Integrating Novel Class Detection with Classification for Concept-Drifting Data Streams

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

[1] University of Texas, Dallas
[2] University of Illinois, Urbana Champaign
mehedy@utdallas.edu, jinggao3@uiuc.edu,
lkhan@utdallas.edu, hanj@cs.uiuc.edu, bhavani.thuraisingham@utdallas.edu

Abstract. In a typical data stream classification task, it is assumed that the total number of classes are fixed. This assumption may not be valid in a real streaming environment, where new classes may evolve. Traditional data stream classification techniques are not capable of recognizing novel class instances until the appearance of the novel class is manually identified, and labeled instances of that class are presented to the learning algorithm for training. The problem becomes more challenging in the presence of concept-drift, when the underlying data distribution changes over time. We propose a novel and efficient technique that can automatically detect the emergence of a novel class in the presence of concept-drift by quantifying cohesion among unlabeled test instances, and separation of the test instances from training instances. Our approach is non-parametric, meaning, it does not assume any underlying distributions of data. Comparison with the state-of-the-art stream classification techniques prove the superiority of our approach.

1 Introduction

It is a major challenge to data mining community to mine the ever-growing streaming data. There are three major problems related to stream data classification. First, it is impractical to store and use all the historical data for training, since it would require infinite storage and running time. Second, there may be concept-drift in the data, meaning, the underlying concept of the data may change over time. Third, novel classes may evolve in the stream. There are many existing solutions in literature that solve the first two problems, such as single model incremental learning algorithms [1,2,11], and ensemble classifiers [3,5,9]. However, most of the existing techniques are not capable of detecting novel classes in the stream. On the other hand, our approach can handle both concept-drift, and detect novel classes at the same time.

Traditional classifiers can only correctly classify instances of those classes with which they have been trained. When a new class appears in the stream, all instances belonging to that class will be misclassified until the new class

W. Buntine et al. (Eds.): ECML PKDD 2009, Part II, LNAI 5782, pp. 79–94, 2009.
© Springer-Verlag Berlin Heidelberg 2009

has been manually identified by some experts and a new model is trained with the labeled instances of that class. Our approach provides a solution to this problem by incorporating a novel class detector within a traditional classifier so that the emergence of a novel class can be identified without any manual intervention. The proposed novel class detection technique can benefit many applications in various domains, such as network intrusion detection and credit card fraud detection. For example, in the problem of intrusion detection, when a new kind of intrusion occurs, we should not only be able to detect that it is an intrusion, but also that it is a new kind of intrusion. With the intrusion type information, human experts would be able to analyze the intrusion more intensely, find a cure, set an alarm in advance and make the system more secure.

We propose an innovative approach to detect novel classes. It is different from traditional novelty (or anomaly/outlier) detection techniques in several ways. First, traditional novelty detection techniques [4,6,10] work by assuming or building a model of normal data, and simply identifying data points as outliers/anomalies that deviate from the "normal" points. But our goal is not only to detect whether a single data point deviates from the normality, but also to discover whether a group of outliers have any strong bond among themselves. Second, traditional novelty detectors can be considered as a "one-classs" model, which simply distinguish between normal and anomalous data, but cannot distinguish between two different kinds of anomalies. But our model is a "multi-class" model, meaning, it can distinguish among different classes of data and at the same time can detect presence of a novel class data, which is a unique combination of a traditional classifier with a novelty detector.

Our technique handles concept-drift by adapting an ensemble classification approach, which maintains an ensemble of M classifiers for classifying unlabeled data. The data stream is divided into equal-sized chunks, so that each chunk can be accommodated in memory and processed online. We train a classification model from each chunk as soon as it is labeled. The newly trained model replaces one of the existing models in the ensemble, if necessary. Thus, the ensemble evolves, reflecting the most up-to-date concept in the stream.

The central concept of our novel class detection technique is that each class must have an important property: the data points belonging to the same class should be closer to each other (cohesion) and should be far apart from the data points belonging to other classes (separation). Every time a new data chunk appears, we first detect the test instances that are *well-separated* from the training data (i.e. outliers). Then filtering is applied to remove the outliers that possibly appear as a result of concept-drift. Finally, if we find strong cohesion among those filtered outliers, we declare a novel class. When the true labels of the novel class(es) arrive and a new model is trained with the labeled instances, the existing ensemble is updated with that model. Therefore, the ensemble of models is continuously enriched with new classes.

We have several contributions. First, we provide a detailed understanding of the characteristic of a novel class, and propose a new technique that can detect novel classes in the presence of concept-drift in data streams. Second, we

establish a framework for incorporating novel class detection mechanism into a traditional classifier. Finally, we apply our technique on both synthetic and real-world data and obtain much better results than state-of the art stream classification algorithms.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 provides an overview of our approach and Section 4 discusses our approach in detail. Section 5 then describes the datasets and experimental evaluation of our technique. Section 6 concludes with discussion and suggestions for future work.

2 Related Work

Our work is related to both stream classification and novelty detection. There have been many works in stream data classification. There are two main approaches - single model classification, and ensemble classification. Some single-model techniques have been proposed to accommodate concept drift [1,2,11]. However, Our technique follows the ensemble approach. Several ensemble techniques for stream data mining have been proposed [3,5,9]. These ensemble approaches require simple operations to update the current concept, and they are found to be robust in handling concept-drift. Although these techniques can efficiently handle concept-drift, none of them can detect novel classes in the data stream. On the other hand, our technique is not only capable of handling concept-drift, but also able to detect novel classes in data streams. In this light, our technique is also related to novelty detection techniques.

