

Classification and Novel Class Detection of Data Streams in a Dynamic Feature Space

Mohammad M. Masud¹, Qing Chen¹, Jing Gao²,
Latifur Khan¹, Jiawei Han², and Bhavani Thuraisingham¹

¹ University of Texas at Dallas

² University of Illinois at Urbana Champaign

{mehedy, qingch}@utdallas.edu, jinggao3@uiuc.edu

lkhan@utdallas.edu, hanj@cs.uiuc.edu, bhavani.thuraisingham@utdallas.edu

Abstract. Data stream classification poses many challenges, most of which are not addressed by the state-of-the-art. We present DXMiner, which addresses four major challenges to data stream classification, namely, infinite length, concept-drift, concept-evolution, and feature-evolution. Data streams are assumed to be infinite in length, which necessitates single-pass incremental learning techniques. Concept-drift occurs in a data stream when the underlying concept changes over time. Most existing data stream classification techniques address only the infinite length and concept-drift problems. However, concept-evolution and feature-evolution are also major challenges, and these are ignored by most of the existing approaches. Concept-evolution occurs in the stream when novel classes arrive, and feature-evolution occurs when new features emerge in the stream. Our previous work addresses the concept-evolution problem in addition to addressing the infinite length and concept-drift problems. Most of the existing data stream classification techniques, including our previous work, assume that the feature space of the data points in the stream is static. This assumption may be impractical for some type of data, for example text data. DXMiner considers the dynamic nature of the feature space and provides an elegant solution for classification and novel class detection when the feature space is dynamic. We show that our approach outperforms state-of-the-art stream classification techniques in classifying and detecting novel classes in real data streams.

1 Introduction

The goal of data stream classification is to learn a model from past labeled data, and classify future instances using the model. There are many challenges in data stream classification. First, data streams have *infinite length*, and so, it is impossible to store all the historical data for training. Therefore, traditional learning algorithms that require multiple passes over the whole training data 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. A classification model must adapt itself to the most recent concept in order to

cope with concept-drift. Third, novel classes may appear in the stream, which we call *concept-evolution*. In order to cope with concept-evolution, a classification model must be able to automatically detect novel classes.

Finally, the feature space that represents a data point in the stream may change over time. For example, consider a text stream where each data point is a document, and each word is a feature. Since it is impossible to know which words will appear in the future, the complete feature space is unknown. Besides, it is customary to use only a subset of the words as the feature set because most of the words are likely to be redundant for classification. Therefore at any given time, the feature space is defined by the useful words (i.e., features) selected using some selection criteria. Since in the future, new words may become useful and old useful words may become redundant, the feature space changes dynamically. We call this dynamic nature of features as *feature-evolution*. In order to cope with feature-evolution, the classification model should be able to correctly classify a data point having a different feature space than the feature space of the model. Most existing data stream classification techniques address only the infinite length, and concept-drift problems [1, 11, 5, 3, 9]. Our previous work XMiner [6] addresses the concept-evolution problem in addition to the infinite length and concept-drift problems. In this paper, we propose DXMiner, which addresses *feature-evolution* as well as the other three challenges. Dealing with the feature-evolution problem becomes much challenging in the presence of concept-drift and concept-evolution.

DXMiner addresses the infinite length and concept-drift problems by applying a hybrid batch-incremental process [6,9], which is done as follows. The data stream is divided into equal sized chunks and a classification model is trained from each chunk. An ensemble of L such models is used to classify the unlabeled data. When a new model is trained from a data chunk, it replaces one of the existing models in the ensemble. In this way the ensemble is kept up-to-date. The infinite length problem is addressed by maintaining a fixed sized ensemble, and the concept-drift is addressed by keeping the ensemble up-to-date. DXMiner solves the concept-evolution problem by automatically detecting novel classes in the data stream [6]. In order to detect novel class, it first builds a decision boundary around the training data. During classification of unlabeled data, it first identifies the test data points that are outside the decision boundary. Such data points are called filtered outliers (*F-outliers*), and they represent data points that are *well separated* from the training data. Then if sufficient number of *F-outliers* are found that show strong cohesion among themselves (i.e., they are close together), the *F-outliers* are classified as novel class instances. Finally, DXMiner solves the feature-evolution problem by applying effective feature selection technique and dynamically converting the feature spaces of the classification models and the test instances.

We have several contributions. First, we propose a framework for classifying a data stream that observes infinite-length, concept-drift, concept-evolution, and feature-evolution. To the best of our knowledge, this is the first work that addresses all these challenges in a single framework. Second, we propose a realistic

feature extraction and selection technique for data streams, which selects the features for the test instances without knowing their labels. Third, we propose a fast and effective feature space conversion technique to address the feature-evolution problem. In this technique, we convert different heterogeneous feature spaces into one homogeneous space without losing any feature value. The effectiveness of this technique is established both analytically and empirically. Finally, we evaluate our framework on real data streams, such as Twitter messages, and NASA safety aviation reports, and achieve satisfactory performance over existing state-of-the-art data stream classification techniques.

The rest of the paper is organized as follows. Section 2 discusses relevant works in data stream classification. Section 3 describes the proposed framework in details, and Section 4 then explains our feature space conversion technique to cope with dynamic feature space. Section 5 reports the experimental results and analyzes them. Finally, Section 6 concludes with directions to future works.

