Difficult Novel Class Detection in Semisupervised Streaming Data

Peng Zhou, Ni Wang, Shu Zhao, Yanping Zhang, and Xindong Wu, *Fellow, IEEE*

***Abstract*— Streaming data mining can be applied in many practical applications, such as social media, market analysis, and sensor networks. Most previous efforts assume that all training instances except for the novel class have been completely labeled for novel class detection in streaming data. However, a more realistic situation is that only a few instances in the data stream are labeled. In addition, most existing algorithms are potentially dependent on the strong cohesion between known classes or the greater separation between novel class and known classes in the feature space. Unfortunately, this potential dependence is usually not an inherent characteristic of streaming data. Therefore, to classify data streams and detect novel classes, the proposed algorithm should satisfy: 1) it can handle any degree of separation between novel class and known classes (both easy and difficult novel class detection) and 2) it can use limited labeled instances to build algorithm models. In this article, we tackle these issues by a new framework called semisupervised streaming learning for difficult novel class detection (SSLDN), which consists of three major components: an effective novel class detector based on random trees, a classifier by using the information of nearest neighbors, and an efficient updating process. Empirical studies on several datasets validate that SSLDN can accurately handle different degrees of separation between the novel and known classes in semisupervised streaming data.**

***Index Terms*— Data stream, novel class detection, semisupervised learning (SSL), streaming classification.**

I. Introduction

MANY advanced traditional machine learning methods are proposed based on the assumption that the learning environment is static. However, data are often dynamic in practical applications, such as traffic control, social media, weblogs, market analysis, and sensor networks [1]. Large amounts of instances are continuously generated and

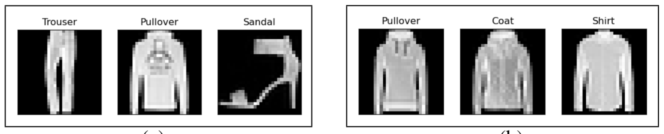
Manuscript received 12 October 2021; revised 13 April 2022, 22 June 2022, and 4 August 2022; accepted 4 October 2022. Date of publication 25 October 2022; date of current version 6 October 2023. This work was supported in part by the National Natural Science Foundation of China under Grant 61906056, Grant 61806002, and Grant 62120106008; and in part by the Program for Changjiang Scholas and Innovative Research Team in University (PCSIRT) of the Ministry of Education under Grant IRT17R32. *(Corresponding author: Shu Zhao.)*

Peng Zhou, Ni Wang, Shu Zhao, and Yanping Zhang are with the Key Laboratory of Intelligent Computing and Signal Processing, Ministry of Education, the School of Computer Science and Technology, and the Information Materials and Intelligent Sensing Laboratory of Anhui Province, Anhui University, Hefei 230601, China (e-mail: doodzhou@ahu.edu.cn; wangni613@hotmail.com; zhaoshuzs@ahu.edu.cn; zhangyp2@gmail.com).

Xindong Wu is with the Key Laboratory of Knowledge Engineering with Big Data, Ministry of Education of China, Hefei University of Technology, Hefei 230009, China (e-mail: xwu@hfut.edu.cn).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TNNLS.2022.3213682.

Digital Object Identifier 10.1109/TNNLS.2022.3213682



(a) (b)

Fig. 1. Illustrative examples of detecting new emerging classes in (a) data stream A and (b) data stream B using images in the Fashion-MNIST dataset. In stream A, detecting the emerging novel class of "Sandal" is easy because the feature characteristics of the novel class and known classes are significantly different. In stream B, detecting the novel class of "Shirt" is more difficult because "Shirt" may look very similar to those of "Pullover" or "Coat."

exist in the form of data streams. In the past ten years, to extract relevant information or patterns from these stream data, data stream mining has attracted extensive research attention [2], [3].

In many real-world data stream applications, the concept of underlying problems can evolve [4]. For example, in social media, such as Twitter, Facebook, and Weibo, new hot topics often appear [5], [6]. In other words, new patterns (classes) can emerge during the streaming data. Therefore, novel class detection in streaming data refers to learning algorithms that can detect and learn novel classes [3]. Most of these algorithms work in a supervised setting, assuming that all the training data are labeled except for the novel class [7], [8], [9]. However, a more realistic situation is that only a few instances in the data stream are labeled and the stream data are semisupervised [10]. Due to practical constraints, such as time and resource cost, it is impossible to require the actual labels of all instances in the data stream. Therefore, only a small amount of data are labeled in practice, and the rests are unlabeled. Semisupervised data stream learning was proposed to deal with this new but realistic issue and has gained growing attention in recent years [11], [12], [13].

Besides, most of the existing data stream novel class detection methods usually have an implicit assumption: the novel class is far away from the known class in the feature geometric space or the density distribution of the known classes is relatively concentrated [4], [5], [6], [11], [12], [13]. In other words, their novel class detection mechanism relies on strong cohesion (i.e., small intraclass distance) or significant data separation (i.e., large interclass distance) of the instances in the observation feature space. However, in a real-world streaming data environment, data distribution is often not as ideal as assumed. For example, Fig. 1 gives a cloth identification example of two streams. In stream A, the known classes

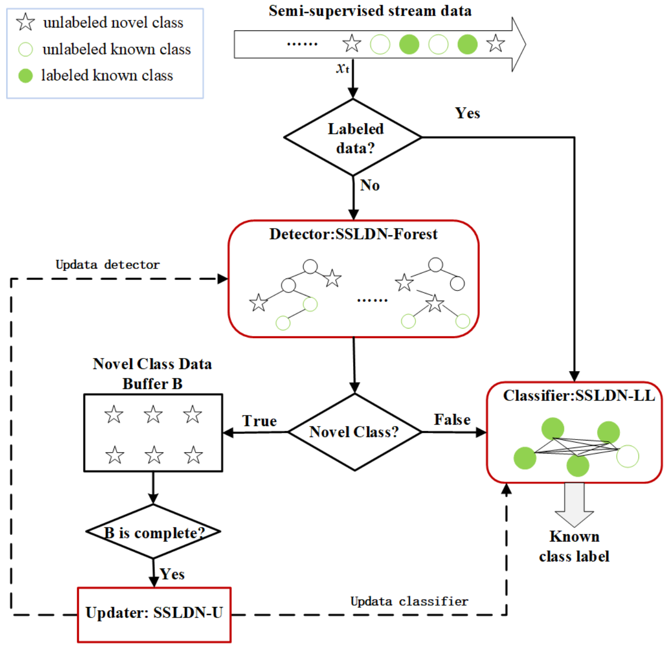


Fig. 2. Key steps of our proposed framework. First, we use the initial semisupervised training data to construct the detector SSLDN-Forest and the classifier SSLDN-LL. During the streaming data, for a new arrival instance x_t, if x_t is labeled, it will be used to update the classifier SSLDN-LL; otherwise, the detector SSLDN-Forest determines whether x_t is a novel class or not. If x_t is a novel class, we put it in the novel class buffer B. If x_t is a known class, we use SSLDN-LL to classify it. When the buffer B is complete, we update SSLDN-Forest and SSLDN-LL with the cached instances in B.

are “Trouser” and “Pullover,” detecting the emerging novel class of “Sandal” is easy because the feature characteristics of the novel class and known classes are significantly different. By contrast, if the known classes are “Pullover” and “Coat” in stream B, the novel class is “Shirt,” where “Shirt” may look very similar to those of “Pullover” or “Coat,” and this novel class is more challenging to detect. Thus, when the novel class and known classes are closer to each other in the feature geometric space, it is more difficult to detect the novel class in this case.