A comprehensive study on novelty detection has been discussed in [4]. The authors categorize novelty detection techniques into two categories: statistical and neural network based. Our technique is related to the statistical approach. Statistical approaches are of two types: parametric, and non-parametric. Parametric approaches assume that data distributions are known (e.g. Gaussian), and try to estimate the parameters (e.g. mean and variance) of the distribution. If any test data falls outside the normal parameters of the model, it is declared as novel [6]. Our technique is a non-parametric approach. Non-parametric approaches like parzen window method [10] estimate the density of training data and reject patterns whose density is beyond a certain threshold. K-nearest neighbor (K-NN) based approaches for novelty detection are also non-parametric [12]. All of these techniques for novelty detection only consider whether a test instance is sufficiently close (or far) from the training data based on some appropriate metric (e.g., distance, density etc.). Our approach is different from these approaches in that we not only consider separation from normal data but also consider cohesion among the outliers. Besides, our model assimilates a novel class into the existing model, which enables it to distinguish future instances of that class from other classes. On the other hand, novelty detection techniques just remember the "normal" trend, and do not care about the similarities or dissimilarities among the anomalous instances.

A recent work in data stream mining domain [7] describes a clustering approach that can detect both concept-drift and novel class. This approach

assumes that there is only one 'normal' class and all other classes are novel. Thus, it may not work well if more than one classes are to be considered as 'normal' or 'non-novel', but our approach can handle any number of existing classes. This makes our approach more effective in detecting novel classes than [7], which is justified by the experimental results.

3 Overview

Algorithm 1 outlines a summary of our technique. The data stream is divided into equal sized chunks. The latest chunk, which is unlabeled, is provided to the algorithm as input. At first it detects if there is any novel class in the chunk (line 1). The term "novel class" will be defined shortly. If a novel class is found, we detect the instances that belong to the class(es) (line 2). Then we use the ensemble $L = \{L_1, ..., L_M\}$ to classify the instances that do not belong to the novel class(es). When the data chunk becomes labeled, a new classifier L' trained using the chunk. Then the existing ensemble is updated by choosing the best M classifiers from the $M + 1$ classifiers $L \cup \{L'\}$ based on their accuracies on the latest labeled data chunk. Our algorithm will be mentioned henceforth as

Algorithm 1. MineClass

Input: D_n: the latest data chunk
L: Current ensemble of best M classifiers
Output: Updated ensemble L
1: found ← **DetectNovelClass**(D_n,L) (algorithm 2, section 4.3)
2: **if** found **then** Y ← Novel_instances(D_n), $X \leftarrow D_n - Y$ **else** $X \leftarrow D_n$
3: **for** each instance $x \in X$ **do** **Classify**(L,x)
4: /*Assuming that D_n is now labeled*/
5: L' ← **Train-and-create-inventory**(D_n) (section 4.1)
6: L ← **Update**(L,L',D_n)

"MineClass", which stands for Mining novel Classes in data streams. MineClass should be applicable to any base learner. The only operation that is specific to a learning algorithm is *Train-and-create-inventory*. We will illustrate this operation for two base learners.

3.1 Classifiers Used

We apply our novelty detection technique on two different classifiers: decision tree, and K-NN. We keep M classification models in the ensemble. For decision tree classifier, each model is a decision tree. For K-NN, each model is usually the set of training data itself. However, storing all the raw training data is memory-inefficient and using them to classify unlabeled data is time-inefficient. We reduce both the time and memory requirement by building K clusters with the training data, saving the cluster summaries as classification models, and

discarding the raw data. This process is explained in details in [5]. The cluster summaries are mentioned henceforth as "pseudopoint"s. Since we store and use only K pseudopoints, both the time and memory requirements become functions of K (a constant number). The clustering approach followed here is a constraint-based K-means clustering where the constraint is to minimize cluster impurity while minimizing the intra-cluster dispersion. A cluster is considered pure if it contains instances from only one class. The summary of each cluster consists of the centroid, and the frequencies of data points of each class in the cluster. Classification is done by finding the nearest cluster centroid from the test point, and assigning the class, that has the highest frequency, to the test point.

3.2 Assumptions

We begin with the definition of "novel" 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 the instances of class c. Otherwise, c is a novel class.*

We assume that any class has the following essential property:

Property 1. *A data point should be closer to the data points of its own class (cohesion) and farther apart from the data points of other classes (separation).*

Our main assumption is that the instances belonging to a class c is generated by a an underlying generative model θ_c, and the instances in each class are independently identically distributed. With this assumption, we can reasonably argue that the instances which are close together are supposed to be generated

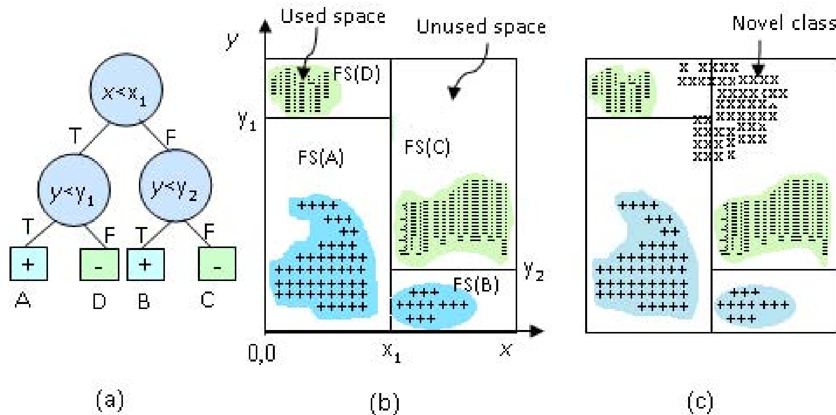

Fig. 1. (a) A decision tree and (b) corresponding feature space partitioning. FS(X) denotes the feature space defined by a leaf node X. The shaded areas show the used spaces of each partition. (c) A Novel class (denoted by x) arrives in the unused space.

