Classification under Streaming Emerging New Classes: A Solution using Completely Random Trees

Xin Mu*, Kai Ming Ting♯, Zhi-Hua Zhou**

*National Key Laboratory for Novel Software Technology
Nanjing University, Nanjing 210093, China

♯ School of Engineering and Information Technology,
Federation University, Victoria, Australia.

Abstract

This paper investigates an important problem in stream mining, i.e., classification under streaming emerging new classes or *SENC*. The common approach is to treat it as a classification problem and solve it using either a supervised learner or a semi-supervised learner. We propose an alternative approach by using unsupervised learning as the basis to solve this problem. The *SENC* problem can be decomposed into three sub problems: detecting emerging new classes, classifying for known classes, and updating models to enable classification of instances of the new class and detection of more emerging new classes. The proposed method employs completely random trees which have been shown to work well in unsupervised learning and supervised learning independently in the literature. This is the first time, as far as we know, that completely random trees are used as a single common core to solve all three sub problems: unsupervised learning, supervised learning and model update in data streams. We show that the proposed unsupervised-learning-focused method often achieves significantly better outcomes than existing classification-focused methods.

Key words: Data stream, Emerging new class, Ensemble method, Completely Random Trees

This paper investigates an important problem in data streams, i.e., classification under streaming emerging new class or *SENC*. In many real-world data mining problems, the environment is open

*Corresponding author. Email: zhouzh@nju.edu.cn

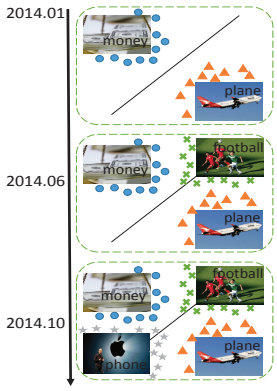


Figure 1: Image classification in a data stream

and changes gradually. In the streaming classification problem, some new classes are likely to emerge as the environment changes. The predictive accuracy of a previously trained classifier will be severely degraded if it is used to classify instances of a previously unseen class in the data stream. Ideally, we would like instances of a new class to be detected as soon as they emerge in the data stream; and only instances which are likely to belong to known classes are passed to the classifier to predict their classes.

It is assumed that true class labels are not available throughout the entire process, except a training set of known classes which is used to train a classifier (and a detector for new classes) at the beginning of the data stream. After the deployment of the classifier (and the detector), any future updates of the models must rely on the unlabelled instances as they appear in the data stream. Note that this assumption does not prevent the proposed method from using true class labels when they are available. It sets the hardest condition in the *SENC* problem.

An illustrative example is provided in Figure 1 which shows a news image classifier system making predictions in a data stream. Assume that a classifier about news content is built in early 2014, which starts with two classes (money and airplane); then some new classes (football and phone) emerge in two later periods in the data stream. The system must have the ability to detect those new classes and update itself timely in order to maintain the predictive accuracy.

Conceptually, the *SENC* problem can be decomposed into three sub problems: detecting emerging new classes, classifying known classes, and updating models to enable classification of instances of the new classes and detection of more emerging new classes. For every test instance in a data stream, the detector acts as a filter to determine whether it is likely to belong to a known class.

If it is, the instance is passed on to the classifier to produce a class prediction. Otherwise, the instance is declared a new class and placed in a buffer which stores candidates of previously unseen class. When the candidates have reached the buffer size, they are used to update both the classifier and the detector. The process repeats in the data stream after the models are updated.

The overall aim of the task is to maintain high classification accuracy continuously in a data stream. Thus, the challenges in the *SENC* problem are to detect emerging new classes and classify instances of known classes with high accuracy, and to perform model update efficiently in data streams. In order to maintain the model complexity to a reasonable size, model components related to currently inactive classes must be eliminated from the current model.

We show that these challenges can be met by using completely random trees, and the proposed method often achieves significantly better outcomes than existing more complicated methods. The proposed method has the following distinguishing features:

- The proposed method employs an unsupervised learning method as the basis to solve the *SENC* problem, and has a single common core which acts as distinct unsupervised learner and supervised learner. In contrast, most existing methods treat this problem as a classification problem and employ a supervised or semi-supervised learning approach [MP03, DYZ14] to solve it.
- The method explicitly differentiates anomalies of known classes from instances of emerging new classes using an unsupervised learning anomaly detection approach.
- The model is updated without the initial training set because the proposed method does not need to train new models for every future model updated. In contrast, most existing methods must keep this training set in order to train new models (e.g., LACU-SVM [DYZ14].)

Note that most of the existing methods mentioned above are designed to solve part of the *SENC* problem only. Details are provided in Section 2.

Our main contribution is the proposal to shift the focus of treating *SENC* as a classification problem to one based on unsupervised anomaly detection problem. In other words, the focus is shifted from the second sub problem to the first sub problem which is more critical in solving the

entire problem. This shift brings about an integrated approach to solve all three sub problems in *SENC*. No such solution exists in the current classification-focused approaches, as far as we know.

The rest of this paper is organized as follows: Section 1 describes the intuition of the proposed algorithm. Section 2 reviews the related work. Section 4 and 5 describe related definitions and the details of the proposed algorithm. We report the experimental results in Section 6. The conclusion is provided in the last section.

1. The intuition

1.1. Detecting emerging new classes

The intuition is that anomalies of known classes are at the fringes of the data cloud of known classes, and instances of any emerging new classes are far from the known classes. To detect emerging new classes, we propose to treat instances of any new class as "outlying" anomalies which are significantly different from both instances and anomalies of the known classes.

The anomaly detector for the *SENC* problem must be able to differentiate between these two types of anomalies. The assumption is that anomalies of the known classes are more "normal" than the "outlying" anomalies. This is a reasonable assumption in this context because only instances of the known classes are available to train the anomaly detector.

An anomaly detector often categorises the feature space into two types of regions: anomaly and normal. Following the above idea, we propose to further subdivide each anomaly region into two sub regions: "outlying" anomaly sub region and anomaly sub region: (1) The instances in *anomaly sub region* is closer to the region of normal instances than instances from emerging new classes as the anomalies and normal instances are generated from the same distribution. (2) *"Outlying" anomaly sub region* is further away from the normal region and anomaly sub region. A test instance is regarded as belonging to an emerging new class if it falls in the "outlying" anomaly sub region.

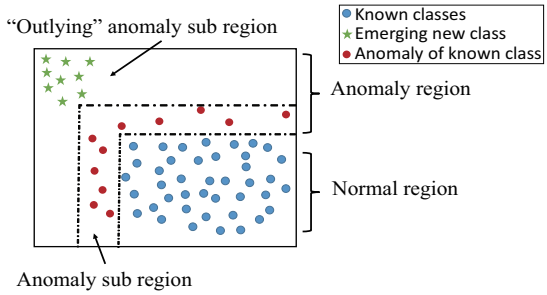


Figure 2: An illustration to build an "outlying" anomaly sub region

Figure 2 illustrates the normal and anomaly regions constructed by an anomaly detector. The anomaly region is further partitioned into two sub regions. The sub region outside the anomaly sub region is the "outlying" anomaly sub region.

The construction of "outlying" anomaly sub regions assumes that anomaly regions can be identified. We show in Section 4.2 that this can be easily achieved using a threshold of the anomaly scores provided by an anomaly detector to categorise all regions into two types: anomaly and normal.

1.2. Classification and efficient model update

If we treat the second sub problem, i.e., classification, as having no relation to the first sub problem for detecting emerging new classes, then any classifier can be applied. However, in order to facilitate efficient model update that enables classification of newly detected class and detection of more emerging new classes in data streams, we suggest an integrated approach which has a single common core for both the detection and classification tasks.

An unsupervised learner iForest[LTZ08], which induces completely random trees, has enabled us to implement the integrated approach with ease. This is because previous works [FWYM03, LTF05] have shown that, ensemble of completely random trees [Zho12, Chap.3.5], as an extreme case of variable-random trees [LTYZ08], can be successfully applied as a powerful classifier. We use exactly the same completely random trees, generated for the purpose of anomaly detection, for classification. This can be easily achieved by simply recording the class labels (provided in the training set) in each leaf. This is the only additional step that needs to be done in the training process to produce an ensemble of completely random trees that will act as both an unsupervised

learner (to detect emerging new classes) and an supervised learner (to classify known classes) in data streams.

As the single core for both tasks is completely random trees only, they can be updated easily when a sufficient number of instances of emerging new classes have been detected. The single core also facilitates to maintain the model complexity in a reasonable size by using effective model retiring mechanism and growing mechanism in the model update process.

In a nutshell, we introduce a simple and unique method to solve the *SENC* problem and show that the proposed method can detect emerging new classes and classify known classes with high accuracy, and perform model update efficiently in data streams. Our empirical evaluation shows that it often performs significantly better than existing more complex methods.