2 Related Work

The challenges of data stream classification are addressed by different researchers in different ways. These approaches can be divided into three categories. Approaches belonging to the first category address the infinite length and concept-drift problems; approaches belonging to the second category address the infinite length, concept-drift, and feature-evolution problems; and approaches belonging to the third category address the infinite length, concept-drift, and concept-evolution problems.

Most of the existing techniques fall into the first category. There are two different approaches: single model classification, and ensemble classification. The single model classification techniques apply some form of incremental learning to address the infinite length problem, and strive to adapt themselves to the most recent concept to address the concept-drift problem [3, 1, 11]. Ensemble classification techniques [9, 5, 2] maintain a fixed-sized ensemble of models, and use ensemble voting to classify unlabeled instances. These techniques address the infinite length problem by applying a hybrid batch-incremental technique. Here the data stream is divided into equal sized chunks and a classification model is trained from each chunk. This model replaces one of the existing models in the ensemble, keeping the ensemble size constant. The concept-drift problem is addressed by continuously updating the ensemble with newer models, and striving to keep the ensemble consistent with the current concept. DXMiner also applies an ensemble classification technique.

Techniques in the second category address the feature-evolution problem on top of the infinite length and concept-drift problems. Katakis et al. [4] propose a feature selection technique for data streams having dynamic feature space. Their technique consists of an incremental feature ranking method and an incremental learning algorithm. Wenerstrom and Giraud-Carrier [10] propose a technique, called FAE, which also applies incremental feature selection, but their incremental learner is an ensemble of models. Their approach achieves relatively better

performance than the approach of Katakis et al [4]. There are several differences in the way that FAE and DXMiner approaches the feature-evolution problem. First, FAE uses the χ^2 statistics for feature selection, whereas DXMiner uses deviation weight (section 3.2). Second, in FAE, if a test instance has a different feature space than the classification model, the model uses its own feature space, but the test instance uses only those features that belong to the model’s feature space. In other words, FAE uses a Lossy-L conversion, whereas DXMiner uses Lossless conversion (see section 4). Furthermore, none of the proposed approaches of the second category detects novel class, but DXMiner does.

Techniques in the third category deal with the concept-evolution problem in addition to addressing the infinite length and concept-drift problems. An unsupervised novel concept detection technique for data streams is proposed in [8], but it is not applicable to multi-class classification. Our previous works MineClass and XMiner [6] address the concept-evolution problem on a multi-class classification framework. They can detect the arrival of a novel class automatically, without being trained with any labeled instances of that class. However, they do not address the feature-evolution problem. On the other hand, DXMiner addresses the more general case where features can evolve dynamically. DXMiner differs from all other data stream classification techniques in that it addresses all four major challenges in a single framework, whereas previous techniques address three or less challenges. Its effectiveness is shown analytically and demonstrated empirically on a number of real data streams.

3 Overview of DXMiner

In this section, we will briefly describe the system architecture of DXMiner (or DECSMiner), which stands for Dynamic feature based Enhanced Classifier for Data Streams with novel class Miner. Before describing the system, we define the concept of *novel class* and *existing class*.

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

3.1 Top Level Description

Algorithm 1 sketches the basic steps of DXMiner. The system consists of an ensemble of L classification models, $\{M_1, \dots, M_L\}$. The data stream is divided into equal sized chunks. When the data points of a chunk are labeled by an expert, it is used for training. The initial ensemble is built from first L data chunks (line 1).

Feature extraction and selection: It is applied on the raw data to extract all the features and select the best features for the latest unlabeled data chunk D_u (line 5). The feature selection technique is described in section 3.2. However, if the feature set is pre-determined, then the function (Extract&SelectFeatures) simply returns that feature set.

Algorithm 1. DXMiner

```

1:  $M \leftarrow \text{Build-initial-ensemble}()$ 
2:  $buf \leftarrow \text{empty}$  //temporary buffer
3:  $D_u \leftarrow \text{latest chunk of unlabeled instances}$ 
4:  $D_l \leftarrow \text{sliding window of last } r \text{ data chunks}$ 
5:  $\mathcal{F}_u \leftarrow \text{Extract\&Select-Features}(D_l, D_u)$  //Feature set for  $D_u$  (section 3.2)
6:  $Q \leftarrow D_u$  //FIFO queue of data chunks waiting to be labeled
7: while true do
8:   for all  $x_j \in D_u$  do
9:      $M', x'_j \leftarrow \text{Convert-Featurespace}(M, x_j, \mathcal{F}_u)$  //(section 3.4)
10:    NovelClass-Detection\&Classification $(M', x'_j, buf)$  //(section 3.5)
11:   end for
12:   if the instances in  $Q.front()$  are now labeled then
13:      $D_f \leftarrow Q$  //Dequeue
14:      $M \leftarrow \text{Train\&Update}(M, D_f)$  //(section 3.3)
15:      $D_l \leftarrow \text{move-window}(D_l, D_f)$  //slide the window to include  $D_f$ 
16:   end if
17:    $D_u \leftarrow \text{new chunk of unlabeled data}$ 
18:    $\mathcal{F}_u \leftarrow \text{Extract\&Select-Features}(D_l, D_u)$  //Feature set for  $D_u$ 
19:    $Q \leftarrow D_u$  //Enqueue
20: end while

```

D_u is enqueued into a queue of unlabeled data chunks waiting to be labeled (line 6). Each instance of the chunk D_u is then classified by the ensemble M (lines 8-11). Before classification, the models in the ensemble, as well as the test instances need to pass through a feature space conversion process.