by the same model, i.e., belong to the same class. We now show the basic idea of novel class detection using decision tree in figure 1. We introduce the notion of ***used space*** to denote a feature space occupied by any instance, and ***unused space*** to denote a feature space unused by an instance. According to property 1(separation), a novel class must arrive in the unused spaces. Besides, there must be strong cohesion (e.g. closeness) among the instances of the novel class. Thus, the two basic principles followed by our approach are: keeping track of the used spaces of each leaf node in a decision tree, and finding strong cohesion among the test instances that fall into the unused spaces.

4 Novel Class Detection

We follow two basic steps for novel class detection. First, the classifier is trained such that an inventory of the used spaces (described in section 3.2) is created and saved. This is done by clustering and and saving the cluster summary as "pseudopoint" (to be explained shortly). Secondly, these pseudopoints are used to detect outliers in the test data, and declare a novel class if there is strong cohesion among the outliers.

4.1 Saving the Inventory of Used Spaces During Training

The general idea of creating the inventory is to cluster the training data, and save the cluster centroids and other useful information as pseudopoints. These pseudopoints keep track of the use spaces. The way how this clustering is done may be specific to each base learner. For example, for decision tree, clustering is done at each leaf node of the tree, since we need to keep track of the used spaces for each leaf node separately. For the K-NN classifier discussed in section 3.1, already existing pseudopoints are utilized to store the inventory.

It should be noted here that K-means clustering appears to be the best choice for saving the decision boundary and computing the outliers. Density-based clustering could also be used to detect outliers but it has several problems. First, we would have to save all the raw data points at the leaf nodes to apply the clustering. Second, the clustering process would take quadratic time, compared to linear time for K-means. Finally, we would have to run the clustering algorithm for every data chunk to be tested. However, the choice of parameter K in K-means algorithm has some impact on the overall outcome, which is discussed in the experimental results.

Clustering: We build total K clusters per chunk. For K-NN, we utilize the existing clusters that were created globally using the approach discussed in section 3.1. For decision tree, clustering is done locally at each leaf node as follows. Suppose S is the chunk-size. During decision tree training, when we reach a leaf node l_i, we build $k_i = (t_i/S) * K$ clusters in that leaf, where t_i denotes the number of training instances that ended up in leaf node l_i.

Storing the cluster summary information: For each cluster, we store the following summary information in memory: i) ***Weight***, w: Defined as the

total number of points in the cluster. ii) ***Centroid***, ζ. iii) ***Radius***, $\mathcal{R}$: Defined as the maximum distance between the centroid and the data points belonging to the cluster. iv) ***Mean distance***, μ_d: The mean distance from each point to the cluster centroid. The cluster summary of a cluster $\mathcal{H}_i$ will be referred to henceforth as a "pseudopoint" ψ_i. So, $w(\psi_i)$ denotes the weight of pseudopoint ψ_i. After computing the cluster summaries, the raw data are discarded. Let Ψ_j be the set of all pseudopoints stored in memory for a classifier L_j.

4.2 Outlier Detection and Filtering

Each pseudopoint ψ_i corresponds to a hypersphere in the feature space having center $\zeta(\psi_i)$ and radius $\mathcal{R}(\psi_i)$. Thus, the pseudopoints 'memorize' the used spaces. Let us denote the portion of feature space covered by a pseudopoint ψ_i as the "region" of ψ_i or $RE(\psi_i)$. So, the union of the regions covered by all the pseudopoints is the union of all the used spaces, which forms a decision boundary $\mathcal{B}(L_j) = \cup_{\psi_i \in \Psi_j} RE(\psi_i)$, for a classifier L_j. Now, we are ready to define outliers.

Definition 2 (Routlier). *Let x be a test point and ψ_{min} be the pseudopoint whose centroid is nearest to x. Then x is an Routlier (i.e., raw outlier) if it is outside $RE(\psi_{min})$, i.e., its distance from $\zeta(\psi_{min})$ is greater than $\mathcal{R}(\psi_{min})$.*

In other words, any point x outside the decision boundary $\mathcal{B}(L_j)$ is an *Routlier* for the classifier L_j. For K-NN, *Routlier*s are detected globally by testing x against all the psuedopoints. For decision tree, x is tested against only the psueodpoints stored at the leaf node where x belongs.

Filtering: According to definition 2, a test instance may be erroneously considered as an *Routlier* because of one or more of the following reasons: i) The test instance belongs to an existing class but it is a noise. ii) There has been a concept-drift and as a result, the decision boundary of an existing class has been shifted. iii) The decision tree has been trained with insufficient data. So, the predicted decision boundary is not the same as the actual one.