2. Related work

The *SENC* problem has the following challenges:

1. In the extreme case, no true labels except in the initial training set, i.e, true labels are not available after the model deployment.
2. A prediction must be made immediately for each incoming instance in the stream.
3. Store no data permanently from the data stream.
4. Fast model update.

Note that, as far as we know, there is no an algorithm that using one single core to conquer the whole *SENC* challenges. We review the related work with respect to these challenges as following.

Class-incremental learning (C-IL) [ZC02] is a branch of incremental learning which modifies a previously trained classifier to deal with emerging new classes. It has been found to be useful in various applications, e.g., detecting bots [CRT11], face recognition [HAY+07] and video concept detection[YYH07]. C-IL problems includes open set recognition[SdRRSB13], Learning with Augmented Class (LAC)[DYZ14]. All of these works are in the batch mode setting. The *SENC* problem is a C-IL problem in the data stream context.

In addition, many existing methods treat the *SENC* problem as a classification problem. This is the reason why they have employed supervised learning or semi-supervised learning approaches. Moreover, most of these studies assume that instances of an emerging new class are identified by some other mechanism and focuses on methods to train and incorporate classifiers which can classify new classes incrementally with previously trained classifiers[DYZ14, KOC13]. As a result, no existing methods in C-IL meet the four challenges mentioned above.

Learning with Augmented Class (LAC) [DYZ14] is a new effort for C-IL and addresses a research gap, i.e., to produce a detector for emerging new classes. Utilising unlabelled instances through semi-supervised learning, LACU-SVM [DYZ14] modifies a previously trained classifier to identify emerging new classes. Assuming the set of unlabelled instances containing sufficient instances of an emerging new class, a trained LACU-SVM can then assign a test instance to either one of the known classes or emerging new class. While it solves the first and second sub problems, it is a batch-mode method that requires to store all training data. Thus, it is not suitable in data streams and does not meet the four challenges.

The aim of *novel class detection* is to identify new data which are not previously seen by a machine learning system during training. This is the first sub problem of *SENC*. An example of this work in Bioinformatics [SdC04] employs an one-class SVM approach to detect novel classes. It is interesting to note that this approach does not make a distinction between novel class detection and *anomaly detection* (or outlier detection) [CBK09], which is the identification of items, events or observations which do not conform to an expected pattern in a data set in batch mode. It thus also does not meet the four *SENC* challenges in data streams.

The goal of *change point detection* is to detect changes in the generating distributions of the time-series. Many works have been conducted to tackle this problem [BN96] which include parametric methods [DDD05] and non-parametric methods [BD93]. This problem is equivalent to the first sub problem in *SENC*, without addressing the classification and model update issues. Yet, others have focused on classification in data streams [BHP$^+$09, JA03, KM07], without addressing the emerging new classes problem.

Another related work, ECSMiner, [MGK$^+$11] tackles the novel class detection and classification problems by introducing time constraints for delayed classification. ECSMiner assumes that true labels of new emerging class can be obtained after some time delay; otherwise, models cannot be

updated. In contrast, our proposed method assumes that no labels are available for the entire duration of a data stream.

The *SENC* problem can be solved by treating the first two sub problems independently by using existing methods, i.e., a new class detector and a known classes classifier. To detect emerging new class, existing anomaly detectors (such as LOF [BKNS00], iForest [LTZ08] and one-class SVM [MP03]) can be employed; and multi-class SVM [CL11]) can be used as an the classifier for known classes. In addition, existing supervised or semi-supervised batch classification methods can be adapted to solve the *SENC* problem, e.g., One-vs-rest SVM [RK04] and LACU-SVM [DYZ14].

However, all these algorithms do not solve the *SENC* problem satisfactorily. Table 1 summarizes the ability of these algorithms and the proposed *SENCForest* to meet the four challenges.

Table 1: Ability of algorithms to meet the challenges of the *SENC* problem.

Algorithm	Challenge			
	1	2	3	4
LOF+SVM	×	✓	×	×
1SVM+SVM	×	✓	×	×
One-vs-rest SVM	×	✓	×	×
LACU-SVM	×	✓	×	×
iForest+SVM	×	✓	×	✓
ECSMiner	×	×	✓	✓
SENCForest	✓	✓	✓	✓

Details about those algorithms implemented and the proposed *SENCForest* are provided in following sections.

SENCForest is the only one which can meet all four challenges. Only ECSMiner, among existing algorithms, can meet Challenge #3. Note that all existing algorithms assume that true labels are made available after the model deployment at some points in time—unable to meet Challenge #1.

3. Terminology Definition

Before introducing the detail of our proposed algorithm, we will give the formal definitions of many important concepts used in this paper.

Definition 3.1 *Classification under Streaming Emerging new Class (SENC) problem: Given a training data set* $D = \{(x_i, y_i)\}_{i=1}^{L}$, *where* $x_i \in R^d$ *is a training instance and* $y_i \in Y = \{1, 2, \ldots, K\}$ *is the associated class label. A streaming data* $S = \{(x'_t, y'_t)\}_{t=1}^{\infty}$, *where* $x' \in R^d$, $y' \in Y' = \{1, 2, \ldots, K, K+1, \ldots, M\}$ *with* $M > K$. *The goal of learning with the SENC problem is to learn a model* f *with* D *initially; then* f *is used as a detector for emerging new class and a classifier for known class.* f *is updated timely such that it maintains accurate predictions for known and emerging new classes on streaming data* S.

The *SENC* problem can have different variations. The hardest condition is when true class labels are not available throughout the entire process, except that the initial training set of known classes is used to train a classifier (and a detector for new classes) at the beginning of the data stream. A relaxation of this condition produces easier *SENC* problems. For example, true class labels are available at some intervals in streaming data S. In this paper, we show that the proposed method can deal with the hardest condition (in Section 5.2) as well as some easier conditions (in Section 5.3).

Definition 3.2 *Scores for test instances: Model* f *yields a score for a test instance* x, *which determines* x *as belonging to either a known class or an emerging new class (i.e., an "outlying" anomaly.)*

Definition 3.3 *Known Class Region and Anomaly Region: Based on the score from* f, *the feature space is divided into two types of regions : (a) known class regions* K *which have score* $\geq \hat{\tau}$, *(b) anomaly regions* A *which have score* $< \hat{\tau}$, *where* $\hat{\tau}$ *is a threshold.*

Definition 3.4 *Anomalies of Known Classes: Let* $\mathcal{O} = \{x_1, \ldots, x_n\}$ *be the training instances in an anomaly region* A. *The center of* $\mathcal{O}$ *is defined as* $c = \frac{1}{n}\sum_{x \in \mathcal{O}} x$. *Let* $e \in \mathcal{O}$ *be the farthest instance from* c. *A ball* B *centered at* c *with radius* $r = dist(c, e)$ *is an anomaly sub region. Instances which fall into anomaly sub regions are Anomalies of Known Classes.*

Definition 3.5 *Instances of an emerging new class are "outlying" anomalies:* $Q = A \backslash B$.

4. The Proposed Algorithm

In this section, we propose an efficient algorithm to deal with the *SENC* problem named *SENC-Forest* which is composed of *SENCTrees* and assigns each instance, as it appears in a data stream, a class label: Emerging New Class or one of the known classes. Instead of treating it as a classification problem, we formulate it as a new class detection problem and solve it using an unsupervised anomaly detector as the basis to build *SENCForest* which will finally act as both unsupervised learner and supervised learner.

We provide an overview of the procedure in section 4.1. The pertinent details in the procedure are then provided in the following three sections.

4.1. SENCForest: An Overview

SENCForest has four major steps:

1. Train a detector for emerging new classes. Given the initial training set of known classes D, an unsupervised anomaly detector *SENCForest* is trained, ignoring the class information, as follows:

1. Build an iForest [LTZ08].
2. Determine the path length[LTZ08] threshold $\hat{\tau}$.
3. Within each region A, construct ball B which covers all training instances which fall into this region. The area of the ball B is anomaly sub region A. Any test instances which fall into B are regarded as anomalies of known classes; those that fall outside B are regarded as instances from an emerging new class.

The path length is introduced in iForest[LTZ08], which can be regard as an anomaly score for determining known class region and anomaly region(like Definition 3.3). After training the detector of *SENCForest*, model *SENCForest* can yields a new class score for a test instance x through aggregating results of each tree in *SENCForest*. Detail of iForest will be described in the following section.