Feature space conversion (line 9): It is not needed if the feature set for the whole data stream is static. However, if the feature space is dynamic, then we would have different feature sets in different data chunks. As a result, each model in the ensemble would be trained on different feature sets. Besides, the feature space of the test instances would also be different from the feature space of the models. Therefore, we apply a feature space conversion technique to homogenize the feature sets of the models and the test instances. See section 4 for details.

Novel class detection and classification (line 10): After the conversion of feature spaces, the test instance is examined by the ensemble of models to determine whether the instance should be identified as a novel class instance, or as one of the existing class instances. The buffer buf is used to temporarily store potential novel class instances. See section 3.5 for details.

The queue Q is checked to see if the chunk at the front (i.e., oldest chunk) is labeled. If yes, the chunk is dequeued, used to train a model, and the sliding window of labeled chunks is shifted right. By keeping the queue to store unlabeled data, we eliminate the constraint imposed by many approaches (e.g. [10]) that each new data point arriving in the stream should be labeled as soon as it is classified by the existing model.

Training and update(line 14): We learn a model from the training data. We also build a decision boundary around the training data in order to detect novel

classes. Each model also saves the set of features with which it is trained. The newly trained model replaces an existing model in the ensemble. The model to be replaced is selected by evaluating each of the models in the ensemble on the training data, and choosing the one with the highest error. See section 3.3 for details. Finally, when a new data chunk arrives, we again select best features for that chunk, and enqueue the chunk into Q .

3.2 Feature Extraction and Selection

The data points in the stream may or may not have a fixed feature set. If they have a fixed feature set, then we simply use that feature set. Otherwise, we apply a feature extraction and feature selection technique. Note that we need to select features for the instances of the test chunk before they can be classified by the existing models, since the classification models require the feature vectors for the test instances. However, since the instances of the test chunk are unlabeled, we cannot use supervised feature selection (e.g. information gain) on that chunk. To solve this problem, we propose two alternatives: *predictive* feature selection, and *informative* feature selection, to be explained shortly. Once the feature set has been selected for a test chunk, the feature values for each instance are computed, and feature vectors are produced. The same feature vector is used during classification (when unlabeled) and training (when labeled).

Predictive feature selection: Here, we predict the features of the test instances without using any of their information, rather we use the past labeled instances to predict the feature set of the test instances. This is done by extracting all features from the last r labeled chunks (D_l in the DXMiner algorithm), and then selecting the best R features using some selection criteria. In our experiments, we use $r=3$. One such popular selection criterion is information gain. We use another criterion which we call *deviation weight*. The deviation weight for the i -th feature for class c is given by: $dw_i = freq_i * \frac{freq_i^c}{N_c} * \frac{N - N_c}{freq_i - freq_i^c + \epsilon}$, where $freq_i$ is the total frequency of the i -th feature, $freq_i^c$ is the frequency of the i -th feature in class c , N_c is the number of instances of class c , N is the total number of instances, and ϵ is a smoothing constant. A higher value of deviation weight means greater discriminating power. For each class, we choose the top r features having the highest deviation weight. So, if there are total $|C|$ classes, then we select $R = |C|r$ features this way. These features are used as the feature space for the test instances. We use deviation weight instead of information gain in some data streams because this selection criterion achieves better classification accuracy (see section 5.3). Although information gain or deviation weight consider fixed number of classes, this does not affect the novel class detection process since the feature selection is used just to select the best features for the test instances. The test instances are still unlabeled, and therefore, novel class detection mechanism is applicable to them.

Informative feature selection: Here, we use the test chunk to select the features. We extract all possible features from the test chunk (D_u in the DXMiner algorithm), and select the best R features in an unsupervised way. For example,

one such unsupervised selection criterion is to choose the R highest frequency features in the chunk. This strategy is very useful in data streams like to the Twitter (see section 5).

3.3 Training and Update

The feature vectors constructed in the previous step (section 3.2) are supplied to the learning algorithm to train a model. In our case, we use a semi-supervised clustering technique to train a K-NN based classifier [7]. We build K clusters with the training data, applying a semi-supervised clustering technique. After building the clusters, we save the cluster summary (mentioned as *pseudopoint*) of each cluster. The summary contains the *centroid*, *radius*, and *frequencies* of data points belonging to each class. The radius of a pseudopoint is defined as the distance between the centroid and the farthest data point in the cluster. The raw data points are discarded after creating the summary. Therefore, each model M_i is a collection of K pseudopoints. A test instance x_j is classified using M_i as follows. We find the pseudopoint $h \in M_i$ whose centroid is nearest from x_j . The predicted class of x_j is the class that has the highest frequency in h . x_j is classified using the ensemble M by taking a majority voting among all classifiers.

Each pseudopoint corresponds to a “hypersphere” in the feature space having center at the centroid, and a radius equal to its radius. Let $\mathcal{S}(h)$ be the *feature space* covered by such a hypersphere of pseudopoint h . The *decision boundary* of a model M_i (or $\mathcal{B}(M_i)$) is the union of the feature spaces (i.e., $\mathcal{S}(h)$) of all pseudopoints $h \in M_i$. The decision boundary of the ensemble M (or $\mathcal{B}(M)$) is the union of the decision boundaries (i.e., $\mathcal{B}(M_i)$) of all models $M_i \in M$.