Due to these reasons, the outliers are filtered to ensure that any outlier that belongs to the existing classes does not end up in being declared as a new class instance. The filtering is done as follows: if a test instance is an *Routlier* to *all* the classifiers in the ensemble, then it is considered as a filtered outlier. All other *Routlier*s are filtered out.

Definition 3 (Foutlier). *A test instance is an Foutlier (i.e., filtered outlier) if it is an Routlier to all the classifiers L_i in the ensemble L.*

Intiuitively, being an *Foutlier* is a necessary condition for being in a new class. Because, suppose an instance x is not an *Routlier* to some classifier L_i in the ensemble. Then x must be inside the decision boundary $\mathcal{B}(L_i)$. So, it violates property 1 (separation), and therefore, it cannot belong to a new class. Although being an *Foutlier* is a necessary condition, it is not sufficient for being in a new class, since it does not guarantee the property 1 (cohesion). So, we proceed to the next step to verify whether the *Foutlier*s satisfy both cohesion and separation.

4.3 Detecting Novel Class

We perform several computations on the *Foutlier*s to detect the arrival of a new class. First, we discuss the general concepts of these computations and later we describe how these computations are carried out efficiently. For every *Foutlier*, we define a λ_c-neighborhood as follows:

Definition 4 (λ_c-neighborhood). *The λ_c-neighborhood of an Foutlier x is the set of $\mathcal{N}$-nearest neighbors of x belonging to class c.*

Here $\mathcal{N}$ is a user defined parameter. For brevity, we denote the λ_c-neighborhood of an *Foutlier* x as $\lambda_c(x)$. Thus, $\lambda_+(x)$ of an *Foutlier* x is the set of $\mathcal{N}$ instances of class c_+, that are closest to the outlier x. Similarly, $\lambda_o(x)$ refers to the set of $\mathcal{N}$ *Foutlier*s that are closest to x. This is illustrated in figure 2, where the *Foutlier*s are shown as black dots, and the instances of class c_+ and class c_- are shown with the corresponding symbols. $\lambda_+(x)$ of the *Foutlier* x is the set of $\mathcal{N}$ (= 3) instances belonging to class c_+ that are nearest to x (inside the circle), and so on. Next, we define the $\mathcal{N}$-neighborhood silhouette coefficient, ($\mathcal{N}$-NSC).

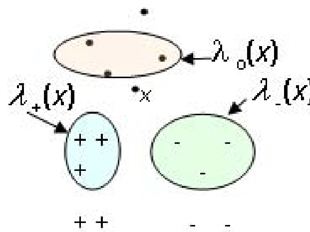

Fig. 2. λ_c-neighborhood with $\mathcal{N}$=3

Definition 5 ($\mathcal{N}$-NSC). *Let $a(x)$ be the average distance from an Foutlier x to the instances in $\lambda_o(x)$, and $b_c(x)$ be the average distance from x to the instances in $\lambda_c(x)$ (where c is an existing class). Let $b_{min}(x)$ be the minimum among all $b_c(x)$. Then $\mathcal{N}$-NSC of x is given by:*

$$\mathcal{N}\text{-}NSC(x) = \frac{b_{min}(x) - a(x)}{max(b_{min}(x), a(x))} \tag{1}$$

According to the definition, the value of $\mathcal{N}$-NSC is between -1 and +1. It is actually a unified measure of cohesion and separation. A negative value indicates that x is closer to the other classes (less separation) and farther away from its own class (less cohesion). We declare a *new class* if there are at least N' ($> \mathcal{N}$) *Foutlier*s, whose $\mathcal{N}$-NSC is positive. In fact, we prove that this is a *necessary and sufficient condition* for a new class. This proof is omitted here due to space limitation, but can be obtained from [8].

It should be noted that the larger the value of $\mathcal{N}$, the greater the confidence with which we can decide whether a novel class has arrived. However, if $\mathcal{N}$ is

too large, then we may also fail to detect a new class if the total number of instances belonging to the novel class in the corresponding data chunk is $\leq \mathcal{N}$. We experimentally find an optimal value of $\mathcal{N}$, which is explained in section 5.

Computing the set of novel class instances: Once we detect the presence of a novel class, the next step is to find those instances, and separate them from the existing class data. According to the *necessary and sufficient condition*, a set of $Foutlier$ instances belong to a novel class if following three conditions satisfy: i) all the $Foutliers$ in the set have positive $\mathcal{N}$-NSC, ii) all the $Foutliers$ in the set have $\lambda_o(x)$ within the set, and iii) cardinality of the set $\geq \mathcal{N}$. Let $\mathcal{G}$ be such a set. Note that finding the exact set $\mathcal{G}$ is computationally expensive, so we follow an approximation. Let $\mathcal{G}'$ be the set of all $Foutliers$ that have positive $\mathcal{N}$-NSC. If $|\mathcal{G}'| \geq \mathcal{N}$, then $\mathcal{G}'$ is an approximation of $\mathcal{G}$. It is possible that some of the data points in $\mathcal{G}'$ may not actually be a novel class instance or vice versa. However, in our experiments, we found that this approximation works well.