As far as we know, very few works focus on the issue of difficult novel class detection in semisupervised stream data. Specifically, Cai et al. [14] used the labeled training set to build a model based on the nearest neighbor ensembles method and then used the novel class score and the known class score to predict the data stream. However, the algorithm has the following limitations: 1) the initial training instances should be all labeled, but in actual application scenarios, labeling the whole initial training set is unrealistic, and the more common case is only a few labeled instances; 2) the judgment threshold is fixed, which limits the performance on different datasets; and 3) for each test stream instance, the algorithm must find the center of the smallest hypersphere that covers it, which is expensive to run. Therefore, this article aims to handle the issue of difficult novel class detection in semisupervised streaming data with more efficient and scalable methods.

Fig. 2 shows our proposed novel semisupervised streaming learning for difficult novel class detection, named SSLDN. The proposed framework mainly consists of three components.

1) *Novel Class Detector SSLDN-Forest:* It determines whether a new arriving stream instance is a novel class or not by the voting mechanism of SSLDN-Trees, where each SSLDN-Tree uses the path length to judge whether the new stream point is a novel class. item *Known Class Classifier SSLDN-LL:* It classifies new unlabeled known class instances according to the decision function obtained with a fixed number of labeled instances.
2) *Updater SSLDN-U:* The current models (SSLDN-Forest and SSLDN-LL) are updated using instances in the buffer B when B is full.

Our main contributions are concluded as follows.

1) We give a formal definition of semisupervised classification for the emerging novel class (SSENC) and analyze the difficulties and challenges of the difficult novel class detection problem in SSENC.
2) We propose the efficiency framework SSLDN to handle the difficult novel class problem in SSENC that consists of three main components: detector SSLDN-Forest, classifier SSLDN-LL, and updater SSLDN-U. SSLDN-Forest comprises several SSLDN-Trees that guarantee accurate judgment of the new stream instance and can automatically update the judgment threshold. In terms of the principle of nearest neighbors, SSLDN-LL can classify unlabeled known class instances by the decision function obtained from the similarity matrix and indicator vector matrix of a small number of labeled instances. SSLDN-U can update the detector and classifier effectively with the instances in the buffer. Based on these, SSLDN can effectively detect difficult novel classes and classify known class instances in semisupervised streaming data.
3) Comprehensive experiments on eight benchmark and real-world datasets, and comparisons with five state-of-the-art algorithms verify the effectiveness and efficiency of our proposed method.

Compared with the work proposed in [14], our new method has the following advantages: 1) SSLDN only needs a few labeled instances when constructing the initial training model; 2) SSLDN determines the threshold automatically; and 3) based on the saved information of center and radius formed by tree nodes, SSLDN is efficient by using very few labeled instances. Besides, in this article, we only focus on the issue of detecting the novel class in streaming data, also called concept evolution [15]. We did not consider the concept drift and drifted data in our framework. Specifically, concept drift occurs in the stream when the underlying concepts of the stream change over time, while concept evolution occurs as a result of novel classes evolving in the stream [16].

This remaining article is organized as follows. Section II provides a brief literature review of existing approaches. Section III introduces the problem formulation. The complete detail of the proposed framework is presented in Section IV. Section V describes experimental settings and results. Finally, Section VI concludes this article.

II. Related Work

With the latest technological advances in processing power and communication tools, many real-world applications and hardware devices (such as sensors) generate large amounts of data in data streams every day. The streaming classification for the emerging novel class (SENC) is a vital issue in a data stream, where the learning process strengthens the previously trained classifier to deal with emerging new classes, causing extensive attention from the machine learning and data mining communities [16], [17]. The handling of the SENC problem is of great significance and has achieved great success in many real-world applications, such as spam detection, user behavior analysis, Web applications, monitoring patient health, fraud detection, and intrusion detection [18], [19], [20].

A. SENC Problem

Learning from streaming data has become a hot topic, and the SENC problem has attracted the attention of many scholars. Din et al. [21] provided a comprehensive overview of the existing works and discussed and analyzed various aspects of the proposed algorithms for data stream classification with concept evolution detection and adaptation. Specifically, ECSMiner proposed by Masud et al. [16] indicated that by introducing a time limit for delayed classification, the method could automatically detect novel classes even if the classification model is not trained with novel class instances. ECSMiner assumed that true labels of new emerging classes could be obtained after some time delay. Wang et al. [5] proposed a convolutional neural network (CNN)-based prototype ensemble (CPE) framework with a smart incremental learning strategy during stream classification and novel class detection for high-dimensional data streams. However, CPE has a deep neural network and update mechanism that consumes tremendous online computation during calculation. To handle the two main challenges in class-incremental learning: how to conduct novelty detection and how to update the model with a few novel class instances, Zhou et al. [22] proposed a novel framework (LC-INC) to handle the incremental new class, which can dynamically combine the prediction information with structure information to detect novel class instances efficiently. To handle the three problems of existing SENC approaches: high false positive for the new class, long prediction time, and true labels for all instances, Zhang et al. [23] proposed the k-Nearest Neighbor ENSemble (KNNENS)-based method to detect the new class, maintained high classification performance for known classes, and did not require true labels of new class instances for the model update. Mu et al. [4] proposed the SENC-MaS that used two low-dimensional matrix sketches for detecting new classes and classifying known classes. These two sketches are continuously updated in the stream. The premise of defining matrix sketches is that the known class feature information can be well represented, while matrix sketches will not represent the feature structure of the new class. Thus, SENC-MaS is based on the assumption that the data of any new class are far away from the data of other known classes in the feature space. Mu et al. [17] introduced an alternative technique as a solution to the problem of classifying new classes of data streams. The completely random trees are used as a single common core to solve all three subproblems: model updates, supervised learning, and unsupervised learning. The limitation of this method is that it does not give true class labels. To solve concept drift and temporal dependency problems at the same time, Song et al. [24] proposed the local drift degree that was used as a drift adaptation technique in a novel drift adaptation regression framework to discard outdated instances in a timely way, thereby guaranteeing that the most relevant instances will be selected during the training process.

Besides, some works only focus on detecting and identifying data that have never been seen before through the training process, such as novel class detection [25], [26], [27] and outlier detection [28], [29], [30]. These works only study subproblems of our setting and ignore the problem of classification and model update. Therefore, the usage of these methods in streaming data is limited, and the way to solve the whole SENC problem should combine with some classification frameworks.

B. Semisupervised Learning

Semisupervised learning (SSL) aims to use unlabeled data for training, typically, a small set of labeled data together with an extensive collection of unlabeled data. In reality, semisupervised streaming data are a more common situation.

Specifically, Haque et al. [10] proposed a semisupervised framework (SAND) that used change detection on classifier confidence to detect concept drifts and to determine chunk boundaries dynamically. SAND intelligently selects a few instances using the estimated classifier confidence scores. However, this kind of active selection is not natural in general semisupervised settings [31]. Yang et al. [32] proposed a semisupervised class-incremental learning without forgetting (CILF) method, which aims to learn adaptive embedding for processing novel class detection and model update in a unified framework. Zheng et al. [33] proposed a semisupervised framework (ESCR) to detect recurring concept drift and concept evolution in data streams with partially labeled data. ESCR used the Jensen–Shannon divergence-based change detection technique on classifier confidence score instead of classification error rate to detect recurring concept drift. However, ESCR uses too many parameters that are difficult to tune. Since labeling all instances in a potentially lifelong data stream is frequently prohibitively expensive, Soares et al. [34] proposed a novel algorithm to exploit unlabeled instances, and the algorithm was an online semisupervised radial basis function neural network (OSNN) with manifold-based training to exploit unlabeled data while tackling concept drifts in classification problems. Zhang et al. [13] proposed an adaptive matrix sketching and clustering method, which cohesively and adaptively classifies known classes, identifies multiple novel classes, and updates the learning model. Any instances far away from these frequent directions are considered to be from the novel class. However, this approach relies on the strong intrinsic cohesion and separation assumption. Zhu et al. [12] presented a semisupervised learning framework (SEEN) that is capable of handling emerging new classes in a dynamic data stream where a few labeled instances are collected together with a large number of unlabeled instances. However, SEEN forest construction uses half of the original features, which may reduce the feature information expression of the data. Din et al. [11] proposed a new online SSL algorithm by modeling concept drifts with a set of microclusters. These microclusters are dynamically maintained to capture the evolving concepts with error-based representative learning. Nevertheless, the kept microclusters could not capture the hidden local cluster structure for incoming data with non-Gaussian distributions, and they tend to fail with high-dimensional data.

