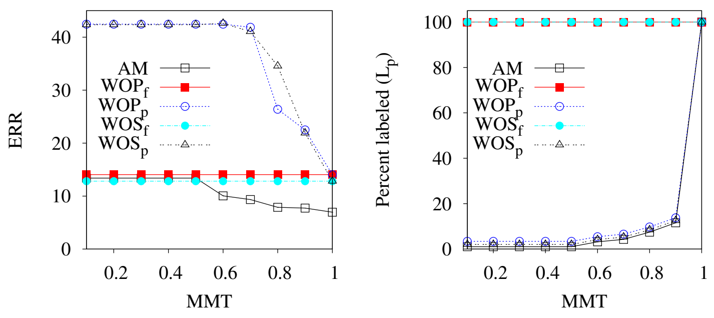

Fig. 2. Effects of increasing the minimum majority threshold (MMT)

Table 2. Running time comparison in all datasets

Dataset	Time(sec)/1K			Time(sec)/1K (including labeling time)		
	AM	WOP$_f$	WOS$_f$	AM	WOP$_f$	WOS$_f$
SynC	0.32	0.41	**0.2**			
SynCN	**1.6**	14.3	3.1			
KDD	1.1	24.0	**0.6**	**34.4**	1,024.0	1,000.6
Forest	0.87	8.5	**0.5**	**66.0**	1,008.5	1,000.5

Table 2 reports the running times of AM and other baseline techniques on different datasets with decision tree. Running times with k-NN also have similar characteristics. Since WOP$_f$ and WOP$_p$ have the same running times, we report only WOP$_f$. The same is true for WOS$_f$ and WOS$_p$. The columns headed by "Time (sec)/1K " show the average running times (train and test) in seconds per 1000 points excluding data labeling time, and the columns headed by "Time (sec)/1K (including labeling time)" show the same including data labeling time. For example, excluding the data labeling time, AM takes 1.1 seconds to process 1K instances on the KDD dataset, whereas WOP$_f$, WOP$_p$ takes 24.0, and 0.5 seconds, respectively. In general, WOP$_f$ is much slower than AM, requiring about C times more runtime than AM. This is because WOP maintains C parallel *OLINDDA* models to detect novel classes. Besides, OLINDDA creates clusters using an internal buffer every time it encounters an instance that is identified as *unknown*, which consumes much of its running time. On the other hand, WOS$_f$ runs slightly faster than AM in three datasets. But this advantage of WOS$_f$ is undermined by its much poorer performance in classification accuracy than AM. If we consider the data labeling time, we get a more compelling picture. We consider the labeling times only for real datasets. Suppose the labeling time for each data point for the real datasets is 1 sec, although in real life, data labeling may require much longer time [12]. Out of each 1000 instances, AM requires only 33, and 65 instances to have labels for the KDD, and Forest datasets, respectively (see table 1 under L_p). Whereas WOP$_f$ and WOS$_f$ require all the 1000 instances to have labels. Therefore, the total running time of AM per 1000 instances including data labeling time is only 3.4% and 6.5% of that of WOP$_f$ and WOS$_f$ for KDD and Forest datasets, respectively. Thus, AM outperforms the baseline techniques both in classification accuracies and running times.

5 Conclusion

Our approach, ActMiner, provides a more complete framework for data stream classification than existing techniques. ActMiner integrates the solutions to four major data stream classification problems: infinite length, concept-drift, concept-evolution, and limited labeled data. Most of the existing techniques address only two or three of these four problems. ActMiner reduces data labeling time and cost by requiring only a few selected instances to be labeled. Even with this limited amount of labeled data, it outperforms state-of-the-art data stream classification techniques that use ten times or more labeled data. In future, we would like to address the dynamic feature set problem and multi-label classification problems in data stream classification.

References

1. Hulten, G., Spencer, L., Domingos, P.: Mining time-changing data streams. In: Proc. KDD '01, pp. 97–106 (2001)
2. Wang, H., Fan, W., Yu, P.S., Han, J.: Mining concept-drifting data streams using ensemble classifiers. In: Proc. KDD '03, pp. 226–235 (2003)
3. Chen, S., Wang, H., Zhou, S., Yu: Stop chasing trends: Discovering high order models in evolving data. In: Proc. ICDE '08, pp. 923–932 (2008)
4. Masud, M.M., Gao, J., Khan, L., Han, J., Thuraisingham, B.M.: Integrating novel class detection with classification for concept-drifting data streams. In: Buntine, W., Grobelnik, M., Mladenić, D., Shawe-Taylor, J. (eds.) ECML PKDD 2009. LNCS, vol. 5782, pp. 79–94. Springer, Heidelberg (2009)
5. Kolter, J.Z., Maloof, M.A.: Using additive expert ensembles to cope with concept drift. In: Proc. ICML '05, Bonn, Germany, pp. 449–456 (2005)
6. Yang, Y., Wu, X., Zhu, X.: Combining proactive and reactive predictions for data streams. In: Proc. KDD '05, pp. 710–715 (2005)
7. Zhu, X., Zhang, P., Lin, X., Shi, Y.: Active learning from data streams. In: Perner, P. (ed.) ICDM 2007. LNCS (LNAI), vol. 4597, pp. 757–762. Springer, Heidelberg (2007)
8. Fan, W., an Huang, Y., Wang, H., Yu, P.S.: Active mining of data streams. In: SDM, pp. 457–461 (2004)
9. Masud, M.M., Gao, J., Khan, L., Han, J., Thuraisingham, B.M.: A practical approach to classify evolving data streams: Training with limited amount of labeled data. In: Proc. ICDM '08, pp. 929–934 (2008)
10. Spinosa, E.J., de Leon, F., de Carvalho, A.P., Gama, J.: Cluster-based novel concept detection in data streams applied to intrusion detection in computer networks. In: Proc. SAC '08, pp. 976–980 (2008)
11. Tumer, K., Ghosh, J.: Error correlation and error reduction in ensemble classifiers. Connection Science 8(304), 385–403 (1996)
12. van Huyssteen, G.B., Puttkammer, M.J., Pilon, S., Groenewald, H.J.: Using machine learning to annotate data for nlp tasks semi-automatically. In: Proc. Computer-Aided Language Processing, CALP'07 (2007)