Speeding up the computation: Computing $\mathcal{N}$-NSC for every $Foutlier$ instance x takes quadratic time in the number of $Foutlier$s. In order to make the computation faster, we also create K_o pseudopoints from $Foutlier$s using K-means clustering and perform the computations on the pseudopoints (referred to as $Fpseudopoint$s), where $K_o = (N_o/S) * K$. Here S is the chunk size and N_o is the number of $Foutlier$s. Thus, the time complexity to compute the $\mathcal{N}$-NSC of all of the $Fpseudopoint$s is $O(K_o * (K_o + K))$, which is constant, since both K_o and K are independent of the input size. Note that $\mathcal{N}$-NSC of a $Fpseudopoint$ is actually an approximate average of the $\mathcal{N}$-NSC of each $Foutlier$ in that $Fpseudopoint$. By using this approximation, although we gain speed, we also lose some precision. However, this drop in precision is negligible when we keep sufficient number of pseudopoints, as shown in the exprimental results. The novel class detection process is summarized in algorithm 2 (DetectNovelClass).

This algorithm can detect one or more novel classes concurrently (i.e., in the same chunk) as long as each novel class follows property 1 and contains at least $\mathcal{N}$ instances. This is true even if the class distributions are skewed. However, if more than one such novel classes appear concurrently, our algorithm will identify the instances belonging to those classes as novel, without imposing any distinction among dissimilar novel class instances (i.e., it will treat them simply as "novel"). But the distinction will be learned by our model as soon those instances are labeled, and a classifier is trained with them.

Time complexity: Lines 1-3 of algorithm 2 requires $O(KSL)$ time where S is the chunk size. Line 4 (clustering) requires $O(KS)$ time, and the last for loop (5-10) requires $O(K^2L)$ time. Thus, the overall time complexity of algorithm 2 is $O(KS + KSL + K^2L) = O(K(S + SL + KL))$. Assuming that $S >> KL$, the complexity becomes $O(KS)$, which is linear in S. Thus, the overall time complexity (per chunk) of MineClass algorithm (algorithm 1) is $O(KS+f_c(LS)+f_t(S))$, where $f_c(n)$ is the time required to classify n instances and $f_t(n)$ is the time required to train a classifier with n training instances.

Algorithm 2. DetectNovelClass(D,L)

Input: D: An unlabeled data chunk
L: Current ensemble of best M classifiers
Output: true, if novel class is found; false, otherwise

1: **for** each instance $x \in D$ **do**
2: **if** x is an *Routlier* to all classifiers $L_i \in L$
then $FList \leftarrow FList \cup \{x\}$ /* x is an $Foutlier$*/
3: **end for**
4: Make K_o=($K * |FList|/|D|$) clusters with the instances in $FList$ using K-means clustering, and create *Fpseudopoints*
5: **for** each classifier $L_i \in L$ **do**
6: Compute $\mathcal{N}$-NSC(ψ_j) for each *Fpseudopoint* ψ_j
7: $\Psi_p \leftarrow$ the set of *Fpseudopoints* having positive $\mathcal{N}$-NSC(.).
8: $w(\Psi_p) \leftarrow$ sum of $w(.)$ of all *Fpseudopoints* in Ψ_p.
9: **if** $w(\Psi_p) > \mathcal{N}$ **then** NewClassVote++
10: **end for**
11: **return** NewClassVote > M - $NewClassVote$ /*Majority voting*/

Impact of evolving class labels on ensemble classification: As the reader might have realized already, arrival of novel classes in the stream causes the classifiers in the ensemble to have different sets of class labels. For example, suppose an older (earlier) classifier L_i in the ensemble has been trained with classes c_0 and c_1, and a newer (later) classifier L_j has been trained with classes c_1, and c_2, where c_2 is a new class that appeared after L_i had been trained. This puts a negative effect on voting decision, since the older classifier mis-classifies instances of c_2. So, rather than counting votes from each classifier, we selectively count their votes as follows: if a newer classifier L_j classifies a test instance x as class c, but an older classifier L_i does not have the class label c in its model, then the vote of L_i will be ignored if x is found to be an outlier for L_i. An opposite scenario occurs when the oldest classifier L_i is trained with some class c', but none of the later classifiers are trained with that class. This means class c' has been outdated, and, in that case, we remove L_i from the ensemble. In this way we ensure that older classifiers have less impact in the voting process. If class c' later re-appears in the stream, it will be automatically detected again as a novel class (see definition 1).

5 Experiments

We evaluate our proposed method on a number of synthetic and real datasets, but due to space limitations, we report results on four datasets.

5.1 Data Sets

Specific details of the data sets can be obtained from [8].

Synthetic data generation: There are two types of synthetic data: synthetic data with *only concept-drift (SynC)* and synthetic data with *concept-drift and novel-class (SynCN)*. SynC is generated using moving hyperplane, which contains 2 classes and 10 numeric attributes. SynCN is generated using Gaussian distribution, which contains 10 classes and 20 numeric attributes.

Real datasets: The two real datasets used in the experiments are the 10% version of the *KDDCup 99 network intrusion detection*, and *Forest Cover* dataset from UCI repository. We have used the 10% version of the KDDcup dataset, where novel classes appear more frequently than the full version, hence it is more challenging. KDDcup dataset contains around 490,000 instances, 23 classes, and 34 numeric attributes. Forest Cover dataset contains 7 classes, 54 attributes and around 581,000 instances. We arrange the Forest Cover dataset so that in any chunk at most 3 and at least 2 classes co-occur, and new classes appear randomly. All datasets are normalized to have attribute values within [0,1].