All these methods mentioned above have an implicit assumption that: the novel class and the known class are far apart in the feature geometric space or the density distribution of the known class is relatively concentrated. This assumption is helpful to detect instances of novel classes in the stream, but it may not be an intrinsic property of stream data in reality. In actual application scenarios, low separations between novel classes and known classes or high cohesion between known classes are very common, and it is usually challenging to detect a novel class in these cases. Unfortunately, there are very few works that focus on this issue. Cai et al. [14] provided a solution to this issue, which detects the novel class and classifies known classes according to the new-class score and known-class score. However, as mentioned above, this method has some limitations that cannot be ignored. Therefore, this article aims to propose a novel semisupervised streaming learning framework that can detect difficult novel classes accurately and classify known classes effectively.

III. Problem Formulation

In the issue of dynamic SSL, instances continue to emerge from the data stream. We use the semisupervised streaming classification for the emerging new class to represent the SENC problem in semisupervised stream data. This section first gives the formal definition and challengings of SSENC. Then, we present the separation indicator α, which measures the difficulty of detecting novel classes. Finally, we analyze the problem of difficult novel class detection in SSENC.

A. Semisupervised Streaming Classification for Emerging New Class

Definition 1 (SSENC): Suppose that $S = \{(x_t, y_t)\}_{t=0}^{t=T_0}$ represents the data streams from timestamp 0 to the initial timestamp T_0, where $x_t \in \mathbb{R}^d$ is the training instance at timestamp t. $Y = \{1, 2, \ldots, k\}$ is the label set of known classes, where k is the number of known classes so far. For semisupervised data, there are two states of y_t: $y_t \in \{0, \neg 0\}$. For the instance x_t in the data stream at timestamp t, if $y_t = 0$, then x_t is unlabeled data; if $y_t \in Y$, then x_t is labeled data. Until timestamp T_0, all arrived instances in S are used to initialize the detector and classifier. With the increase of time, data stream evolution arises. $S' = \{(x'_t, y'_t)\}_{t=T_0+1}^{\infty}$ and $Y' = \{Y, n_1, n_2, \ldots\}$, where n_j is the label given for an emerging novel class. If $y'_t = 0$, x'_t is unlabeled. SSENC aims to learn a model f (initialized with S) such that $f(x'_t) \to Y'$ is "as good as possible." Specifically, for each $(x'_t, y'_t = 0)$ in the semisupervised data stream, f will determine whether it is a novel class or belongs to an existing class.

Most existing methods for the SENC problem are based on an implicit premise that the initial model should be established on enough labeled instances. However, the actual situation is not the case, and we are more likely to retain a small number of labeled instances within a large number of unlabeled data. Therefore, solving the SENC problem in semisupervised streaming data seems to more align with actual needs. The challenges of the SSENC problem include: 1) how to use semisupervised data to build the initial model; 2) how to make better use of a large amount of unlabeled data compared with a small amount of labeled data; and 3) how to achieve an efficient update process, and the model should also meet the requirements of high-precision detection and classification.

B. Separation Indicator

Separation indicator α was first defined in [14] which reflects the separation between known classes and a novel class. For an instance x, η_x represents its nearest neighbor belongs to the same class, and let $\tau(x)$ denote their distance, expressed as: $\tau(x) = \|x - \eta_x\|$. The definition of α is described as follows:

$$\alpha(K_D, N_D) = \frac{M(K_D, N_D)}{C(K_D)} \tag{1}$$

where K_D and N_D represent the known class instances set and novel class instances set, respectively. $M(K_D, N_D) = \min_{x \in K_D, x' \in N_D} \|x - x'\|$ represents the smallest distance between the known classes and novel class. $C(K_D) = (1/|K_D|) \sum_{x \in K_D} \tau(x)$ represents the compactness of the known classes. When $M(K_D, N_D)$ is small, the separation between the new and the known classes is small. On the contrary, if the value of $C(K_D)$ is large, the data in the feature space of the known classes are weakly cohesive. Therefore, the value of α can reflect the difficulty of detecting novel class. Theoretically, the smaller the value of α, the more difficult for the SENC problem.

C. Difficult Novel Class Detection in SSENC

Under the premise of different values of α, the degree of difficulty in SSENC is different. In the semisupervised data stream, both the feature structure of the new data and the relationship with the feature space of the known data are unknown. In actual situations, we cannot make assumptions about the stream data in advance. Therefore, there are two extreme cases in the feature space between the novel class data and the known class data: high-α and low-α.

Specifically, when the distance between the novel class and the known class is large in the feature geometric space (e.g., $M(K_D, N_D)$ is large) or the distribution of known classes is denser (e.g., $C(K_D)$ is small), we call it the high-α situation. It is effortless to detect the novel class because the feature geometric distance differs between the novel class and the known class. Nevertheless, this assumption is not an immutable feature of the actual data stream. When the case is contrary to this assumption (e.g., $M(K_D, N_D)$ is small or $C(K_D)$ is large), we call this the low-α situation. In low-α, the novel class instance looks very similar to the known class instances, so this issue will be more challenging because it is easy to classify such novel class stream data into one of the known classes.

Note that α is an indicator to denote the level of difficulty of the SSENC problem only, and it is not used in the proposed algorithms.

IV. SSLDN Framework

In this section, we proposed a new SSLDN, to solve the SSENC problem with different levels of separation. SSLDN consists of three main components: 1) an effective novel class detector SSLDN-Forest; 2) an accurate classifier SSLDN-LL; and 3) an efficient model update algorithm SSLDN-U. A schematic of the overall procedure of our new framework is given in Fig. 2.

Specifically, for the semisupervised data streams, if the newly arrived instance x_t is labeled, it will be used to update the classifier SSLDN-LL. On the contrary, if the newly arrived instance x_t is unlabeled, it will be identified by the novel class detector SSLDN-Forest built by initial training instances. Then, there are two cases for x_t: 1) x_t is a novel class and will be added into the temporary data buffer B and 2) x_t is not a novel class and will be classified by SSLDN-LL. When the temporary data buffer B reaches the maximum buffer size, an efficient model update algorithm SSLDN-U will be applied on the classifier SSLDN-LL and the detector SSLDN-Forest. The concrete details of our new framework are provided in the following.

A. Novel Class Detection: SSLDN-Forest

SLDN-Forest aims to judge whether the new arriving stream instance is a novel class or not in terms of the difference between the new arriving instance and the characteristic of all arrived samples. We use the distribution of feature spaces and the data scale of different feature values to construct the SSLDN-Forest.