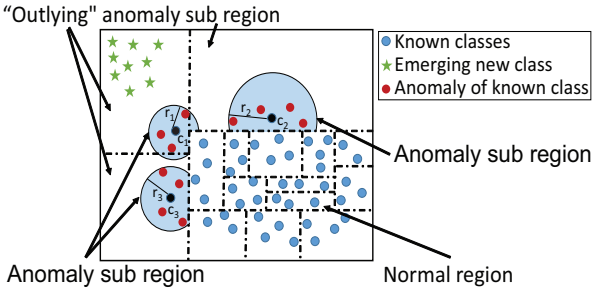


Figure 3: An illustration to build an "outlying" anomaly sub region

Figure 3 illustrates the regions constructed by an iTree which has axis-parallel boundaries, and the additional subdivision employs a ball to partition each anomaly region into two sub regions. The anomaly sub region outside the ball is the "outlying" anomaly sub region.

2. Using known class information to build a classifier from a detector. Once the above new class detector is constructed, class distributions based on known class labels are recorded in each K or B region. Each region with class distribution acts as a classifier that outputs the majority class as the classification result for a test instance which fall into the region.

The training set is discarded once the training process is completed.

3. Deployment in data stream. *SENCForest* is now ready to be deployed in a data stream, and it is assumed that no true class labels are available for model updated throughout the entire data stream. An instance in the data stream is given a class prediction by *SENCForest* if it falls into K or B region; otherwise, it is identified as an instance from an emerging new class and placed in a buffer of size s.

4. Model update. The model update process in *SENCForest* is simple. It begins when the buffer is full. Using instances from the buffer, the same tree growing process is then applied to each leaf of every existing tree until the stopping criterion is satisfied. The rest of the model update process follows the same steps from 1.2 onwards, as described above. Note that the update largely involves newly grown subtrees, i.e., replacing leaf nodes which have the number of instances more than a set limit after taking new instances from the buffer into consideration. Thus, the whole process can be completed quickly. To maintain model size, mechanisms to retire *SENCForest* are also employed in the model update process.

Section 4.2 describes the pertinent details of training *SENCForest* as both unsupervised detector and supervised learner. Deploying *SENCForest* and model update in data streams are provided in Section 4.3 and Section 4.4, respectively.

4.2. SENCForest: Training process

The training procedure to build an *SENCForest* with both detection and classification functions is detailed in Algorithms 1 and 2. These are the combined step to build iForest[LTZ08] and to produce a classifier from a detector. The trees are then used to determine the path length threshold and to construct "outlying" anomaly regions described following respectively. Note that the procedure is the same as in building iForest, except in line 2 of Algorithm 2. As the trees constructed are not exactly iTrees, we name the trees with the new classification capability, *SENCTrees*.

Build an iForest. The unsupervised anomaly detector *iForest* [LTZ08] is an ensemble of *Isolation Tree (iTrees)*. "Isolation" is a unique concept in anomaly detection, as each iTree is built to isolate every instance from the rest of the instances in the training set. The idea is based on the fact that since anomalies are 'few' and 'different', they are more susceptible to isolation than normal instances. Hence, an anomaly can be isolated using fewer partitions in an iTree than a normal instance.

Liu et. al. [LTZ08] show that iTrees can be created using a completely random process to achieve the required isolation. Given a random subsample of size ψ, a partition is produced by randomly selecting an attribute and its cut-point between the minimum and maximum values in the subsample. To produce an iTree, the partitioning process is repeated recursively until every instance in the subsample is isolated. An iForest is an ensemble of z iTrees, each generated using a subsample randomly selected from the given training set.

In the testing process, an instance having a short path length, which is the number of edges it traversed from the root node to a leaf node of an iTree, is more like to be an anomaly. The average path length from all iTrees is used as the anomaly score for each test instance.

For both instances of emerging new class and anomalies of known classes, iForest will produce short path lengths because they all are individually 'few' and 'different' from the known classes.

In order words, they are all in the regions with short path length in iTrees. We called this type of region, anomaly region A to differentiate them from normal region K which have long path length.

In order to detect emerging new class, we first need to determine a path length threshold to differentiate A from K. Then, build a sub region B in each A region which covers all training instances in the region. As these instances are from known classes, they are anomalies of known classes. These two processes are described in the following paragraph.

Determine the path length threshold. As each region in iTree has its own path length, and anomaly regions A are expected to have shorter path length than that from normal regions K, we employ the following method to determine the path length threshold to separate these two types of regions.

We produce a list L which orders all path lengths representing all regions in an iTree in ascending order. A threshold τ in this list yields two sub-lists L^l and L^r. To find the best threshold, we use the following criterion which minimises the difference in standard deviations $\sigma(.)$:

$$\hat{\tau} = \arg\min_{\tau} \ |\sigma(L^r) - \sigma(L^l)|$$

The threshold $\hat{\tau}$ is used to differentiate anomaly regions A from normal regions K, where the former has low path length and the latter has long path length.

Using a tree, Figure 4 shows an example of cumulative distribution for list L and its SD_{diff} ($= |\sigma(L^r) - \sigma(L^l)|$) curve. Note that the minimum SD_{diff} point separates into two clear regions: anomaly and normal regions.

Note that (i) because threshold $\hat{\tau}$ is determined automatically, no additional parameter is introduced; and (ii) this process does not require training data.

Construct "outlying" anomaly sub regions. After $\hat{\tau}$ is determined, a ball B is constructed using all training instances in every region A of a tree, according to Definitions 3.4 and 3.5.

When balls B have been built for all A regions in every *SENCTree*, the *SENCForest* has the first function as an unsupervised detector and is ready to detect instances of emerging new classes.

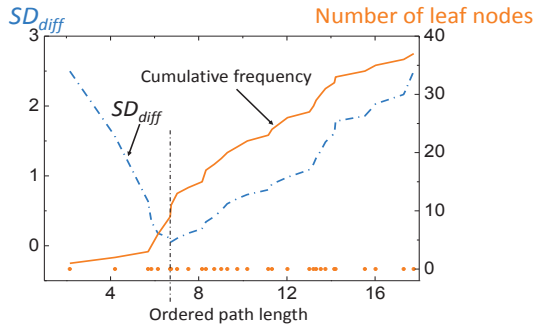


Figure 4: Determining path length threshold: Cumulative frequency and the corresponding SD_{diff} curve, where the x-axis is the ordered path lengths from all regions in an iTree. The point which yields the minimum SD_{diff} is chosen as the threshold to differentiate anomaly regions A from normal regions K.

A test instance which falls into $\mathcal{A}$ but outside B is an "outlying" anomaly, i.e., an instance of an emerging new class.

Produce a classifier from a detector To incorporate the second function of being a classifier into *SENCForest*, all we have to do is to record class distribution $F[j]$ in each region from K and B using the training subsample, where $F[j]$ denotes the number of class j instances in a region. Note that this is the only step class labels are required.

Once the above training steps are completed, *SENCForest* is ready to be deployed to a data stream.

Algorithm 1 Build *SENCForest*

Input: D - input data, z - number of trees, ψ - subsample size.

Output: *SENCForest*

1: **initialize:** $SENCForest \leftarrow \{\}$
2: **for** $i = 1, \ldots, z$ **do**
3: $\quad X_i \leftarrow sample(D, \psi)$
4: $\quad SENCForest \leftarrow SENCForest \cup SENCTree(X_i)$
5: **end for**

4.3. Deployment in data stream

Given a test instance x, $SENCForest(x)$ produces a class label $y \in \{b_1, \ldots, b_m, NewClass\}$, where m is the number of known classes thus far and $NewClass$ is the label given for an emerging new

Algorithm 2 *SENCTree*

Input: X - input data, $MinSize$ - minimum internal node size

Output: *SENCTree*

1: **if** $|X| < MinSize$ **then**
2: return LeafNode$\{|X|, F[\cdot], c, r\}$, as defined in Section 4.2.
3: **else**
4: let Q be a list of attributes in X
5: randomly select an attribute $q \in Q$
6: randomly select a split point p from max and min values of attribute q in X
7: $X_L \leftarrow filter(X, q \leq p)$
8: $X_R \leftarrow filter(X, q > p)$
9: return inNode{Left $\leftarrow$ *SENCTree*(X_L),
10: Right $\leftarrow$ *SENCTree*(X_R),
11: SplittAtt $\leftarrow q$,
12: SplittValue $\leftarrow p$ },
13: **end if**

class. Note that though *SENCForest* can detect instances of any number of emerging new classes, they are grouped into one new class for the purpose of model update. We will focus on model update on one new class in one period (but multiple new classes could emerge in different periods of a data stream) for the rest of the paper. We discuss the issue of model update for multiple new classes in Section 5.4.

Algorithm 3 describes the testing process during the deployment of *SENCForest* in a data stream.

In line 3 of Algorithm 3, *SENCForest*(x) outputs the majority class among all classes produced from z trees. A tree outputs $NewClass$ if test instance x falls into an A region but outside the B region; otherwise, it outputs the majority of class from

$$\arg\max_{j \in \{b_1, \ldots, b_m\}} F[j]$$

where $F[j]$ is the class frequency for class j recorded in the region (K or B) into which x falls.