5.2 Experimental Setup

We implement our algorithm in Java. The code for decision tree has been adapted from the Weka machine learning open source repository (`http://www.cs.waikato.ac.nz/ml/weka/`). The experiments were run on an Intel P-IV machine with 2GB memory and 3GHz dual processor CPU. Our parameter settings are as follows, unless mentioned otherwise: i) K (number of pseudopoints per chunk) = 50, ii) $\mathcal{N} = 50$, iii) M (ensemble size) = 6, iv) Chunk-size = 1,000 for synthetic datasets, and 4,000 for real datasets. These values of parameters are tuned to achieve an overall satisfactory performance.

Baseline method: To the best of our knowledge, there is no approach that can classify data streams *and* detect novel class. So, we compare MineClass with a combination of two baseline techniques: $OLINDDA$ [7], and Weighted Classifier Ensemble (WCE) [9], 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 [7].

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 MC. The parameter settings for $OLINDDA$ are: i) number of data points per cluster (N_{excl}) = 15, ii) least number of normal instances needed to update the existing model = 100, iii) least number of instances needed to build the initial model = 30. These parameters are chosen either according to the default values used in [7] or by trial and error to get an overall satisfactory performance. *We will henceforth use the acronyms **MC for MineClass, W-OP for WCE-OLINDDA_PARALLEL** and **W-OS for WCE-OLINDDA_SINGLE**.*

5.3 Performance Study

Evaluation approach: We use the following performance metrics for evaluation: M_{new} = % of novel class instances Misclassified as existing class, F_{new} = % of existing class instances Falsely identified as novel class, ERR = Total misclassification error (%)(including M_{new} and F_{new}). We build the initial models in each method with the first M chunks. From the M+1st chunk onward, we first evaluate the performances of each method on that chunk, then use that chunk to update the existing model. The performance metrics for each chunk for each method are saved and averaged for producing the summary result.

Results: Figures 3(a)-(d) show the ERR for decision tree classifier of each approach up to a certain point in the stream in different datasets. K-NN classifier also has similar results. For example, at X axis = 100, the Y values show the average ERR of each approach from the beginning of the stream to chunk 100. At this point, the ERR of MC, W-OP, and W-OS are 1.7%, 11.6% and 8.7%, respectively, for the KDD dataset (figure 3(c)). The arrival of novel a class in each dataset is marked by a cross (x) on the top border in each graph at the corresponding chunk. For example, on the SynCN dataset (figure 3(a)), W-OP and W-OS misses most of the novel class instances, which results in the spikes in their curves at the respective chunks (e.g. at chunks 12, 24, 37 etc.). W-OS misses almost 99% of the novel class instances. Similar spikes are observed for both W-OP and W-OS at the chunks where novel classes appear for KDD and Forest Cover datasets. For example, many novel classes appear between chunks 9-14 in KDD, most of which are missed by both W-OP and W-OS. Note that there is no novel class for SynC dataset. MC correctly detects most of these novel classes. Thus, MC outperforms both W-OP and W-OS in all datasets.

Table 1 summarizes the error metrics for each of the techniques in each dataset for decision tree, and K-NN. The columns headed by ERR, M_{new} and F_{new} report the average of the corresponding metric on an entire dataset. For example, while using decision tree in the SynC dataset, MC, W-OP and W-OS have almost the same ERR, which are 11.6%, 13.0%, and 12.5%, respectively. This is because SynC simulates only concept-drift, and both MC and WCE handle concept-drift in a similar manner. In SynCN dataset with decision tree, MC, W-OP, and W-OS have 0%, 89.4%, and 99.7% M_{new}, respectively. Thus, W-OS misses almost all of the novel class instances, whereas W-OP detects only 11% of them. MC correctly

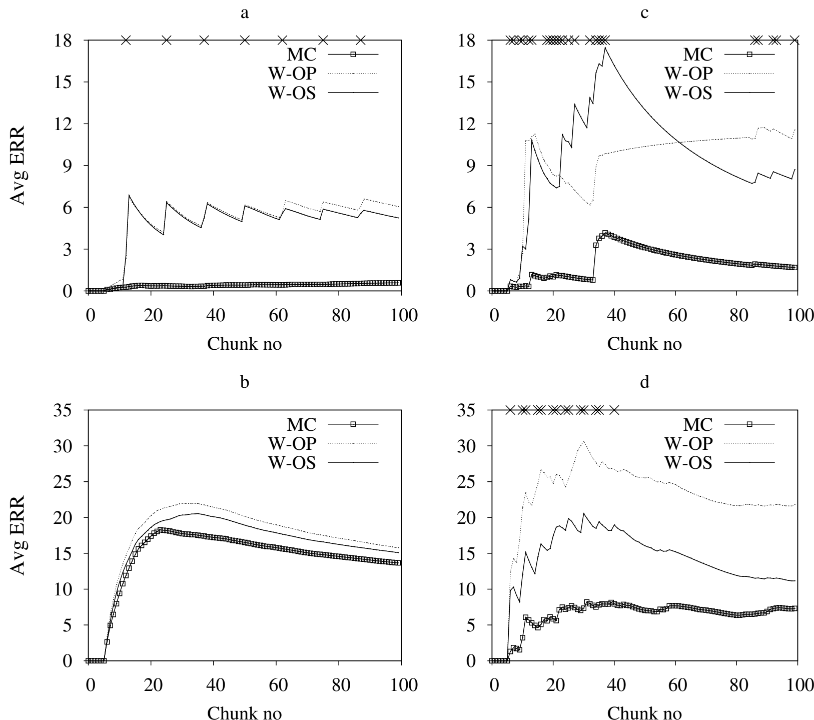

Fig. 3. Error comparison on (a) SynCN, (b) SynC, (c) KDD and (d) Forest Cover

detects all of the novel class instances. It is interesting that all approaches have lower error rates in SynCN than SynC. This is because SynCN is generated using Gaussian distribution, which is naturally easier for the classifiers to learn. W-OS miss-predicts almost all of the novel class instances in all datasets. The comparatively better ERR rate for W-OS over W-OP can be attributed to the lower false positive rate of W-OS, which occurs since almost all instances are identified as "normal" by W-OS. Again, the overall error (ERR) of MC is much lower than other methods in all datasets and for all classifiers. K-NN also has similar results for all datasets.