Specifically, inspired by iForest [30], SSLDN-Forest is composed of several SSLDN-Trees, and each SLDN-Tree is constructed by selecting φ samples from the training set T without replacements. iForest consists of itrees, where each itree randomly selects an attribute and uses the dividing point between the minimum and maximum values in its subsamples to generate partitions. However, iForest cannot be automatically updated for streaming data and is sensitive to the scale of different feature values. Compared to iForest, SSLDN-Forest uses the median as the segmentation that can reduce the impact of the data scales of the sample feature value on the tree segmentation. Meanwhile, because of the automatic update function, SSLDN-Forest is suitable for stream data.

1) SSLDN-Forest: SLDN-Forest is composed of multiple SSLDN-Tree, where each SSLDN-Tree is constructed by randomly selecting φ samples from T. The main steps of forming SSLDN-Forest are shown in Algorithm 1.

Algorithm 1 SSLDN-Forest

Input: T (training set), z (number of trees), φ (number of samples for each tree)
Output: SSLDN-Forest

```
1: Initialize: SSLDN-Forest ← {};
2: for $\upsilon = 1, \ldots, z$ do
3:     $X_\upsilon$ ← Randomly select $\varphi$ samples from $T$ without replacements;
4:     Tree($\upsilon$) ← SSLDN-Tree($X_\upsilon$)
5:     SSLDN-Forest ← SSLDN-Forest ∪ Tree($\upsilon$)
6: end for
7: return SSLDN-Forest
```

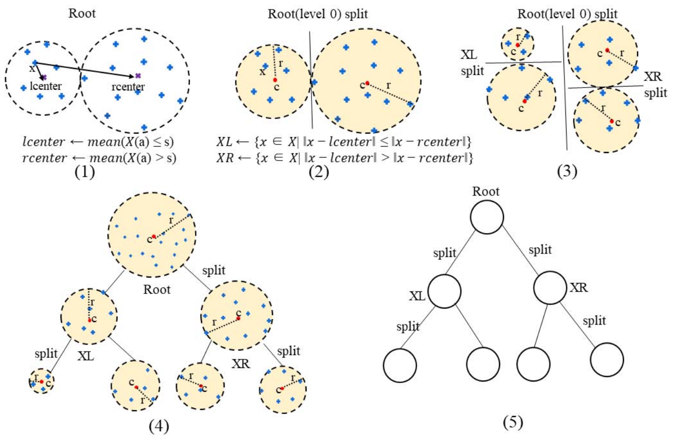


Fig. 3. Main steps of constructing an SSLDN-Tree. Suppose that there are 21 instances for an SSLDN-Tree, and we set the minimum number of instances for the leaf node as 4. From (1) to (4), all these instances are recursively divided into left subtree and right subtree. The final tree is shown as (5) with height 2 and 4 leaf nodes.

2) SSLDN-Tree: Each SSLDN-Tree is constructed by φ samples from the training set. We use Fig. 3 to describe the main steps of constructing an SSLDN-Tree. Specifically, we first randomly select an attribute a. Then, the attribute values of a for all the samples are sorted to get S, and the median s of S is obtained. Next, we divide the samples into two parts according to s and calculate the center points lcenter and rcenter for these two parts [Fig. 3(1)]. For each instance x, if $\|x - \text{lcenter}\| \leq \|x - \text{rcenter}\|$, then x is divided into the left subtree XL; otherwise, divide x into the right subtree XR. In this way, all the sample nodes are divided into two parts, XL and XR, and the center point c and radius r are obtained [Fig. 3(2)]. Then, repeating the process from Fig. 3(1) to Fig. 3(2), XL and XR continue to split [Fig. 3(3)]. When the number of instances in the leaf node is less than the minimum value, the current subtree stops dividing. The specific construction process is shown in Algorithm 2.

Let A be the set of attributes, where $X_{(a)}$ is the attribute value corresponding to attribute a in A. Step 2 calculates the center c and radius r of the current node based on all the attributes in A. Steps 3–13 are recursive processes that divide the instances in X. Specifically, steps 3 and 4 determine whether the number of instances in the current node is less

Algorithm 2 SSLDN-Tree

```
Input: $X$ (training intances), $MinSize$ (minium number instances for the leaf node).
Output: SSLDN-Tree
1: Let $A$ be the set of attributes;
2: Calculate the center $c = mean_{x \in X}(x)$ and the radius $r = max_{x \in X} \|x - c\|$;
3: if $|X| \leq MinSize$ then
4:     return LeafNode{Center←c, Radius← r}
5: else
6:     $a \leftarrow$ randomly select an attribute from $A$;
7:     $S \leftarrow sorted(X_{(a)})$;
8:     $s \leftarrow \frac{S[\frac{|X|}{2}]+S[1+\frac{|X|}{2}]}{2}$;
9:     $lcenter \leftarrow mean(X_{(a)} \leq s)$;
10:    $rcenter \leftarrow mean(X_{(a)} > s)$;
11:    $XL \leftarrow \{x \in X| \ \|x - lcenter\| \leq \|x - rcenter\|\}$;
12:    $XR \leftarrow \{x \in X| \ \|x - lcenter\| > \|x - rcenter\|\}$;
13:    return InNode{Center ← c, Radius ← r,
       SplitValue ← s, SplitCenters← $\{lcenter, rcenter\}$,
       Left ← SSLDN-Tree($XL$), Right ← SSLDN-Tree($XR$)}
14: end if
15: return SSLDN-Tree.
```

than MinSize. If true, the current node is a leaf node, and we terminate the division; otherwise, step 6 randomly selects an attribute from A. Steps 7–10 sort the instances in X, calculate the median, and get the centers of the left subtree and right subtree. Steps 11–13 construct the left subtree (XL) and the right subtree (XR). Let us use a simple example to illustrate the concrete construction process of SSLDN-Tree.

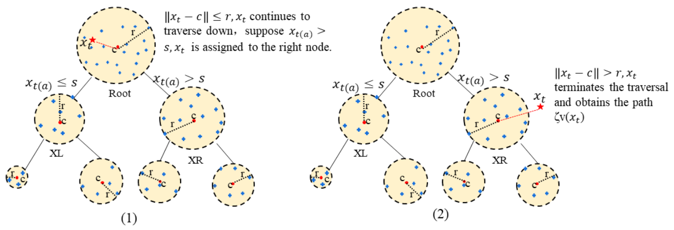
Suppose that $X = \begin{bmatrix} 9 & 2 & 6 & 8 \\ 3 & 4 & 4 & 7 \\ 4 & 8 & 7 & 6 \\ 2 & 5 & 3 & 1 \end{bmatrix} \in R^{4\times 4}$ has four instances $\{x_1, x_2, x_3, x_4\}$ and four attributes $\{a_1, a_2, a_3, a_4\}$. The details of splitting the instances into left and right subtrees of SSLDN-Tree are given as follows.

First, randomly select one attribute from these four attributes. Assume that the selected attribute is a_2. Then, we sort $a_2 = [2, 4, 8, 5]$ as $S = [2, 4, 5, 8]$, and the median $s = (S[2] + S[3])/2 = (4 + 5)/2 = 4.5$.

Second, calculate lcenter and rcenter. Specifically, for $x_1 = [9, 2, 6, 8]$, the value of a_2 is $2 < 4.5$. For x_2, x_3, and x_4, the values of a_2 are $4 < 4.5$, $8 > 4.5$, and $5 > 4.5$, respectively. Therefore, x_1 and x_2 are temporarily one group, and x_3 and x_4 are temporarily another group. Then, lcenter = mean$(X(a) \leq s)$ = mean(x_1, x_2) = $[6, 3, 5, 7.5]$ and rcenter = mean$(X(a) > s)$ = mean(x_3, x_4) = $[3, 6.5, 5, 3.5]$.