If *SENCForest*(x) outputs *NewClass*, x is placed in buffer $\mathcal{B}$ which stores the candidates of the previously unseen class (line 5). When the number of candidates has reached the buffer size, the

candidates are used to update both the classifier and the detector (line 7). Once these updates are completed, the buffer is reset and the new model is ready for the next test instance in the data stream.

Algorithm 3 Deploying *SENCForest* in data stream

Input: *SENCForest*, $\mathcal{B}$ - buffer of size s

Output: y - class label for each x in a data stream

```
1: while not end of data stream do
2:   for each x do
3:     y ← SENCForest(x)
4:     if y = NewClass then
5:       B ← B ∪ {x}
6:       if |B| ≥ s then
7:         Update (SENCForest, B)
8:         B ← NULL
9:         m ← m + 1
10:      end if
11:    end if
12:    Output y ∈ {b_1, ..., b_m, NewClass}.
13:  end for
14: end while
```

4.4. Model Update

4.4.1. Growing Mechanism

There are two growing mechanisms: one for growing a subtree in an *SENCTree*, and the other for the growing multiple *SENCForests*.

Growing a subtree in an *SENCTree*. Updating *SENCForest* with buffer $\mathcal{B}$ is a simple process of updating each leaf node in every tree using ψ instances, randomly selected from $\mathcal{B}$. This is depicted in Algorithm 4. The update at each node (line 10) involves either a replacement with a newly grown subtree or a simple update of the class frequency to include the new class b_{m+1}.

Algorithm 4 Update *SENCForest*

Input: *SENCForest* - existing model, $\mathcal{B}$ - input data

Output: a new model of *SENCForest*

initialize: All instances in $\mathcal{B}$ are assigned a new class b_{m+1}
for $i = 1, ..., z$ **do**
  $\mathcal{B}' \leftarrow sample(\mathcal{B}, \psi)$
  Tree $\leftarrow$ *SENCForest.*Tree$[i]$
  for $j = 1, ...,$Tree.LeafNodeNumber **do**
    $X' \leftarrow$ instances of $\mathcal{B}'$ which fall into Tree.LeafNode$_j$
    if $|X'| > 0$ **then**
      $X \leftarrow$ Pseudo instances from Tree.LeafNode$_j$
      $X' \leftarrow X' \cup X$
      Tree.LeafNode$_j \leftarrow$ *SENCTree*(X')
    end if
  end for
  recalculate $\hat{\tau}$ for Tree
  *SENCForest.*Tree$[i] \leftarrow$ Tree
end for

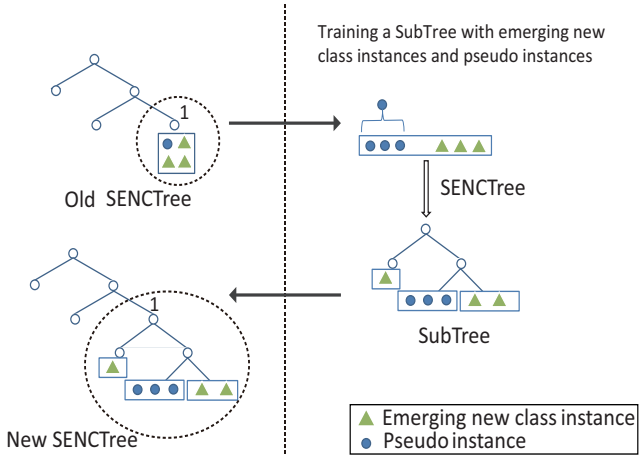


Figure 5: Replacing a leaf node with a trained subtree

If there are some instances which fall into a leaf node, a subtree needs to be grown as follows. As the previous training set is not stored, pseudo instances are generated for the leaf node which have the same attribute-values as centre c. The number of pseudo instances for each class j is as recorded in $F[j]$. The combined set of pseudo instances X and X' (i.e., the subset of $\mathcal{B}'$ which falls into the same leaf node) is used as input to *SENCTree* (line 10). An example procedure is depicted in Figure 5. In the top left figure, we assume that some emerging new class instances (green triangle) fell into node 1 (there are three instances fell into in training process) in an *SENCTree*. Then the combined set consists of pseudo instances and instances of the emerging new class. A new *subTree* is built by using the combined set. Finally, in the bottom left figure, node 1 is replaced with this new *subTree*. Every leaf node goes through the same process.

Note that the update process retains the original tree structure, and all pseudo instances in a leaf node will still be placed into a single leaf node of the newly grown *subtree*. Thus, the predictions for the known classes are not altered in the model update process.

Once each tree has completed the model update, $\hat{\tau}$ is recalculated as described in Section 4.2.

Growing multiple *SENCForests*. When the number of classes in a *SENCForest* reaches ρ, its *SENCTrees* will stop growing for any emerging new class. A new *SENCForest* is grown instead for the next ρ emerging new classes. This user-defined parameter is set based on the memory space available.

4.4.2. Prediction using Multiple SENCForests

In a model with multiple *SENCForests*, the final prediction is resolved as follows. For a given x, *SENCForest* i yields prediction y_i and probability

$$p_i = \frac{\text{Number of } \textit{SENCTrees} \text{ predicting } y_i}{\text{Total number of } \textit{SENCTrees}}$$

The final prediction is $NewClass$ only if all *SENCForests* predict x as belonging to $NewClass$. Otherwise, the final prediction is the known class which has the highest p_i. This procedure is given in Algorithm 5.

Algorithm 5 Final Prediction from E *SENCForests*

Input: x - an instance in the data stream

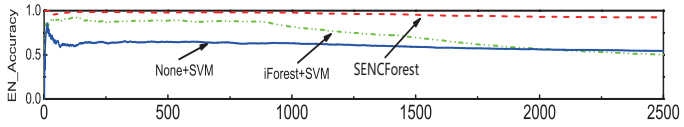
Output: $y_\imath$ - class label for x

1: **for** $i = 1, ..., E$ **do**
2: $\langle y_i, p_i \rangle \leftarrow SENCForest_i(x)$
3: **end for**
4: **if** $\forall_i \;\; y_i = NewClass$ **then**
5: $\imath \leftarrow 1$
6: **else**
7: $L \leftarrow \{i \in \{1, \ldots, E\} \mid y_i \neq NewClass\}$
8: $\imath \leftarrow \arg\max_{i \in L} \; p_i$
9: **end if**
10: Output $y_\imath$

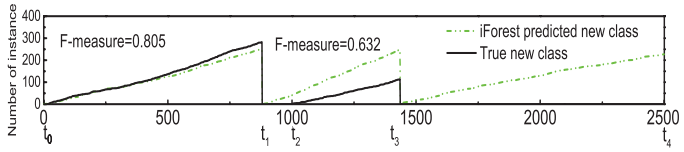
4.4.3. Retiring Mechanism

A mechanism to retire *SENCForest* is required as the data stream progresses. A *SENCForest* is retired under the following scenarios:

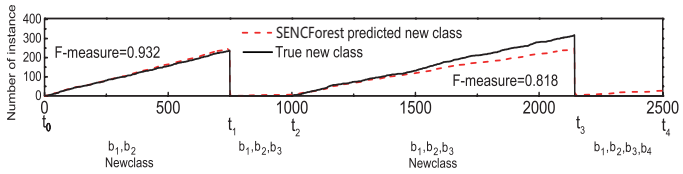
1. When a *SENCForest* is not used for predicting known classes for a certain period of time, it is eliminated for any future predictions. In other words, a *SENCForest* outputs "NewClass" for a long time, this *SENCForest* will be retired



(a) Classification result in a data stream



(b) Emerging new class detection result of iForest+SVM



(c) Emerging new class detection result of ENCiFer

Figure 6: An example data stream on the KDDCUP 99 data set. The x-axis is the time steps in the data stream. The known classes at each duration (t_i - t_{i+1}) are denoted as b_1,b_2,b_3, and b_4. The details of the two methods, iForest+SVM and None+SVM, are described in Table 2.

2. In the event that the number of *SENCForests* has reached the preset limit ρ and no *SENCForest* can be retired based on (1), then the least used *SENCForest* in the last period is chosen to retire.

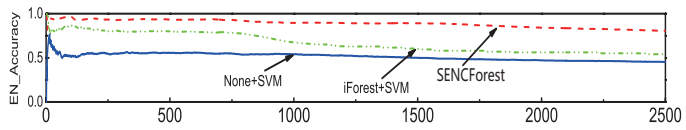
The number of known class predictions is recorded for each *SENCForest* in data stream. The one which has made the minimum number of predictions for known classes is identified to be the least used *SENCForest*.