Figures 4(a)-(d) illustrate how the error rates of MC change for different parameter settings on KDD dataset and decision tree classifier. These parameters have similar effects on other datasets, and K-NN classifier. Figure 4(a) shows the effect of chunk size on ERR, F_{new}, and M_{new} rates for default values of other parameters. M_{new} reduces when chunk size is increased. This is desirable, because larger chunks reduce the risk of missing a novel class. But F_{new} rate slightly increases since the risk of identifying an existing class instance as novel also rises a little. These changes stabilizes from chunk size 4,000 (for Synthetic dataset, it is 1,000). That is why we use these values in our experiments. Figure 4(b) shows the effect of number of clusters (K) on error. Increasing K generally

Table 1. Performance comparison

Classifier	Dataset	ERR			M_{new}			F_{new}		
		MC	W-OP	W-OS	MC	W-OP	W-OS	MC	W-OP	W-OS
Decision tree	SynC	**11.6**	13.0	12.5	0.0	0.0	0.0	**0.0**	1.0	0.6
	SynCN	**0.6**	6.1	5.2	**0.0**	89.4	99.7	0.0	0.6	0.0
	KDD	**1.7**	11.6	8.7	**0.7**	26.7	99.4	1.5	7.0	**0.0**
	Forest Cover	**7.3**	21.8	8.7	**9.8**	18.5	99.4	1.7	15.0	**0.0**
K-NN	SynC	**11.7**	13.1	12.6	0.0	0.0	0.0	**0.0**	1.0	0.6
	SynCN	**0.8**	5.8	5.6	**0**	90.1	99.7	0.9	0.6	**0.0**
	KDD	**2.3**	10.0	7.0	**2.7**	29.0	99.4	2.2	7.1	**0.0**
	Forest Cover	**5.4**	19.2	8.9	**1.0**	18.5	94.0	4.5	15.0	**0.3**

reduces error rates, because outliers are more correctly detected, and as a result, M_{new} rate decreases. However, F_{new} rate also starts increasing slowly, since more test instances are becoming outliers (although they are not). The combined effect is that overall error keeps decreasing up to a certain value (e.g. K=50), and then becomes almost flat. This is why we use K=50 in our experiments. Figure 4(c) shows the effect of ensemble size (M) on error rates. We observe that the error rates decrease up to a certain size (=6), and become stable since then. This is because when M is increased from a low value (e.g., 2), classification error naturally decreases up to a certain point because of the reduction of error variance [9]. Figure 4(d) shows the effect of $\mathcal{N}$ on error rates. The x-axis in this chart is drawn in a logarithmic scale. Naturally, increasing $\mathcal{N}$ up to a certain point (e.g. 20) helps reducing error, since we know that a higher value of $\mathcal{N}$ gives us a greater confidence in declaring a new class (see section 4.3). But a too large value of $\mathcal{N}$ increases M_{new} and ERR rates, since a new class is missed by the algorithm if it has less than $\mathcal{N}$ instances in a data chunk. We have found that any value between 20 to 100 is the best choice for $\mathcal{N}$.

Running time: Table 2 compares the running times of MC, W-OP, and W-OS on each dataset for decision tree. K-NN also shows similar performances. The columns headed by "Time (sec)/chunk " show the average running times (train and test) in seconds per chunk, the columns headed by "Points/sec" show how many points have been processed (train and test) per second on average, and the columns headed by "speed gain" shows the ratio of the speed of MC to that of W-OP and W-OS, respectively. For example, MC is 2,095, and 105 times faster than W-OP on KDD dataset, and Forest Cover dataset, respectively. Also, MC is 203 and 27 times faster than W-OP and W-OS, respectively, on the SynCN dataset. W-OP and W-OS are slower on SynCN than on SynC dataset because SynCN dataset has more attributes (20 vs 10) and classes (10 vs 2). W-OP is relatively slower than W-OS since W-OP maintains C parallel models, where C is the number of existing classes, whereas W-OS maintains only one model. Both W-OP and W-OS are relatively faster on Forest Cover than KDD since Forest Cover has less number of classes, and relatively less evolution than KDD. The main reason for this extremely slow processing of W-OP and W-OS is that the number of clusters for each OLINDDA model keeps increasing linearly with