Third, divide the instances in X into XL or XR in terms of the distances to lcenter and rcenter. Because $\|x_1 - \text{lcenter}\| = \sqrt{11.25} < \|x_1 - \text{rcenter}\| = \sqrt{77.5}$, x_1 is divided into the left subtree XL. Similarly, we can get XL = $\{x_1, x_2\}$ and XR = $\{x_3, x_4\}$.

Fourth, calculate the center and radius of the nodes formed by XL and XR. The center point formed by XL is c_1 = lecenter = $[6, 3, 5, 7.5]$ and the radius is $r_1 = \|x_2 - \text{lcenter}\| = \sqrt{11.25} = 3.3541$. The center point formed by XR is c_2 = rcenter = $[3, 6.5, 5, 3.5]$ and the radius is $r_2 = \|x_4 - \text{rcenter}\| = \sqrt{13.5} = 3.6742$.



$\|x_t - c\| \leq r, x_t$ continues to traverse down, suppose $x_{t(a)} > s, x_t$ is assigned to the right node.
x_t
r
c
$x_{t(a)} \leq s$
$x_{t(a)} > s$
Root
XL
XR
(1)
$\|x_t - c\| > r, x_t$ terminates the traversal and obtains the path $\zeta v(x_t)$
(2)

Fig. 4. Detection process of new arriving instance x_t within the υth SSLDN-Tree. We first judge whether x_t is in the normal area of the root, shown in (1). If x_t is in the normal area, then we traverse x_t down. When x_t is in the abnormal area of some node [shown as (2)] or in the leaf node, we terminate the traversal and return the path length.

Recursively split the left and right subtrees until the number of instances in the leaf node is less than the minimum value.

SSLDN-Tree uses the median of the attribute value to divide subsamples into two parts, gets the center points of two parts, and then calculates the distance between each sample and the two center points to determine whether this sample is divided into left subtree or right subtree. SSLDN-Tree chooses the median as the split point, which avoids different eigenvalue scales on tree segmentation. We divide all the instances into the left subtree or the right subtree according to the distance between the instances and the two center points. This approach ensures that the samples of the same node have similar characteristic attributes, which is advantageous for judging whether a new streaming instance is a novel class or not.

3) Detecting Process Using SSLDN-Forest: iForest [30] believed that normal points need more divisions to be isolated, while abnormal points need fewer divisions to be isolated. An instance with a shorter path length is more anomalous than an instance with a longer path length. For SSLDN-Forest, we also use the path length to judge the new arriving streaming instances.

After constructing each SSLDN-Tree, we calculate the path length between each node to the root and then get the average path length of each tree. We use formula (2) to express the threshold of the υth SSLDN-Tree

$$\text{tree}^{\upsilon}_{\text{avgpath}} = \frac{1}{m}\sum_{g=1}^{m} l_g \tag{2}$$

where m is the number of nodes in the υ-th SSLDN-Tree and l_g is the length of the path from each node to the root.

Besides, during the building of SSLDN-Tree, we calculate the center and radius of the samples in each node to represent the regional range. The calculation of the center and radius is shown in the following equation:

$$c = \underset{x \in X}{\text{mean}}\ (x), \qquad r = \underset{x \in X}{\max}\ \|x - c\|. \tag{3}$$

We use Fig. 4 to illustrate the detection process of the new arriving instance (test point) x_t. In Fig. 4(1), we first judge whether x_t is in the normal area at the root node. Specifically, the distance between x_t (red star) and the center point c is

Algorithm 3 Detecting Process Using SSLDN-Forest

```
Input: SSLDN-Forest, $z$ (number of SSLDN-Tree), $x_t$ (new arriving instance)
Output: True or False.
1: $num \leftarrow 0$;
2: for SLDN-Tree($\upsilon = 1, \ldots, z$) do
3:   $c, r$: the root node center and radius of SLDN-Tree(v)
4:   if $\|x_t - c\| \le r$ then
5:     $x_t$ continues to traverse down;
6:   else
7:     calculate the path length $\zeta_\upsilon(x_t)$;
8:   end if
9:   if $\zeta_\upsilon(x_t) < tree^{\upsilon}_{avgpath}$ then
10:    SSLDN-Tree($\upsilon$) outputs True, $num \leftarrow num + 1$ ;
11:  else
12:    SSLDN-Tree($\upsilon$) outputs False;
13:  end if
14: end for
15: if $num > \lfloor \frac{z}{2} \rfloor$ then
16:   return True
17: else
18:   return False
19: end if
```

compared with r to determine whether it is in the normal area ($\|x - c\| \le r$) or outside the normal area ($\|x - c\| > r$). If x_t is in the normal area, we traverse x_t down the tree. Suppose that $x_t(a) > s$, and then, x_t traverses to the XR node. In XR, if $\|x-c\| > r$, x_t falls outside the XR node [Fig. 4(2)], and the path length $\zeta_\upsilon(x_t)$ of XR is obtained. After that, if $\zeta_\upsilon(x_t) < \text{tree}^{\upsilon}_{\text{avgpath}}$, x_t is regarded as an instance of the novel class. Then, the υth SSLDN-Tree outputs "True" and count num plus one, where num is the number of SSLDN-Tree that outputs "True." Otherwise, x_t belongs to the known class and the υth SSLDN-Tree outputs "False." The final result adopts the majority voting mechanism, and SSLDN-Forest(x_t) is determined by z SSLDN-Tree. If num $> \lfloor z/2 \rfloor$, SSLDN-Forest outputs "True," and x_t is assigned as a novel class instance; otherwise, SSLDN-Forest outputs "False," and x_t is a known class instance. Algorithm 3 describes the details of the testing process by using SSLDN-Forest in the data stream.

If x_t belongs to one of the known classes, we determine the label of x_t by the know class classifier SSLDN-LL in Section IV-B; otherwise, if x_t is a novel class instance, we add x_t into the buffer B, which stores the previously unseen class instances. Once the buffer B is complete, we execute the model updater SSLDN-U in Section IV-C, and the new model can be ready for the subsequent test instances in the data stream.

B. Known Classes Classification: SSLDN-LL

If the new arriving instance is not a novel class, we use the known classes classifier to classify it into one of the known classes. To make accurate predictions on a large amount of newly arrived unlabeled instances and a few labeled instances, we propose the classifier SSLDN-LL that limits the number of labeled instances.

Specifically, recent advances in multiview [35] and adaptive clustering [36] believe that the smaller the distance between two data points should have a larger probability to be neighbors, which can be used to learn the similarity matrix and indicator matrix. SSLDN-LL establishes a similarity matrix and an indicator matrix between labeled instances to classify unlabeled data based on this assumption. For a newly arrived instance x_t, if x_t is labeled, we use x_t to update the SSLDN-LL classifier. If x_t is unlabeled, SSLDN-LL determines the label of x_t by the decision function composed of a similarity matrix and an indicator matrix.

1) Classification Principle: Given a set of data points $\{x_1, x_2, \ldots\ldots, x_n\}$, $x \in \mathbb{R}^d$, where l and u represent the number of labeled points and unlabeled points, $Y_l = [y_1, \ldots, y_l]^T$, where $y_i \in \mathbb{R}^{c\times 1}$ represents the known class indicator vector of the ith sample and c is the number of classes, and $y_{ij} = 1$ means that the ith sample belongs to the jth class. We adopt the data preprocessing proposed in [36] as

$$\min_{s_i \in n\times 1} \sum_{i,j}^{n} \|x_i - x_j\|_2^2 s_{ij} + \alpha \|S\|_F^2$$
$$\text{s.t. } \forall_i, s_i^T \mathbf{1} = 1, \quad 0 \le s_{ij} \le 1, \ \text{rank}(L_s) = n - c \tag{4}$$

where S represents the similarity matrix formed between data points and α is the regularization parameter. In [36], it assigns adaptive neighbors to each sample, which means that the similarity between data points will change. Thus, the similarity matrix S will be modified until it contains exact c connected components.