The ensemble is updated by the newly trained classifier as follows. Each existing model in the ensemble is evaluated on the latest training chunk, and their error rates are obtained. The model having the highest error is replaced with the newly trained model. This ensures that we have exactly L models in the ensemble at any given point of time.

3.4 Feature Space Conversion: Explained in Details in Section 4

3.5 Classification and Novel Class Detection

Each instance in the most recent unlabeled chunk is first examined by the ensemble of models to see if it is outside the decision boundary of the ensemble (i.e., $\mathcal{B}(M)$). If it is inside, then it is classified normally (i.e., using majority voting) using the ensemble of models. Otherwise, it is declared as an *F-outlier*, or filtered outlier. We assume that any class of data has the following 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).*

So, if there is a novel class in the stream, instances belonging to the class will be far from the existing class instances and will be close to other novel class

instances. Since F -outliers are outside $\mathcal{B}(M)$, they are far from the existing class instances. So, the separation property for a novel class is satisfied by the F -outliers. Therefore, F -outliers are potential novel class instances, and they are temporarily stored in the buffer buf (see algorithm 1) to observe whether they also satisfy the cohesion property. We then examine whether there are enough F -outliers that are close to each other. This is done by computing the following metric, which we call the q -Neighborhood Silhouette Coefficient, or q -NSC [6] (to be explained shortly).

Definition 2 (λ_c -neighborhood). *The λ_c -neighborhood of an Foutlier x is the set of q -nearest neighbors of x belonging to class c .*

Here q is a user defined parameter. For brevity, we denote the λ_c -neighborhood of an F -outlier x as $\lambda_c(x)$. Thus, $\lambda_+(x)$ of an F -outlier x is the set of q instances of class c_+ , that are closest to the outlier x . Similarly, $\lambda_o(x)$ refers to the set of q F -outliers that are closest to x . Let $\bar{D}_{c_{out},q}(x)$ be the mean distance from an F -outlier x to its q -nearest F -outlier instances (i.e., to its $\lambda_o(x)$ neighborhood), Also, let $\bar{D}_{c_{min},q}(x)$ be the mean distance from x to its closest existing class neighborhood ($\lambda_{c_{min}}(x)$). Then q -NSC of x is given by:

$$q\text{-NSC}(x) = \frac{\bar{D}_{c_{min},q}(x) - \bar{D}_{c_{out},q}(x)}{\max(\bar{D}_{c_{min},q}(x), \bar{D}_{c_{out},q}(x))} \quad (1)$$

q -NSC, a unified measure of cohesion and separation, yields a value between -1 and +1. A positive value indicates that x is closer to the F -outlier instances (more cohesion) and farther away from existing class instances (more separation), and vice versa. q -NSC(x) of an F -outlier x must be computed separately for each classifier $M_i \in M$. We declare a *new class* if there are at least q' ($> q$) F -outliers having positive q -NSC for all classifiers $M_i \in M$. In order to reduce the time complexity in computing q -NSC(), we cluster the F -outliers, and compute q -NSC() of those clusters only. The q -NSC() of each such cluster is used as the approximate q -NSC() value of each data point in the cluster.

It is worthwhile to mention here that we do not make any assumption about the number of novel classes. If there are two or more novel classes appearing at the same time, all of them will be detected as long as each one of them satisfies property-1 and each of them has $> q$ instances. However, we will tag them simply as “novel class”, i.e., no distinction will be made among them. But the distinction will be learned by our model as soon as those instances are labeled by human experts, and a classifier is trained with them.

4 Feature Space Conversion

It is obvious that the data streams that do not have any fixed feature space (such as text stream) will have different feature spaces for different models in the ensemble, since different sets of features would likely be selected for different chunks. Besides, the feature space of test instances is also likely to be different

from the feature space of the classification models. Therefore, when we need to classify an instance, we need to come up with a homogeneous feature space for the model and the test instances. There are three possible alternatives: i) Lossy fixed conversion (or *Lossy-F* conversion in short), ii) Lossy local conversion (or *Lossy-L* conversion in short), and iii) Lossless homogenizing conversion (or *Lossless* conversion in short).

4.1 Lossy Fixed (Lossy-F) Conversion

Here we use the same feature set for the entire stream, which had been selected for the first data chunk (or first n data chunks). This will make the feature set fixed, and therefore all the instances in the stream, whether training or testing, will be mapped to this feature set. We call this a lossy conversion because future models and instances may lose important features due to this conversion.

Example: let $\mathcal{F}_S = \{\mathcal{F}_a, \mathcal{F}_b, \mathcal{F}_c\}$ be the features selected in the first n chunks of the stream. With the Lossy-F conversion, all future instances will be mapped to this feature set. That is, suppose the set of features for a future instance x be: $\{\mathcal{F}_a, \mathcal{F}_c, \mathcal{F}_d, \mathcal{F}_e\}$, and the corresponding feature values of x be: $\{x_a, x_c, x_d, x_e\}$. Then after conversion, x will be represented by the following values: $\{x_a, 0, x_c\}$. In other words, any feature of x that is not in F_S (i.e., \mathcal{F}_d and \mathcal{F}_e) will be discarded, and any feature of F_S that is not in x (i.e., \mathcal{F}_b) will be assumed to have a zero value. All future models will also be trained using F_S .

4.2 Lossy Local (Lossy-L) Conversion

In this case, each training chunk, as well as the model built from the chunk, will have its own feature set selected using the feature extraction and selection technique. When a test instance is to be classified using a model M_i , the model will use its own feature set as the feature set of the test instance. This conversion is also lossy because the test instance might lose important features as a result of this conversion.