5. Experiment

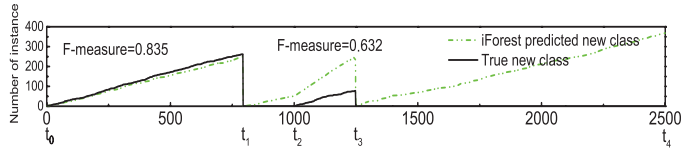
This section reports the empirical evaluation we have conducted to assess the performance of *SENCForest* in comparison with several state-of-the-art methods.

5.1. Experimental Setup

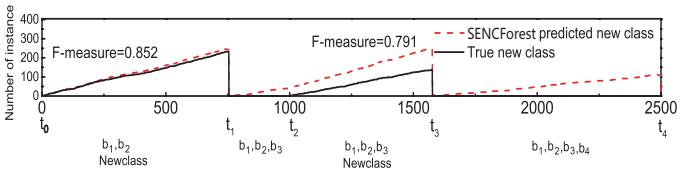
Data Stream: To simulate emerging new classes in a data stream, we assume that an initial training set with two known classes are available to train the initial models. When the trained



(a) Classification result in a data stream



(b) Emerging new class detection result of iForest+SVM



(c) Emerging new class detection result of ENCiFer

Figure 7: An example data stream on the MNIST data set.

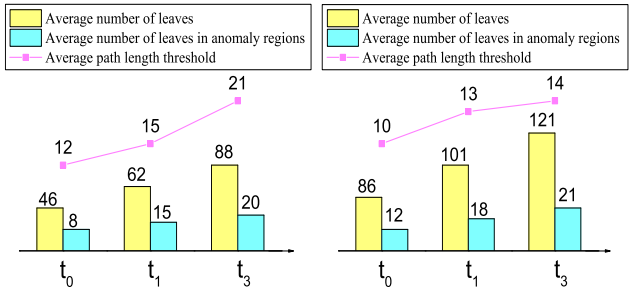
models are deployed at the beginning of a data stream, instances of the two known classes and an emerging new class appear in the first period of the data stream with uniform distribution. It is assumed that the method employed will update its models sometime within the first period. In the second period, instances of the three classes seen in the first period and another emerging new class appear with uniform distribution. Instances appear one at a time, and the deployed method is expected to make a prediction for each instance before processing the next, i.e., each instance is predicted as belonging to either an emerging new class or one of the known classes thus far.

No true class labels for all instances are available throughout the entire data stream.[1] Model update is based on the instances of the emerging new class identified at the time the model update is triggered.

Figures 6 and 7 show example data streams using the KDDCUP 99 data set and the MNIST data set. The class composition in the two distinct periods in the data stream are described as follows:

[1]This is a more stringent condition than previous studies (e.g., [MGK+11]) which assume that true labels are available for model update, after some time delay.

		Known classes	New class
First period: t_0 - t_2	t_0 - t_1	b_1,b_2	✓
	t_1 - t_2	b_1,b_2,b_3	×
Second period: t_2 - t_4	t_2 - t_3	b_1,b_2,b_3	✓
	t_3 - t_4	b_1,b_2,b_3,b_4	×



(a) (b)

Figure 8: Information of the evolving *SENCForest* at three different times in the data stream on (a) KDDCUP 99 data set; (b) MNIST data set.

In the first period, all instances of the emerging new class identified by a method is placed in a buffer $\mathcal{B}$ of size s. When the buffer is full (marked as t_1), the method updates its model before processing the next instance. Note that t_1 differs for different methods as their detection rates for the new class are different, as shown in Figures 6(b) and 6(c) for iForest+SVM and *SENCForest* (so as in Figures 7(b) and 7(c).) The buffer is reset to be empty when the model of a method has been updated. Note that after the model is updated, the new class in t_0 - t_1 becomes a known class b_3 of the updated model in t_1 - t_2, as shown in the table above.

Similarly, in the second period between t_2 and t_4, t_3 is the time when the buffer is full and the model of a method is updated for the second time. The new class in t_2 - t_3 becomes a known class b_4 of the updated model in t_3 - t_4.

Figure 8 shows the information of the evolving *SENCForest* at three different times in the data stream on two data sets.

Evaluation measures: To evaluate the predictive accuracy of algorithms in the *SENC* problem, we introduce EN_Accuracy in a fixed window size. Let N be the total number of instances in a window; A_n be the total number of emerging class instances identified correctly; and A_o be the

Table 2: Methods used in the empirical evaluation. D is the training set for the current models; $\mathcal{B}$ is the set of new class instances in the buffer and model update is triggered when the buffer is full. After each model update, $D \leftarrow D \cup \mathcal{B}$; and D needs to be stored for the next model update for all methods, except *SENCForest* and None+SVM. U is an additional set of unlabelled instances used by LACU-SVM only. In the experiments, the data size of U is the total data size of D and $\mathcal{B}$.

Method	Detection	Classification	Model Update
LOF+SVM	LOF	multi-class SVM	train new LOF and SVM with $D \cup \mathcal{B}$
1SVM+SVM	one-class SVM	multi-class SVM	train new 1SVM and SVM with $D \cup \mathcal{B}$
1R-SVM	One-vs-rest SVM		train new 1R-SVM with $D \cup \mathcal{B}$
LACU-SVM	LACU-SVM		train new LACU-SVM with $D \cup \mathcal{B}$ and U
ECSMiner	ECSMiner		train a new classifier in each fixed interval, assuming true labels are given
iForest+SVM	iForest	multi-class SVM	train new iForest and SVM with $D \cup \mathcal{B}$
SENCForest+SVM	*SENCForest*	multi-class SVM	Update *SENCForest* with $\mathcal{B}$ and train new SVM with $D \cup \mathcal{B}$
None+SVM	No detector	multi-class SVM	no model update
SENCForest	*SENCForest*		Update *SENCForest* with $\mathcal{B}$

total number of known class instances classified correctly,

$$EN\_Accuracy = \frac{A_n + A_o}{N}$$

Figures 6(a) and 7(a) show examples of EN_Accuracy results of three methods in a data stream.

To evaluate the accuracy of new class detection, we compute F-measure in t_0 - t_1 and t_2 - t_3 to measure the detection performance in these two durations. This measure produces a combined effect of precision (P) and recall (R) of the detection performance. F-measure = 1 if a detector identifies all instances of emerging new class with no false positives.

$$F\text{-}measure = \frac{2 * P * R}{P + R}$$

The cumulative numbers of instances of the true and predicted new class are also plotted in four consecutive durations. In t_1-t_4, it shows that both methods make some false positives resulting in more instances predicted as belonging to the new class than it actually has. The F-measures achieved by each detection method in t_0-t_1 and t_2-t_3 are shown in Figures 6(b) & 6(c) and Figures 7(b) & 7(c). In this example, *SENCForest* performs better than iForest+SVM because it has better F-measure, fewer false positives and higher EN_Accuracy.

In the experiments reported in Section 5.2, the difference in performance between two methods is considered to be significance on paired t-tests at 95% significance level in our paper

Contenders: The complete list of the methods used for new class detection, classification and model update methods is shown in Table 2. As some of these methods can act as a new class detector only, a state-of-the-art classifier, i.e., multi-class SVM [CL11], is employed to classify instances of known classes. Note that three types of information, additional to that was provided to *SENCForest*, are required for other methods. First, true labels must be provided at each model update. Otherwise, no models could be updated. ECSMiner assumes that true labels are given at the end of a fixed interval (T_l) in order to update model. Other existing methods requires all instances in $\mathcal{B}$ must be given the true labels. Second, LACU-SVM needs to have additional unlabelled data before training at each model update. Third, the initial training set must be stored and incorporated at each model update. *SENCForest* is the only method which does not require (i) true labels during the entire data stream after training, (ii) to store the initial training set, and (iii) unlabelled training set.

A brief description of each of the methods used in the experiment is given as follows:

1. **LOF** or Local Outlier Factor [BKNS00] is a density-based anomaly detector which employs k-nearest neighbour procedure to estimate density.

2. **One-class SVM** [SPST$^+$01] is a state-of-the-art outlier detector [MP03] which learns from normal instances only. It computes a binary function to capture regions in input space where the probability density lives.

3. **One-vs-rest SVM** is a scheme for multi-class classification [RK04] where a two-class SVM $f_k(\cdot)$ is built for each class. In the original One-vs-rest SVM, a test instance x is predicted as belonging to class k if $f_k(\cdot)$ produces the highest confidence. To adapt One-vs-rest SVM to predict the emerging new class, the classifier produces a classification prediction only if $\max_k f_k(x) > 0$; otherwise x is predicted as belonging to the emerging new class.