In the spectral analysis, $L_S = D_S - ((S^T + S)/2)$ is called the Laplacian matrix, where the degree matrix D_S in $S \in \mathbb{R}^{n\times n}$ is the diagonal matrix whose ith diagonal element is $\sum_i ((s_{ij} + s_{ji})/2)$. According to the content of semisupervised classification mentioned in [35], we use $F = [f_1, f_2, \ldots, f_n]$ to represent the class indicator matrix, where f_i is the class indicator vector with labeled data, and then rearrange all the points and make the first l points marked. We redivide L_s and F into blocks, so they could be expressed as $L_s = \begin{pmatrix} L_{ll} & L_{lu} \\ L_{ul} & L_{uu} \end{pmatrix}$ and $F = [F_l; F_u]$, where $F_l = Y_l$.

Unlike the method [35] that works on multiview, our calculation of the following two distances is only from the perspective of a single view. Specifically, $d_{ij}^x = \|x_i - x_j\|_2^2$ means the distance between x_i and x_j, and $d_{ij}^f = \|f_i - f_j\|_2^2$ represents the distance between f_i and f_j. We denote $d_i \in \mathbb{R}^{n\times 1}$ a vector with the jth element as $d_{ij} = d_{ij}^x + \lambda d_{ij}^f$. Thus, the regularization parameter α is

$$\alpha = \frac{1}{n} \sum_{i=1}^{n} \left(\frac{k}{2} d_{i,k+1} - \frac{1}{2} \sum_{j=1}^{k} d_{ij} \right) \tag{5}$$

where k refers to the number of nearest neighbors.

When solving S and F with labeled data, problem (4) can be transformed into the following formula:

$$\min_{S,F} \sum_{i,j}^{n} \|x_i - x_j\|_2^2 s_{ij} + \alpha \|S\|_F^2 + 2\lambda T_r(F^T L_s F)$$
$$\text{s.t. } s_i^T \mathbf{1} = 1, \quad 0 \le s_{ij} \le 1, \ F_l = Y_l \tag{6}$$

Algorithm 4 SSLDN-LL

```
Initialize: With training set $T$, $F_l$: class indicator matrix, $D_F$: distance matrix of $F_l$, $Y_x$: $l$ labeled instances, $D_X$: distance matrix of $Y_x$.
Input: $x_t$ (new stream instance), $p$ (the number of labeled instance per class).
Output: $None$ OR $y$.
1:  if $x_t$ is labeled then
2:      $y_t \in \mathbb{R}^{c\times 1}$: the known class indicator vector;
3:      $D_F \leftarrow D_F + dist(y_t, F_l)$;    //add new distance
4:      $D_X \leftarrow D_X + dist(x_t, Y_x)$;
5:      $F_l \leftarrow F_l \cup [y_t]^T$, $Y_x \leftarrow Y_x \cup [x_t]^T$;
6:      if $\exists$ class $j$, $\sum_i y_{ij} > p$ then
7:          $F_l \leftarrow$ delete the oldest $y_i$ in $F_l$;
8:          $Y_x \leftarrow$ delete the oldest $x_i$ in $Y_x$;
9:      end if
10: else
11:     $f_{x_t}$ = zero matrix $\in \mathbb{R}^{c\times 1}$;
12:     $D_F \leftarrow [D_F + dist(f_{x_t}, F_l)] \in \mathbb{R}^{(l+1)\times(l+1)}$;
13:     $D_X \leftarrow [D_X + dist(x_t, Y_x)] \in \mathbb{R}^{(l+1)\times(l+1)}$;
14:     initial $\alpha$ and $S$, $L_S = D_S - \frac{S^T+S}{2}$, $F_u = -L_{uu}^{-1}L_{ul}Y_l$;
15:     $y = \underset{j}{argmax} F_{ij}$,    $i = l+1, \forall j = 1, 2, \ldots, k$;
16:     $D_F \in \mathbb{R}^{l\times l} \leftarrow$ delete $dist(f_{x_t}, F_l)$ from $D_F$;
17:     $D_X \in \mathbb{R}^{l\times l} \leftarrow$ delete $dist(x_t, Y_x)$ from $D_X$;
18:     return $y$.
19: end if
```

that could be written as

$$\min_{F\in\mathbb{R}^{(l+1)\times c}, F_l=Y_l} T_r(F^T L_s F). \tag{7}$$

According to the work [37], the final part of F_u can be optimized and expressed as: $F_u = -L_{uu}^{-1}L_{ul}Y_l$. Then, the final prediction result of unlabeled data is passed through the decision function

$$y = \underset{j}{\operatorname{argmax}}\ F_{ij}, \quad i = l+1 \quad \forall j = 1, 2, \ldots, k. \tag{8}$$

2) Classification Process Using SSLDN-LL: SSLDN-LL always keeps a fixed number (p) of labeled instances for each known class. Therefore, the class indicator matrix (F_l) and similarity matrix (D_F, D_X) built with labeled data are constantly updated. Once the unlabeled instance x_t is predicted, SSLDN-LL will delete the class indicator vector (y_u) and similarity matrix vector [dist(y_u, F_l) and dist(x_t, Y_x)]. In other words, SSLDN-LL always uses a fixed number ($l = p * k$, where k is the number of known classes) of labeled instances to predict the target one unlabeled new arriving instance. The details of SSLDN-LL are shown in Algorithm 4.

Specifically, with the training set T, SSLDN-LL initializes and maintains the following matrix: $F_l = Y_l$ represents the class indicator matrix, Y_x represents the matrix formed by labeled instances, and D_F and D_X represent the distance matrix of F_l and Y_x. During data stream, when a new instance x_t arrives, if x_t is labeled, SSLDN-LL uses x_t to updates D_F and D_X in steps 2–5. Once the number of labeled instances for class j exceeds p, steps 6–9 delete the information of the oldest labeled instance that belongs to class j. If the sample x_t is unlabeled, steps 11–18 determine the class label (y) of x_t. Step 11 initializes the indicator vector of x_t as $f_{x_t} = [0, 0, \ldots, 0]^T \in \mathbb{R}^{c\times 1}$, $F = [F_l; f_{x_t}]$. Then, we update D_F and D_X with the adding new similarity matrix vector (dist(f_{x_t}, F_l)), dist(x_t, Y_x)). Next, step 15 determines the class label of x_t by the decision function [see (8)] mentioned above. After predicting x_t, we delete the distance information of x_t in both D_F and D_x. There are two main reasons for this: 1) avoid the impact of misclassification of x_t on the subsequent stream instances and 2) ensure the fixed number of labeled instances for the subsequent classification.