Example: the same example of section 4.1 is applicable here, if we let F_S to be the selected feature set for a model M_i , and let x to be an instance being classified using M_i . Note that for the Lossy-F conversion, F_S is the same over all models, whereas for Lossy-L conversion, F_S is different for different models.

4.3 Lossless Homogenizing (Lossless) Conversion

Here, each model has its own selected set of features. When a test instance x is to be classified using a model M_i , both the model and the instance will convert their feature sets to the union of their feature sets. We call this conversion “lossless homogenizing” since both the model and the test instance preserve their dimensions (i.e., features), and the converted feature space becomes homogeneous for both the model and the test instance. Therefore, no useful features are lost as a result of the conversion.

Example: continuing from the previous example, let $\mathcal{F}_S = \{\mathcal{F}_a, \mathcal{F}_b, \mathcal{F}_c\}$ be the feature set of a model M_i , $\{\mathcal{F}_a, \mathcal{F}_c, \mathcal{F}_d, \mathcal{F}_e\}$ be the feature set of the test instance x , and $\{x_a, x_c, x_d, x_e\}$ be the corresponding feature values of x . Then after conversion, both x and M_i will have the following features: $\{\mathcal{F}_a, \mathcal{F}_b, \mathcal{F}_c, \mathcal{F}_d, \mathcal{F}_e\}$. Also, x will be represented with the following feature values: $\{x_a, 0, x_c, x_d, x_e\}$. In other words, all the features of x will be included in the converted feature set, and any feature of F_S that is not in x (i.e., \mathcal{F}_b) will be assumed to be zero.

4.4 Advantage of Lossless Conversion over Lossy Conversions

Lossless conversion is preferred over Lossy conversions because no features are lost due to this conversion. Our main assumption is that Lossless conversion preserves the properties of a novel class. That is, if an instance belongs to a novel class, it remains outside the decision boundary of any model M_i of the ensemble M in the converted feature space. However, this is not true for a Lossy-L conversion, as the following theorem states.

Lemma 1. *If a test point x belongs to a novel class, it will be mis-classified by the ensemble M as an existing class instance under certain conditions when the Lossy-L conversion is used.*

Proof. According to our algorithm, if x remains inside the decision boundary of any model $M_i \in M$, then the ensemble M considers it as an existing class instance. Let $M_i \in M$ be the model under question. Without loss of generality, let M_i and x have m and n features, respectively, l of which are common features. That is, let the features of the model be $\{\mathcal{F}_{i_1}, \dots, \mathcal{F}_{i_m}\}$ and the features of x be $\{\mathcal{F}_{j_1}, \dots, \mathcal{F}_{j_n}\}$, where $i_k = j_k$ for $0 \leq k \leq l$. In the boundary case, $l=0$, i.e., no features are common between M_i and x . Let h be the pseudopoint in M_i that is closest to x , and also, R be the radius of h , and \mathcal{C} be the centroid of h . The Lossless feature space would be the union of the features of M_i and x , which is: $\{\mathcal{F}_{i_1}, \dots, \mathcal{F}_{i_l}, \mathcal{F}_{i_{l+1}}, \dots, \mathcal{F}_{i_m}, \mathcal{F}_{j_{l+1}}, \dots, \mathcal{F}_{j_n}\}$ According to our assumption that the properties of novel class are preserved with the Lossless conversion, x will remain outside the decision boundary of all models $M_i \in M$ in the converted feature space. Therefore, the distance from x to the centroid \mathcal{C} will be greater than R .

Let the feature values of the centroid \mathcal{C} in the original feature space be: $\{y_{i_1}, \dots, y_{i_m}\}$, where y_{i_k} is the value of feature \mathcal{F}_{i_k} . After Lossless conversion, the feature values of \mathcal{C} in the new feature space would be: $\{y_{i_1}, \dots, y_{i_m}, 0, \dots, 0\}$. That is, all feature values for the added features $\{\mathcal{F}_{j_{l+1}}, \dots, \mathcal{F}_{j_n}\}$ are zeros. Also, let the feature values of x in the original feature space be: $\{x_{j_1}, \dots, x_{j_n}\}$. The feature values of x after the Lossless conversion would be:

$\{x_{j_1}, \dots, x_{j_l}, 0, \dots, 0, x_{j_{l+1}}, \dots, x_{j_n}\}$, that is, the feature values for the added features are all zeros. Without loss of generality, let Euclidean distance be the distance metric. Let D be the distance from x to the centroid \mathcal{C} . Therefore, we can deduce:

$$\begin{aligned}
 D^2 &= (\mathbf{C} - \mathbf{x})^2 > R^2 \\
 \Rightarrow R^2 &< \sum_{k=1}^l (y_{i_k} - x_{j_k})^2 + \sum_{k=l+1}^m (y_{i_k} - 0)^2 + \sum_{k=l+1}^n (0 - x_{j_k})^2 \quad (2)
 \end{aligned}$$

Now, let $A^2 = \sum_{k=1}^l (y_{i_k} - x_{j_k})^2 + \sum_{k=l+1}^m (y_{i_k} - 0)^2$, and $B^2 = \sum_{k=l+1}^n (0 - x_{j_k})^2$. Note that with the Lossy-L conversion, the distance from x to \mathcal{C} would be A , since the converted feature space is the same as the original feature space of M_i . So, it follows that:

$$\begin{aligned}
 R^2 &< A^2 + B^2 \Rightarrow R^2 = A^2 + B^2 - e^2 \quad (\text{letting } e > 0) \\
 \Rightarrow A^2 &= R^2 + (e^2 - B^2) \Rightarrow A^2 < R^2 \quad (\text{provided that } e^2 - B^2 < 0)
 \end{aligned}$$

Therefore, in the Lossy-L converted feature space, the distance from x to the centroid \mathcal{C} is less than the radius of the pseudopoint h , meaning, x is inside the region of h , and as a result, x is inside decision boundary of M_i . Therefore, x is mis-classified as an existing class instance by M_i when the Lossy-L conversion is used, under the condition that $e^2 < B^2$ \square .