4. **LACU-SVM** [DYZ14] is a semi-supervised learner which modifies a previously trained model by considering the structure presented in the unlabelled data so that the misclassification risks among the known classes as well as between the new and the known classes are minimized simultaneously. It produces a classifier which predicts one of the known classes or the new class. This method also trains k binary classifiers $f_k(\cdot)$ for each known class. Like

One-vs-rest SVM, LACU-SVM makes a prediction for the known class if $\max_k f_k(x) > 0$; otherwise x is predicted as belonging to the emerging new class.

5. **ECSMiner** [MGK$^+$11] is an algorithm for novel class detection and classification. It employs the clusters identified by k-means to detect novel classes: instances which are not within the boundaries of any clusters are treated as novel class candidates and placed in a buffer, then a new measure is defined to decide whether they are emerging new classes. K nearest neighbor is used as the classifier to make predictions for instances of known classes. Model update can only occurs if true labels are available within some fixed duration.

6. **iForest** [LTZ08] is an unsupervised anomaly detector which builds a model to isolate each training instance from the rest of the training set.

In the experiments, all methods were executed in the MATLAB environment. The following implementations are used: SVM in the LIBSVM package [CL11]; LACU-SVM and iForest were the codes as released by the corresponding authors; and LOF is in the outlier detection toolbox.[2] The ECSMiner code is completed based on the authors' paper [MGK$^+$11]. We set the max size of each tree to 300, which avoids to the worst case that growing infinitely by random partition. The parameter settings used for these algorithms are provided in Table 5 in Appendix A.

Data sets: Five data sets are used to assess the performance of all methods , including Synthetic, KDDCup 99[3], Forest Cover[4], MHAR and MNIST[5]. For KDDCup 99 data set, we use the four largest classes, i.e., normal, neptune, smurf and back. For Forest Cover data set, we use 10 attributes, and all binary attributes are removed. A description for Synthetic and MHAR data sets are provided in Appendix B. A summary of the data characteristics is provided in Table 3.

Simulation: In the following experiment, each data set is used to simulate a data stream over ten trials. In each trial, the initial training set has two classes, and the emerging new class in each period is a class different from the known classes. These classes are randomly selected from the available classes. The instances in the initial training set and the data sequence in the data

[2]https://goker.wordpress.com/2011/12/30/outlier-detection-toolbox-in-matlab/

[3]http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

[4]https://kdd.ics.uci.edu/databases/covertype/covertype.data.html

[5]http://cis.jhu.edu/ sachin/digit/digit.html

Table 3: A summary of data sets used in the experiments.

Data set	#classes	#attributes
Synthetic	4	2
KDDCup 99	4	41
Forest Cover	7	10
MHAR	6	561
MNIST	10	784

stream are randomly selected from the given data set, but following uniform class distribution. For all real-world data sets, the data size of the initial training set D is 500 per class; the buffer size $|\mathcal{B}| = 250$; and the total number of instances which have appeared in the data stream at the end of the first period at t_2 is 1000; and the second period (t_2 - t_4) has a total of 1500 instances. As we can afford to generate more data in the synthetic data set, D, $\mathcal{B}$, and the data size at each period are double to examine the effect of larger data sizes. The average result of ten trials is reported.

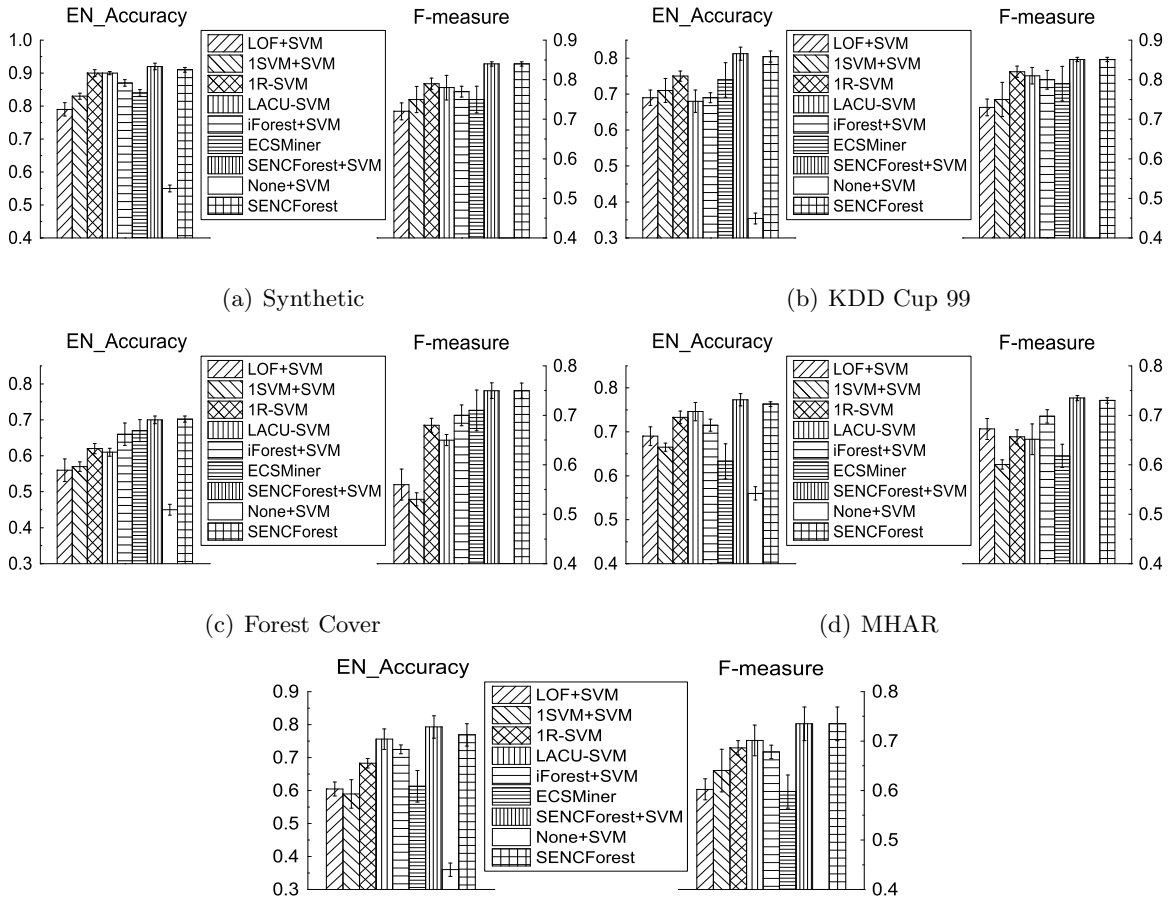
The following sections will give related evaluation results. Section 5.2 describes the empirical evaluation under the condition that no true labels are available after the data stream has started. Section 5.3 reports results under the long streams situation. Section 5.4 describes using *SENCForest* under the condition that emerging multiple new classes in a period.

5.2. Empirical results

The results for the five data sets are shown in Figure 9.

In terms of new class detection, *SENCForest* produced the highest F-measure in all data sets. Recall that *SENCForest*+SVM uses *SENCForest* only for new class detection; thus both *SENCForest* and *SENCForest*+SVM have the same F-measure performance.

The closest contenders are LACU-SVM and 1R-SVM, each had the second or third highest F-measure in three data sets. *SENCForest* was significantly better than all contenders, except in MNIST (wrt LACU-SVM) and Forest Cover (wrt ECSMiner).



(a) Synthetic

(b) KDD Cup 99

(c) Forest Cover

(d) MHAR

(e) MNIST

Figure 9: Average result over ten trials. In each trial, the following is computed: Average F-measure in t_0 - t_1 and t_2 - t_3; and average accuracy over the entire duration from t_0 to t_4. Two standard errors over ten trials are shown as the error bar. Note that *SENCForest* and *SENCForest*+SVM are using the same detector to detect emerging new class; thus they have the equivalent F-measure result.

In terms of EN_Accuracy, *SENCForest* and *SENCForest*+SVM produced the highest performance in all data sets. This result shows that (i) the accurate detection of emerging new class leads directly to high classification accuracy; and (ii) *SENCForest* as a classifier is competitive to SVM. LACU-SVM was the closest contender which had the second highest accuracy in three data sets. Beside *SENCForest*+SVM, *SENCForest* performed significantly better than the other contenders in three data sets. The two exceptions are wrt to LACU-SVM (in MNIST and Synthetic) and 1R-SVM (in Synthetic).

An analysis is provided below:

- LOF and one-class SVM: the poor detection performance of these two methods wrt to

iForest is likely to be due to the parameter search, i.e., a search for a wider range of values may improve their performance. However, such search is a computationally expensive process, and this makes them unsuitable for data stream applications.

- iForest performed worse than *SENCForest* in all data sets, and the differences were significance in four data sets. This shows that an unsupervised anomaly detector can be successfully used in the *SENC* problem if anomaly regions are reshaped (as described in Sections 4.2) to detect emerging new classes.