Algorithm 5 SSLDN-U

```
Input: $B$ (instances buffer), $s$ (maximum size of $B$), $p$ (the number of labeled instance per class).
Output: $None$.
1:  if $|B| \geq s$ then
2:      $T \leftarrow T \cup B$;
3:      rebuild SSLDN-Forest($T, z, \varphi$);
4:      $B_{center} \leftarrow mean_{b\in B}(b)$;
5:      $X_p \in R^{p\times d}$, $y_p \in R^{c\times p} \leftarrow$ select $p$ instances from $B$ that are close to $B_{center}$
6:      For current SSLDN-LL, $F_l$: original class indicator matrix, $D_F$: distance matrix of $F_l$, $Y_x$: original $l$ labeled data, $D_X$: distance matrix of $Y_x$.
7:      $D_F \leftarrow D_F + dist(y_p, F_l)$;
8:      $D_X \leftarrow D_X + dist(X_p, Y_x)$;
9:      $F_l \leftarrow F_l \cup [y_p]^T$, $Y_x \leftarrow Y_x \cup [X_p]^T$;
10:     retrain SSLDN-LL($F_l, D_F, Y_x, D_X, p$);
11: end if
```

C. Update Model: SSLDN-U

The model update procedure starts when the number of instances in novel class data buffer B reaches the maximum buffer size s. When the buffer is full, the model needs to be updated. All the instances in the buffer are regarded as a novel class, and we specify a new label for them. The instances in B are marked as the same new label to update the detector SSLDN-Forest and the classifier SSLDN-LL. The detail of the update algorithm SSLDN-U is shown in Algorithm 5.

Specifically, in steps 2 and 3, SSLDN-U combines the original data collector T with B and rebuilds the SSLDN-Forest. SSLDN-U treats all instances in B as the same new known class that facilitates the detection of potential novel class. Steps 4–10 are the update of the classifier SSLDN-LL. In step 4, SSLDN-U calculates the center point of all the instances in B. Step 5 selects the p nearest instances close to the center of B. This selection can make the classifier to be more robust in the classification for the following data stream. In steps 6–9, SSLDN-U updates D_F, D_X, F_l, and Y_x. Finally, step 10 rebuilds the classifier SSLDN-LL.

D. Complexity Analysis

In this section, we analyze the time and space complexity of our new framework according to the three main components: SSLDN-Forest, SSLDN-LL, and SSLDN-U.

For SSLDN-Forest, in the training phase, the time complexity is to construct z SSLDN-Trees by randomly selecting

φ samples from T for each tree. For each SSLDN-Tree, the time complexity of allocating φ data points would be $O(\varphi \log_2(N)d)$, where d is the feature dimension of target dataset and N is the number of nodes. In the detection phase, the most time-consuming is to compare the area range of the test point with the tree nodes. The time complexity of the detection phase is $O(z \log_2(N)d)$. Thus, the total time complexity of SSLDN-Forest is $O(z\varphi \log_2(N)d)$. Meanwhile, the space required for SSLDN-Forest includes the buffer with size s and the centers for all tree nodes. Thus, the space complexity is $O(s + zN)$.

The time complexity of SSLDN-LL is related to the number of labeled instances. The maximum time consumption is to add new distance into D_F and D_X. Therefore, the time complexity of SSLDN-LL is $O(ld)$, where l is the number of label instances in the classifier. The space consumption for SSLDN-LL is to store the l labeled instances and one unlabeled new stream instance, where each instance has d-dimensional features. Thus, the space complexity is $O(ld + d)$.

For SSLDN-U, the main time consumption is to calculate the center point of the data in the buffer and the time complexity is $O(sd)$. In the update phase, data are mainly operated from the buffer, so the maximum space complexity at this stage is the size of the buffer $O(s)$.

V. Experiments

A. Experimental Setup

This section describes experimental settings, including the details of datasets, data stream simulation, performance evaluation, competing algorithms, and parameters' setting. All algorithms are executed in the PYTHON environment.

1) Datasets: To evaluate the performance of the proposed approach, we adopt two kinds of datasets, i.e., benchmark and real-world textual data stream. For benchmark datasets, we use seven multiclass datasets, including (satimage, pendigits, HAR),[1] (USPS, MNIST-10K, MNIST),[2] and Fashion-MNIST.[3] In addition, we use NYTimes [14] as a real-world data stream, which is crawled news from the website between 2014 and 2017 using the New York Times API.[4] Specific descriptions of these eight datasets are shown in Table I. We use only two known classes and one novel class to obtain the greatest difficulty for these datasets during the novel class detection task. We tested different combinations of known and novel classes and chose the combinations of high-α and low-α in purpose. Besides, the datasets need to be normalized to handle features with different scales.

TABLE I

Summary of Datasets Used in the Experiments. The Left Part Is the Basic Information of Each Dataset. The Right Part Is the Value of α Varying With Different Classes. For Example, "3,1,7(8.9975)" Means That the Known Class Labels Are "3" and "1," the Novel Class Label Is "7," and the Separation Factor α Between the Novel Class and the Known Class Is 8.9975, Which Represents the Difficulty of Detecting the Novel Class

Dataset	# classes	# attributes	# instances	class(high-α)	class(low-α)
satimage	6	32	4,435	3,1,7(8.9975)	1,4,3(1.9228)
USPS	10	256	9,298	8,2,1(11.1887)	3,4,5(2.2781)
HAR	6	561	10,299	1,3,6(4.1886)	1,2,3(1.8814)
MNIST-10K	10	784	10,000	1,6,7(2.2629)	9,4,7(1.2078)
pendigits	10	16	10,992	3,5,9(13.7423)	0,1,2(3.4485)
MNIST	10	784	60,000	1,8,9(2.5718)	0,5,6(1.1626)
Fashion-MNIST	10	784	60,000	1,2,5(2.8114)	2,4,6(1.7589)
NYTimes	6	100	156,683	5,4,6(5.0139)	1,2,4(1.9592)

A more detailed description of these datasets is given as follows.

1) *Satimage:* The satimage database consists of the multispectral values of pixels in 3 $\times$ 3 neighborhoods in a satellite image and the classification associated with the central pixel in each neighborhood. The attributes are numerical, in the range of 0–255 (8 bits).
2) *USPS:* In this article, the USPS handwritten digit database contains 9298 16 $\times$ 16 handwritten digit images, which are then split into 7291 training images and 2007 test images, including all digits "0–9."
3) *HAR:* The HAR dataset is the Human Activity Recognition database. It is built from the recordings of 30 subjects performing activities of daily living (ADLs) while carrying a waist-mounted smartphone with embedded inertial sensors. Associated tasks of HAR can be used for classification and clustering.
4) *MNIST-10K:* MNIST-10K contains 10 000 samples, a trimmed version of the larger MNIST handwritten digits. The MNIST-10K dataset is a ten-class classification dataset consisting of 10 000 grayscale images of 28 $\times$ 28 pixels representing handwritten digits and their associated label (a number between 0 and 9).
5) *Pendigits:* The pendigits dataset is used for evaluation. This dataset is on pen-based digit recognition of handwritten digits. The pendigits dataset has a multiclass classification that uses ten classes and four types of representations. The number of attributes is 16 inputs and one class attribute. All the attributes in the dataset are numeric, and there are no missing attribute values.
6) *MNIST:* The MNIST dataset is from the National Institute of Standards and Technology (NIST). The training set consists of handwritten numbers from 250 different people, of which 50% are high school students and 50% are from the Census Bureau. The MNIST dataset contains 60 000 images in the training set, each size 28 $\times$ 28 pixels representing handwritten digits and their associated label (a number between 0 and 9).
7) *Fashion-MNIST:* Fashion-MNIST is a new dataset comprising 28 $\times$ 28 grayscale images of 70 000 fashion products from ten categories. The training set has 60 000 images, and the test set has 10 000 images. Fashion-MNIST is intended to serve as a direct drop-in replacement for the original MNIST dataset for benchmarking machine learning algorithms, as it shares the same image size, data format, and the structure of training and testing splits.