This lemma is supported by our experimental results, which show that Lossy-L conversion mis-classifies most of the novel class instances as existing class. It might appear to the reader that increasing the dimension of the models and the test instances may have an undesirable side effect due to curse of dimensionality. However, it is reasonable to assume that the feature set of the test instances is not dramatically different from the feature sets of the classification models because the models usually represent the most recent concept. Therefore, the converted dimension of the feature space should be almost the same as the original feature spaces. Furthermore, this type of conversion has been proved to be successful in other popular classification techniques such as Support Vector Machines.

5 Experiments

5.1 Dataset

We use four different datasets having different characteristics (see table 1).

Twitter dataset (Twitter): This dataset contains 170,000 Twitter messages (tweets) of seven different trends (classes). These tweets have been retrieved from <http://search.twitter.com/trends/weekly.json> using a tweets crawling program written in Perl script. The raw data is in free text and we apply preprocessing to get a useful dataset. The preprocessing consists of two steps. First,

Table 1. Summary of the datasets used

Dataset	Concept-drift	Concept-evolution	Feature-evolution	Features	Instances	Classes
Twitter	✓	✓	✓	30	170,000	7
ASRS	X	X	✓	50	135,000	13
KDD	✓	✓	X	34	490,000	22
Forest	X	✓	X	54	581,000	7

filtering is performed on the messages to filter out words that match against a stop word list. Examples of stop words are articles ('a', 'an', 'the'), acronyms ('lol', 'btw') etc. Second, we use Wiktionary to retrieve the parts of speech (POS) of the remaining words, and remove all pronouns (e.g., 'I', 'u'), change tense of verbs (e.g. change 'did' and 'done' to 'do'), change plurals to singulars and so on. We apply the *informative feature selection* (section 3.2) technique on the Twitter dataset. Also, we generate the feature vector for each message using the following formula: $w_{ij} = \beta * f(a_i, m_j) / \sum_{j=1}^S f(a_i, m_j)$ where w_{ij} is the value of the i th feature (a_i) for the j th message in the chunk, $f(a_i, m_j)$ is the frequency of feature a_i in message m_j , and β is a normalizing constant.

NASA Aviation Safety Reporting System dataset (ASRS): This dataset contains around 135,000 text documents. Each document is actually a report corresponding to a flight anomaly. There are 13 different types of anomalies (or classes), such as "aircraft equipment problem : critical", "aircraft equipment problem : less severe". These documents are treated as a data stream by arranging the reports in order of their creation time. The documents are normalized using a software called PLADS, which removes stop words, expands abbreviations, and performs stemming (e.g. changing tense of verbs). The instances in the dataset are multi-label, meaning, an instance may have more than one class labels. We transform the multi-label classification problem into 13 separate binary classification problems, one for each class. When reporting the accuracy, we report the average accuracy of the 13 datasets. We apply the *predictive feature selection* (section 3.2) for the ASRS dataset. We use deviation weight for feature selection, which works better than information gain (see section 5.3). The feature values are produced using the same formula that is used for Twitter dataset.

KDD cup 1999 intrusion detection dataset (KDD) and Forest cover dataset from UCI repository (Forest): See [6] for details.

5.2 Experimental Setup

Baseline techniques: **DXMiner:** This is the proposed approach with the *Lossless* feature space conversion. **Lossy-F:** This approach the same as DXMiner except that the *Lossy-F* feature space conversion is used. **Lossy-L:** This is DXMiner with the *Lossy-L* feature space conversion. **O-F:** This is a combination of the *OLINDDA* [8] approach with *FAE* [10] approach. We combine these two, because to the best of our knowledge, no other approach can work with dynamic feature vector and detect novel classes in data streams. In this combination, *OLINDDA works as the novel class detector, and FAE performs classification.* This is done as follows: 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 *FAE*. *FAE* uses the *Lossy-L* conversion of feature spaces. *OLINDDA* is also adapted to this conversion. For fairness, the underlying learning algorithm for *FAE* is chosen the same as that of *DXMiner*. Since *OLINDDA* assumes that there is only one "normal" class, 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.

Parameters settings: DXMiner: R (feature set size) = 30 for Twitter and 50 for ASRS. Note that R is only used for data streams having feature-evolution. K (number of pseudopoints per chunk) = 50, S (chunk size) = 1000, L (ensemble size) = 6, and q (minimum number of F -outliers required to declare a novel class) = 50. These parameter values are reasonably stable, which are obtained by running DXMiner on a number of real and synthetic datasets. Sensitivity to different parameters are discussed in details in [6].