- While One-vs-rest SVM performed reasonably well in classification, it is not a good choice for detection of emerging new classes, in comparison with *SENCForest*.

- LACU-SVM is the only method which requires additional unlabelled instances in training the initial model and in every model update. While obtaining unlabelled instances may not be a problem in real applications, it is important to note that its detection performance is highly depended on the existence of a new class in the set of unlabelled instances. Insufficient instances of the new class will severely limit LACU-SVM's ability to detect the new class. In the experiment, LACU-SVM was provided a set of unlabelled instances in t_0, t_1 and t_3, in addition to those instances in the initial training set and the buffer, in order to update its model. This additional data set was not available to all other methods. Despite this additional training information, LACU-SVM still performed significantly worse than *SENCForest* in four data sets in terms of F-measure.

- ECSMiner is the only algorithm which was provided with true labels in order to train a new classifier in each fixed interval, which occurs more often than at each model update, over the entire data stream. Despite this advantage, it still performed significantly worse than *SENCForest* in four out of five data sets in both measures.[6]

- The result of None+SVM clearly shows that not using a detector is not an option in the *SENC* problem.

- *SENCForest* is the best choice detector and a competitive classifier in the *SENC* problem.

[6]ECSMiner [MGK+11] had employed the KDD CUP 99 and Forest Cover datasets in their evaluation. Our ECSMiner results are compatible with theirs in these two datasets. However, ECSMiner performed poorly in the other three datasets.

While it is possible that a more sophisticated classifier may yield a higher accuracy in classifying known classes, it often comes at a high computational cost in an extensive parameter search.

- While using SVM, in addition to *SENCForest*, could potentially produce a better accuracy than that from *SENCForest* alone, this comes with a computational cost which is usually too expensive in the data streams context. Note that to achieve the performance of *SENCForest*+SVM presented in Figure 9, it needs to store all instances thus far, which is impossible in data streams. In contrast, *SENCForest* achieves comparable result as *SENCForest*+SVM without the need to store any data.

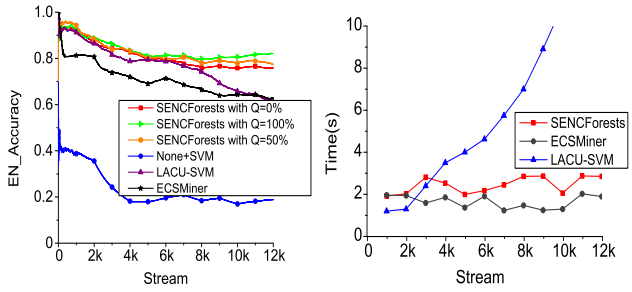
5.3. SENCForest *in long data streams*

The aims of this section are to examine the ability of *SENCForest* to (i) maintain good performance using limited memory in long data streams; and (ii) make use of true class labels when they are available.

We simulate a long data stream using the MNIST data set. This stream has twelve emerging new classes[7]. The initial training data set has 2 classes, and every subsequent period has 1000 instances from one emerging new class and two known classes. The maximum number of classes which can be handled by each *SENCForest* is set to 3. Other settings are the same as used in the last section. In addition, true class labels are assumed to be available in Q percentage of instances in the buffer before a model update. *SENCForest* with $Q = 0\%$, 50% and 100% are compared with LACU-SVM in the experiment. Recall that, as in the previous experiment, LACU-SVM is given 100% true labels at each model update and an additional set of unlabelled instances; and ECSMiner is also provided with 100% true labels at each model update.

Figure 11 shows the average number of leaves of each *SENCForest* at the start of each time period. Note that a new *SENCForest* was produced at periods 2, 5, 8, 11, and the first two *SENCForests*, A and B, were retired at periods 8 and 11, respectively. Table 4 shows the further

[7]Classes are reused in the simulation when they are no more in use in the current period. Because this simulation needs a number of classes, that is why only the MNIST dataset, out of the five datasets, can be used in the long stream simulation.



(a) (b)

Figure 10: (a) Result of long data stream in the MNIST data set; the bar chart on the bottom shows *SENCForests* with $Q = 0\%$, the number of *SENCForests* and the number of retired *SENCForests* at each period. (b) The time spent (in seconds) to do model update in each period.

information about *SENCForests*($Q = 0\%$) at the start of each time period. The first three rows provide the overall information; and the last three lines show the detailed information of the only evolving *SENCForest* at each time period, e.g., periods 2, 3, 4 for $SENCForest_B$, periods 5, 6, 7 for $SENCForest_C$ and so on. Note that the number of leaves in anomaly regions may decrease as *SENCForest* grows. This happens when instances of new classes fall into few leaves only.

The number of *SENCForests* is maintained at a preset memory limit through retiring not-in-use *SENCForests*. Note that the model size is constrained within the set limit of three SENCForests which allows the proposed method to deal with infinite data streams. In contrast, LACU-SVM continues to demand larger and larger memory size to accommodate larger training set size as the stream progresses.

The result in Figure 10 (a) shows that *SENCForest* with $Q = 0\%$ maintains good predictive accuracy over the long stream. *SENCForest* is able to make use of true class labels to improve its performance along the stream. The extent of the improvement increases as Q increases. In contrast, the predictive accuracy of ECSMiner and LACU-SVM continued to decrease as the stream progressed.

As a result, as shown in Figure 10(b), its training time continued to grow as the stream progressed. ECSMiner has the least model update time, because k-nearest neighbor is as base learner, which only spends time in building the clusters in buffer. But, that means it needs to save the cluster summary of each cluster into memory.

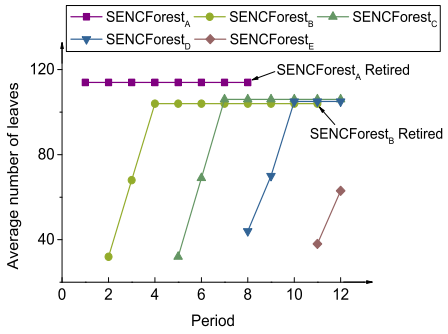


Figure 11: Average number of leaves of each evolving *SENCForest* at the start of each time period.

Table 4: Information of each evolving *SENCForest* (marked with *) at the start of each time period on the simulated data stream using MNIST. Note that only the latest *SENCForest* at any time period is evolving or growing; and all earlier built *SENCForests* (if any) have stopped growing. The subscript indicates the latest *SENCForest* shown in Figure 11.

Period	1	2	3	4	5	6	7	8	9	10	11	12
Number of known classes	3	4	5	6	7	8	9	7	8	9	7	8
Number of *SENCForests*	1	2	2	2	3	3	3	3	3	3	3	3
Number of retired *SENCForests*	0	0	0	0	0	0	0	1	0	0	1	0
Average number of leaves*	114_A	32_B	68_B	104_B	32_C	69_C	106_C	44_D	70_D	105_D	38_E	63_E
Average number of leaves in anomaly regions*	14_A	11_B	15_B	12_B	9_C	12_C	13_C	14_D	13_D	14_D	16_E	15_E
Average path length threshold*	16_A	8_B	16_B	17_B	10_C	17_C	18_C	11_D	17_D	20_D	11_E	15_E

5.4. Multiple new classes in a period

The emergence of multiple new classes in a period is a challenge in the *SENC* problem. Although *SENCForest* is designed to deal with one emerging new class in each period, it can still perform well by treating these emerging classes in a period as a single new class. Figure 12 shows that *SENCForest* performs as well when every period has two emerging new classes. In this stream, there are three periods; each period has 2000 instances and 4 classes (i.e., two emerging new classes and two known classes).

In the event that it is important to identify each class in each period, a clustering algorithm[Agg13] can be used to achieve this aim before proceeding to do the model update.

6. Conclusions and future work

This paper contributes to decompose the *SENC* problem into three sub problems and posits that the ability to tackle the first sub problem of detecting emerging new classes effectively is crucial

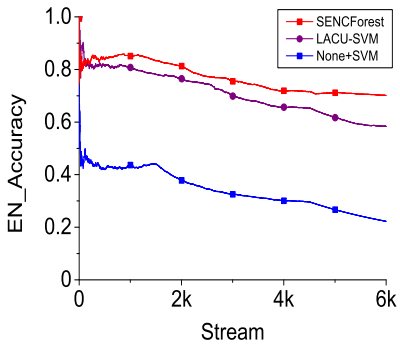


Figure 12: Result of two emerging new classes in each period.

for the whole problem. The difficulty of the *SENC* problem is highlighted by the inability of existing methods to solve it satisfactorily.

We show that the unsupervised-anomaly-detection-focused approach, coupled with an integrated method using completely random trees, provides a complete solution for the entire *SENC* problem. The current classification-focused approach has failed to provide one thus far.