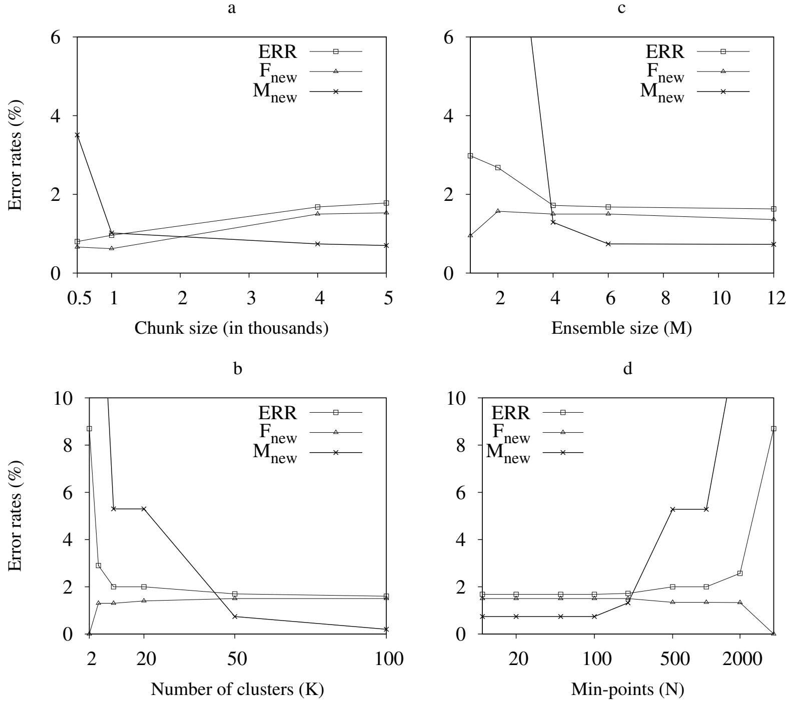

Fig. 4. Sensitivity to different parameters

Table 2. Running time comparison in all datasets

Dataset	Time(sec)/chunk			Points/sec			Speed gain	
	MC	W-OP	W-OS	MC	W-OP	W-OS	MC over W-OP	MC over W-OS
SynC	**0.18**	0.81	0.19	**5,446**	1,227	5,102	**4**	**1**
SynCN	**0.27**	52.9	7.34	**3,656**	18	135	**203**	**27**
KDD	**0.95**	1369.5	222.8	**4,190**	2	17	**2,095**	**246**
Forest Cover	**2.11**	213.1	10.79	**1,899**	18	370	**105**	**5**

the size of the data stream, causing both the memory requirement and running time to increases linearly. But the running time and memory requirement of MC remains the same over the entire length of the stream.

6 Conclusion

We have presented a novel technique to detect new classes in concept-drifting data streams. Most of the novelty detection techniques either assume that there is no concept-drift, or build a model for a single "normal" class and consider all other classes as novel. But our approach is capable of detecting novel classes

in the presence of concept-drift, and even when the model consists of multiple "existing" classes. Besides, our novel class detection technique is non-parametric, meaning, it does not assume any specific distribution of data. We also show empirically that our approach outperforms the state-of-the art data stream based novelty detection techniques in both classification accuracy and processing speed.

It might appear to readers that in order to detect novel classes we are in fact examining whether new clusters are being formed, and therefore, the detection process could go on without supervision. But supervision is necessary for classification. Without external supervision, two separate clusters could be regarded as two different classes, although they are not. Conversely, if more than one novel classes appear in a chunk, all of them could be regarded as a single novel class if the labels of those instances are never revealed. In future, we would like to apply our technique in the domain of multiple-label instances.

Acknowledgment

This research was funded in part by NASA grant NNX08AC35A.

References

1. Chen, S., Wang, H., Zhou, S., Yu, P.: Stop chasing trends: Discovering high order models in evolving data. In: Proc. ICDE, pp. 923–932 (2008)
2. Hulten, G., Spencer, L., Domingos, P.: Mining time-changing data streams. In: Proc. ACM SIGKDD, pp. 97–106 (2001)
3. Kolter, J., Maloof, M.: Using additive expert ensembles to cope with concept drift. In: Proc. ICML, pp. 449–456 (2005)
4. Markou, M., Singh, S.: Novelty detection: A review-part 1: Statistical approaches, part 2: Neural network based approaches. Signal Processing 83 (2003)
5. Masud, M.M., Gao, J., Khan, L., Han, J., Thuraisingham, B.: A practical approach to classify evolving data streams: Training with limited amount of labeled data. In: Proc. ICDM, pp. 929–934 (2008)
6. Roberts, S.J.: Extreme value statistics for novelty detection in biomedical signal processing. In: Proc. Int. Conf. on Advances in Medical Signal and Information Processing, pp. 166–172 (2000)
7. Spinosa, E.J., de Leon, A.P., de Carvalho, F., Gama, J.: Olindda: a cluster-based approach for detecting novelty and concept drift in data streams. In: Proc. 2007 ACM symposium on Applied computing, pp. 448–452 (2007)
8. University of Texas at Dallas Technical report UTDCS-13-09 (June 2009), `http://www.utdallas.edu/~mmm058000/reports/UTDCS-13-09.pdf`
9. Wang, H., Fan, W., Yu, P., Han, J.: Mining concept-drifting data streams using ensemble classifiers. In: Proc. ACM SIGKDD, pp. 226–235 (2003)
10. yan Yeung, D., Chow, C.: Parzen-window network intrusion detectors. In: Proc. International Conference on Pattern Recognition, pp. 385–388 (2002)
11. Yang, Y., Wu, X., Zhu, X.: Combining proactive and reactive predictions for data streams. In: Proc. ACM SIGKDD, pp. 710–715 (2005)
12. Yang, Y., Zhang, J., Carbonell, J., Jin, C.: Topic-conditioned novelty detection. In: Proc. ACM SIGKDD, pp. 688–693 (2002)