OLINDDA: Number of data points per cluster (N_{excl}) = 30, least number of normal instances needed to update the existing model = 100, least number of instances needed to build the initial model = 100. FAE: m (maturity) = 200, p (probation time)=4000, f (feature change threshold) =5, r (growth rate)=10, N (number of instances) =1000, M (feature selected) = same as R of DXMiner. These parameters are chosen either according to the default values used in OLINDDA, FAE, or by trial and error to get an overall satisfactory performance.

5.3 Evaluation

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 3 chunks. From the 4th chunk onward, we first evaluate the performances of each method on that chunk, then use that chunk to update the existing models. The performance metrics for each chunk for each method are saved and averaged for producing the summary result.

Figures 1(a),(c) show the ERR rates and total number of missed novel classes respectively, for each approach throughout the stream in Twitter dataset. For example in figure 1(a), at X axis = 150, the Y values show the average ERR of each approach from the beginning of the stream to chunk 150. At this point, the ERR of DXMiner, Lossy-F, Lossy-L, and O-F are 4.4% 35.0%, 1.3%, and 3.2%, respectively. Figure 1(c) show the total number of novel instances missed for each of the baseline approaches. For example, at the same value of X axis, the Y values show the total novel instances missed (i.e., misclassified as existing class) for each approach from the beginning of the stream to chunk 150. At this point, the number of novel instances missed by DXMiner, Lossy-F, Lossy-L, and O-F are 929, 0, 1731, and 2229 respectively. The total number of novel class instances at this point is 2287, which is also shown in the graph.

Note that although O-F and Lossy-L have lower ERR than DXMiner, they have higher M_{new} rates, as they misses most of the novel class instances. This is because both FAE and Lossy-L use the Lossy-L conversion, which, according to Lemma 1, is likely to mis-classify more novel class instances as existing class instance (i.e., have higher M_{new} rates). On the other hand, Lossy-F has zero

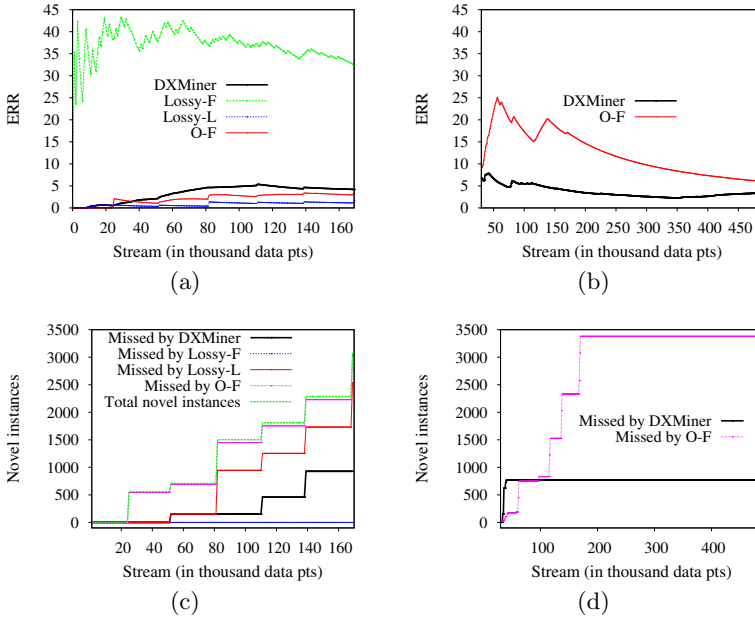


Fig. 1. ERR rates and missed novel classes in Twitter (a,c) and Forest (b,d) datasets

M_{new} rate, but it has very high false positive rate. This is because it wrongly recognizes most of the data points as novel class as a fixed feature vector is used for training the models; although newer and more powerful features evolve often in the stream. Figure 1(b),(d) show the ERR rates and number of novel classes missed, respectively, for Forest dataset. Note that since the feature vector is fixed for this dataset, no feature space conversion is required, and therefore, Lossy-L and Lossy-F are not applicable here. We also generate ROC curves for the Twitter, KDD, and Forest datasets by plotting false novel class detection rate (false positive rate if we consider novel class as positive class and existing classes as negative class) against true novel class detection rate (true positive

Table 2. Summary of the results

Dataset	Method	ERR	M_{new}	F_{new}	AUC	FP	FN
Twitter	DXMiner	4.2	30.5	0.8	0.887	-	-
	Lossy-F	32.5	0.0	32.6	0.834	-	-
	Lossy-L	1.6	82.0	0.0	0.764	-	-
	O-F	3.4	96.7	1.6	0.557	-	-
ASRS	DXMiner	0.02	-	-	0.996	0.00	0.1
	DXMiner(info-gain)	1.4	-	-	0.967	0.04	10.3
	O-F	3.4	-	-	0.876	0.00	24.7
Forest	DXMiner	3.6	8.4	1.3	0.973	-	-
	O-F	5.9	20.6	1.1	0.743	-	-
KDD	DXMiner	1.2	5.9	0.9	0.986	-	-
	O-F	4.7	9.6	4.4	0.967	-	-