The strength of *SENCForest* is its ability to detect new class with high accuracy. The use of an unsupervised anomaly detector, incorporated with the new ability to differentiate between anomalies of known classes and instances of new classes, underlines the source of the strength. Existing supervised and semi-supervised methods are unable to achieve the same level of detection accuracy because the focus was on the second sub problem: classification, rather than the first sub problem: emerging new class detection.

The fact that the unsupervised learner consists of completely random trees facilitate the use of a common core which can be converted to an effective classifier with ease. The common core also makes model updates in data streams to be a simple model adjustment, rather than training a completely new model as in most existing methods. Like in previous work, we show that the completely random trees are a classifier competitive to state-of-the-art classifiers, especially in the data stream context which demands fast model update and classification time.

Our empirical evaluation shows that *SENCForest* outperforms eight existing methods, despite the fact that it was not given the true class labels in the entire data stream; and other methods were given the true class labels at each model update. In addition, it works effectively in long stream with emerging new classes under the limited memory environment. No existing methods have the capability to work under the same condition, as far as we know.

In the future, we plan to improve the proposed method to deal with concept drift and to differentiate two or more emerging new classes before model updates. From a broader perspective, the proposed method is the first implementation of the unsupervised-anomaly-detection-focused approach to the *SENC* problem. We intend to explore other implementations of the same approach.

7. appendices

7.1. Parameter settings

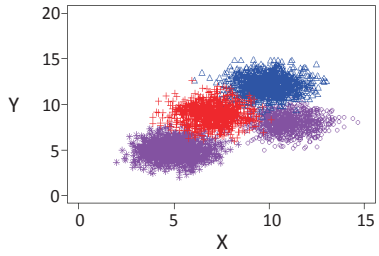
The parameter settings of all algorithms used in the experiments are provided in Table 5. A 10-fold cross-validation on the training set is used in the parameter search to determine the final settings for all SVM algorithms. The parameter search for LOF is as described in [DYZ14]. ECSMiner employs K-means and K is set to 5 in the experiment.

Table 5: The settings used in the experiments.

Method	Parameter setting & search range
LOF	$k = [3, 9]$
1SVM	$c = 0.1 \sim 100$, default settings in others
1R-SVM	$\gamma = 2^{\alpha}/num\_features$, $\alpha = [-5, 5]$ $c = 0.1 \sim 100$, and default in others
LACU-SVM	$ramp_s = -0.3, \eta = 1.3,$ $\lambda = 0.1, max\_iter = 10$
ECSMiner	$S = 250, M = 6, K = 5$ $q = 5, T_l = 200$
iForest	$\psi = 200, t = 100, MinSize = 10$
SENCForest	$\psi = 200, z = 100, \rho = 3, MinSize = 10$

7.2. Descriptions of data sets

Synthetic: We simulate a data stream using a two dimensional synthetic data set as shown below. It contains 20,000 instances and has four overlapping Gaussian distribution. The first two



initial known classes are marked with purple. In the first period, instances of class blue emerge as the first new class. In the second period, instances of class red emerge as the second new class.

MHAR: This data set [AGO+12] is collected from 30 volunteers wearing a smart phone on the waist and performing 6 activities (walking, upstairs, downstairs, standing, sitting, laying). The embedded 3D-accelerometer and 3D-gyroscope of a Samsung Galaxy S2 smart phone were used to collect data at a constant rate of 50 Hz. This data set has 6 classes, 10299 instances and 561 attributes.

References

[Agg13] Charu C. Aggarwal. A survey of stream clustering algorithms. In *Data Clustering: Algorithms and Applications*, pages 231–258. 2013.

[AGO+12] Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra, and Jorge Luis Reyes-Ortiz. Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine. In *Proceedings of the 4th International Workshop on Ambient Assisted Living and Home Care*, pages 216–223, Vitoria-Gasteiz, Spain, 2012.

[BD93] E Brodsky and Boris S Darkhovsky. *Nonparametric methods in change point problems*. Springer Netherlands, 1993.

[BHP+09] Albert Bifet, Geoffrey Holmes, Bernhard Pfahringer, Richard Kirkby, and Ricard Gavaldà. New ensemble methods for evolving data streams. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 139–148, Paris, France, 2009.

[BKNS00] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. LOF: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 93–104, Dallas, Texas, USA, 2000.

[BN96] Michèle Basseville and Igor V. Nikiforov. Detection of abrupt changes: Theory and application. *Automatica*, 32(8):1235–1236, 1996.

[CBK09] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Computing Surveys (CSUR)*, 41(3):15, 2009.

[CL11] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Trans. Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.

[CRT11] Feilong Chen, Supranamaya Ranjan, and Pang-Ning Tan. Detecting bots via incremental LS-SVM learning with dynamic feature adaptation. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 386–394, San Diego, CA, USA, 2011.

[DDD05] Frédéric Desobry, Manuel Davy, and Christian Doncarli. An online kernel change detection algorithm. *IEEE Trans. Signal Processing*, 53(8):2961–2974, 2005.

[DYZ14] Qing Da, Yang Yu, and Zhi-Hua Zhou. Learning with augmented class by exploiting unlabeled data. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence*, pages 1760–1766, Québec City, Québec, Canada, 2014.

[FWYM03] Wei Fan, Haixun Wang, Philip S. Yu, and Sheng Ma. Is random model better? on its accuracy and efficiency. In *Proceedings of the 3rd IEEE International Conference on Data Mining*, pages 51–58, Melbourne, Florida, USA, 2003.

[HAY+07] Chang Huang, Haizhou Ai, Takayoshi Yamashita, Shihong Lao, and Masato Kawade. Incremental learning of boosted face detector. In *Proceedings of the 11th IEEE Conference on Computer Vision*, pages 1–8, Rio de Janeiro, Brazil, 2007.

[JA03] Ruoming Jin and Gagan Agrawal. Efficient decision tree construction on stream-

ing data. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 571–576, New York, NY, USA, 2003.

[KM07] J Zico Kolter and Marcus A Maloof. Dynamic weighted majority: An ensemble method for drifting concepts. *Journal of Machine Learning Research*, 8:2755–2790, 2007.

[KOC13] Ilja Kuzborskij, Francesco Orabona, and Barbara Caputo. From n to n+ 1: Multiclass transfer incremental learning. In *Proceedings of the 26th IEEE Conference on Computer Vision and Pattern Recognition*, pages 3358–3365, Portland, OR, USA, 2013.

[LTF05] Fei Tony Liu, Kai Ming Ting, and Wei Fan. Maximizing tree diversity by building complete-random decision trees. In *Proceedings of the 9th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 605–610, Hanoi, Vietnam, 2005.

[LTYZ08] Fei Tony Liu, Kai Ming Ting, Yang Yu, and Zhi-Hua Zhou. Spectrum of variable-random trees. *Journal of Artificial Intelligence Research*, pages 355–384, 2008.

[LTZ08] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *Proceedings of the 8th IEEE International Conference on Data Mining*, pages 413–422, Pisa, Italy, 2008.

[MGK+11] M.M. Masud, Jing Gao, L. Khan, Jiawei Han, and Bhavani Thuraisingham. Classification and novel class detection in concept-drifting data streams under time constraints. *IEEE Trans. Knowledge and Data Engineering*, 23(6):859–874, 2011.

[MP03] J. Ma and S. Perkins. Time-series novelty detection using one-class support vector machines. In *Proceedings of the International Joint Conference on Neural Networks*, pages 1741–1745, Portland, OR, USA, 2003.

[RK04] Ryan M. Rifkin and Aldebaro Klautau. In defense of one-vs-all classification. *Journal of Machine Learning Research*, 5:101–141, 2004.

[SdC04] Eduardo J. Spinosa and André Carlos Ponce Leon Ferreira de Carvalho. Svms for novel class detection in bioinformatics. In *III Brazilian Workshop on Bioinformatics*, pages 81–88, Brasília, Distrito Federal, Brazil, 2004.

[SdRRSB13] Walter J Scheirer, Anderson de Rezende Rocha, Archana Sapkota, and Terrance E Boult. Toward open set recognition. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 35(7):1757–1772, 2013.

[SPST+01] Bernhard Schölkopf, John C. Platt, John C. Shawe-Taylor, Alex J. Smola, and Robert C. Williamson. Estimating the support of a high-dimensional distribution. *Neural Computation*, 13(7):1443–1471, 2001.

[YYH07] Jun Yang, Rong Yan, and Alexander G Hauptmann. Cross-domain video concept detection using adaptive svms. In *Proceedings of the 15th international conference on Multimedia*, pages 188–197, Augsburg, Germany, 2007.

[ZC02] Zhi-Hua Zhou and Zhao-Qian Chen. Hybrid decision tree. *Knowledge-based systems*, 15(8):515–528, 2002.

[Zho12] Zhi-Hua Zhou. *Ensemble methods: foundations and algorithms*. 2012.