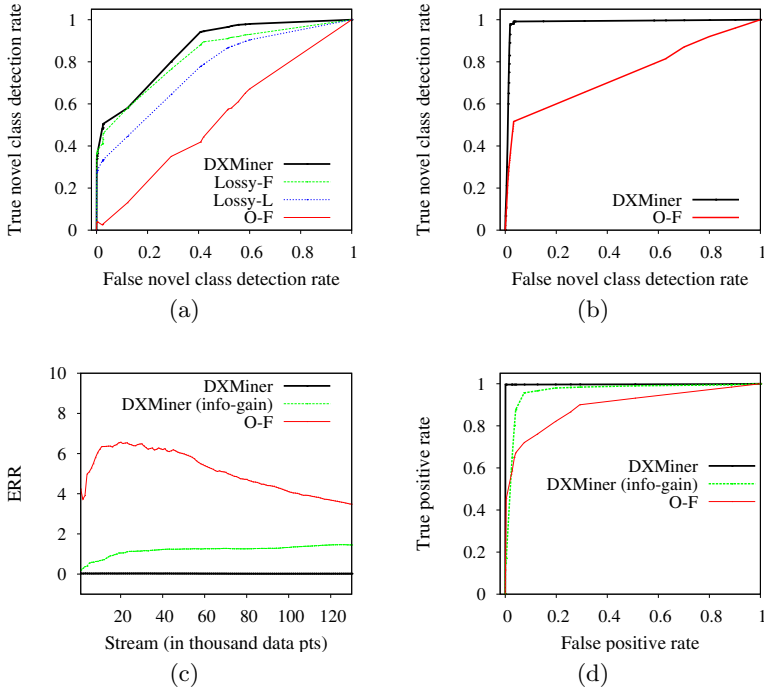


Fig. 2. ROC curves for (a) Twitter, (b) Forest dataset; ERR rates (c) and ROC curves (d) for ASRS dataset

rate). The ROC curves corresponding to Twitter and Forest datasets are shown in figures 2(a,b), and the corresponding AUCs are reported in table 2.

Figure 2(c) shows the ERR rates for ASRS dataset, averaged over all 13 classes. Here DXMiner (with deviation weight feature selection criterion) has the lowest error rate. Figure 2(d) shows the corresponding ROC curves. Each ROC curve is averaged over all 13 classes. Here too, DXMiner has the highest area under the curve (AUC), which is 0.996, whereas O-F has AUC=0.876. Table 2 shows the summary of performances of all approaches in all datasets. Note that for the ASRS we report false positive (FP) and false negative (FN) rates, since ASRS does not have any novel classes. The FP and FN rates are averaged over all 13 classes. For any dataset, DXMiner has the highest AUC.

The running times (training plus classification time per 1,000 data points) of DXMiner and O-F for different datasets are 26.4 and 258 (Twitter), 34.9 and 141 (ASRS), 2.2 and 13.1 (Forest), and 2.6 and 66.7 seconds (KDD), respectively. It is obvious that DXMiner is at least 4 times or more faster than O-F in any dataset. Twitter and ASRS datasets require longer running times than Forest and KDD due to the feature space conversions at runtime. O-F is much slower than DXMiner because $|C|$ OLINDDA models run in parallel, where $|C|$ is the number of classes, making O-F roughly $|C|$ times slower than DXMiner.

6 Conclusion

We have presented a novel technique to detect new classes in concept-drifting data streams having dynamic feature space. Most of the existing data stream classification techniques either cannot detect novel class, or does not consider the dynamic nature of feature spaces. We have analytically demonstrated the effectiveness of our approach, and empirically shown that our approach outperforms the state-of-the art data stream classification techniques in both classification accuracy and processing speed. In the future, we would like to address the multi-label classification problem in data streams.

References

1. Chen, S., Wang, H., Zhou, S., Yu, P.: Stop chasing trends: Discovering high order models in evolving data. In: Proc. ICDE 2008, pp. 923–932 (2008)
2. Fan, W.: Systematic data selection to mine concept-drifting data streams. In: Proc. ACM SIGKDD, Seattle, WA, USA, pp. 128–137 (2004)
3. Hulten, G., Spencer, L., Domingos, P.: Mining time-changing data streams. In: SIGKDD, San Francisco, CA, USA, pp. 97–106 (August 2001)
4. Katakis, I., Tsoumakas, G., Vlahavas, I.: Dynamic feature space and incremental feature selection for the classification of textual data streams. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) PKDD 2006. LNCS (LNAI), vol. 4213, pp. 102–116. Springer, Heidelberg (2006)
5. Kolter, J., Maloof, M.: Using additive expert ensembles to cope with concept drift. In: ICML, Bonn, Germany, pp. 449–456 (August 2005)
6. 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); Extended version is in the preprints, IEEE TKDE, vol. 99 (2010), doi = <http://doi.ieeecomputersociety.org/10.1109/TKDE.2010.61>
7. 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: Perner, P. (ed.) ICDM 2008. LNCS (LNAI), vol. 5077, pp. 929–934. Springer, Heidelberg (2008)
8. Spinosa, E.J., de Leon, A.P., de Carvalho, F., Gama, J.: Cluster-based novel concept detection in data streams applied to intrusion detection in computer networks. In: ACM SAC, pp. 976–980 (2008)
9. Wang, H., Fan, W., Yu, P.S., Han, J.: Mining concept-drifting data streams using ensemble classifiers. In: KDD 2003, pp. 226–235 (2003)
10. Wenerstrom, B., Giraud-Carrier, C.: Temporal data mining in dynamic feature spaces. In: Perner, P. (ed.) ICDM 2006. LNCS (LNAI), vol. 4065, pp. 1141–1145. Springer, Heidelberg (2006)
11. Yang, Y., Wu, X., Zhu, X.: Combining proactive and reactive predictions for data streams. In: Proc. SIGKDD, pp. 710–715 (2005)