[1]https://archive.ics.uci.edu/ml/index.php

[2]http://www.cad.zju.edu.cn/home/dengcai/Data/MLData.html

[3]https://www.kaggle.com/zalando-research/fashionmnist

[4]http://developer.nytimes.com/

8) *NYTimes:* It is crawled over a period of time by using the New York Times API. There are 10k news items categorized into eight classes. In this article, NYTimes is crawled news from the website between 2014 and 2017 and contains 35 000 latest news items. Each news item is classified into six categories, "Arts," "Business Day," "Sports," "U.S.," "Technology," and "World." Each news story is converted into a 100-D vector using the word2vec technique [14].

2) Data Streams Simulation: We apply two types of data stream simulation for each dataset: fixed α and random α. In fixed α, there are two periods in the whole data stream: the training and streaming simulation. We first calculate the values of α for all combinations and then select two values of α (the highest one and the lowest one). For the fixed α problem, we only assume that one novel class appears in the second streaming simulation period. For the random α scenario, as the known class and the novel class are randomly selected from the whole class set, the data stream will have a mixture value of α.

For example, on the MNIST dataset, there are a total of ten classes. When simulating fixed α, we need to calculate different values of α first. The high α category is (1, 8, 9), where the known class labels are 1 and 8, and the new stream class is 9. Therefore, we use classes 1 and 8 as the initial training set, and the size is 2000 per class. Then, we select the instances of class labels 1, 8, and 9 as the stream data simulation, and the size is 1000 per class. The low α simulation stream is similar to the high α. When simulating random α, we randomly select two class labels from ten classes as known classes and randomly select one of the remaining class labels as the novel class.

3) Performance Metrics: Two measurements are used in this article. One is accuracy $= ((N_{\text{new}} + N_{\text{known}})/N)$, where N_{new} is the number of emerging class instances identified correctly, N_{know} is number of known class instances classified correctly, and N is the total number of instances. Another measure is macro-averaged F1 $= ((2 \times P \times R)/(P + R))$, which produces a combined effect of precision (P) and recall (R) of the detection performance in each class, where $P = (\text{TP}/(\text{TP} + \text{FP}))$, $R = (\text{TP}/(\text{TP} + \text{FN}))$, where TP is true positive, FP is false positive, and FN is false negative.

4) Competing Algorithms and Parameters Setting: We compare our new framework with the following.

1) *SEEN [12]:* An SSL framework that is capable of handling emerging new class in a dynamic data stream. In experiments, we set $h_m = 7$, $s = 50$, $\phi = 128$, $\tau = 50$, and $k =$ half of the number of features.
2) *ORSSL [11]:* A semisupervised algorithm exploits the online microclusters to summarize the streaming data in a compact form and is further used to classify the incoming data stream instances. ORSSL employs K-means, where K (number of clusters per class) $= 50$, maxMC $= 1000$, and $\theta = 4$.
3) *SENNE [14]:* It utilizes the nearest neighbor ensemble to deal with problems of different geometric distances. In SENNE, we set $t = 0.88$, $\psi = 20$, $p = 100$, and $s = 300$.
4) *iForest [30]:* It is an unsupervised anomaly detector. In iForest, according to the settings in [30], the number of trees in iForest is set to 100 and $\psi = 256$.
5) *LOF [28]:* A approach that loosely related to density-based clustering. For LOF, we set MinPts $= 50$.

In the following experiments, we use the parameter settings mentioned above for these competing algorithms. Besides, since iForest and LOF do not make classifications, we combine them with SVM as a classifier.

5) Number of Labels: Throughout the experiment, the number of labeled instances is set as follows. SSLDN and SEEN use 1% labeled data in both training and testing. ORSSL applies 100% labeled data in the initial model construction and uses 1% labeled data in the detection part. SENNE uses 100% labeled data. iForest + SVM and LOF + SLDN are unsupervised anomaly detection algorithms and use 1% labeled data in the detection. Besides, we analyze the impact of the number of labeled instances in Section V-E.

B. Parametric Analysis of SSLDN

We analyze the parameters used in SSLDN according to the components as follows.

In SSLDN-Forest, there are four parameters.

1) *φ:* It represents the number of subsamples randomly selected from the training set to construct SSLDN-Tree. The size of φ affects the detection accuracy and the height of the tree. Fig. 5(a) and (b) shows that a fixed value of φ cannot fit all these datasets, which fluctuates greatly on some datasets. Specifically, $\varphi = 20$ is the best for the pendigits, satimage, and USPS datasets. On the MNIST-10K, MNIST, and NYTimes datasets, $\varphi = 9$ is better. On the HAR and Fasion-MNIST datasets, $\varphi = 30$ is a good choice.
2) *z:* It is the number of SSLDN-Trees, and the value of z affects the accuracy of detecting novel class. From Fig. 5(c) and (d), we can see that the influence of different values of z is small on most of these datasets. Therefore, we select $z = 15$ for all these datasets in the experiments.
3) *MinSize:* It is the minimum number of instances of the leaf node for SSLDN-Tree. The value of MinSize determines the height of the constituted SSLDN-Tree. Because we set the values of φ 9, 20, or 30 for these datasets, MinSize must be smaller than φ. Thus, we set MinSize $= 4$ in the experiments.
4) *s:* It is the size of the buffer and is used to store novel class instances. In general, the value of s has little effect on the performance of the entire algorithm. Therefore, we set $s = 200$ as an experience value.

In the classifier SSLDN-LL, there are two required parameters.

1) *p:* It is used to control the number of labeled instances for each known class, and p affects the running time and classification accuracy. The accuracy and F1 varying with different values of p can be seen in Fig. 5(e) and (f), respectively. In general, with the increase of the value of p, the performance decreases. Thus, we select $p = 4$ as the best choice.

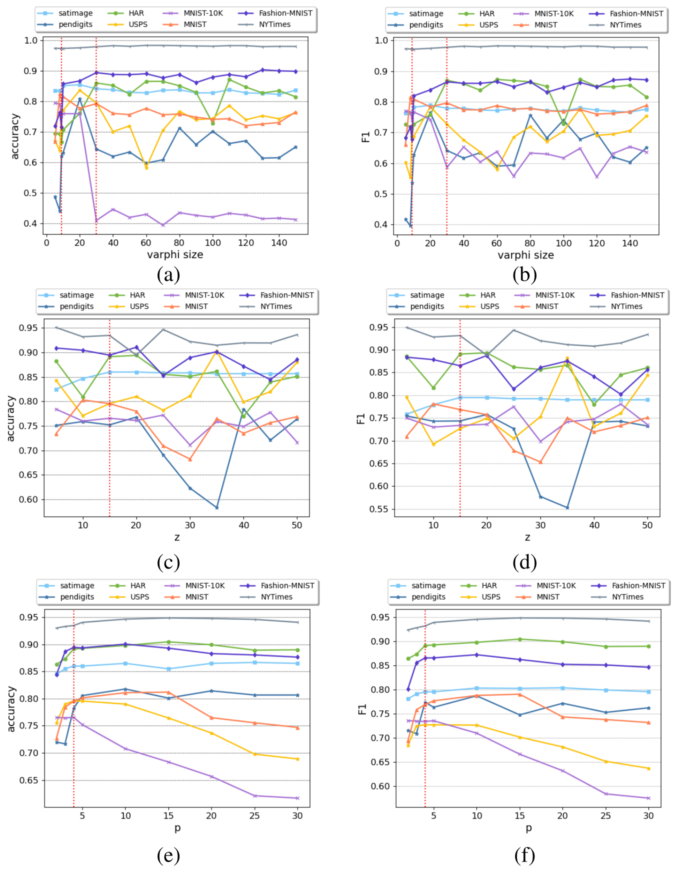


Fig. 5. Accuracy and F1 varying with different values of φ, p, and z. (a) Accuracy varying with φ. (b) F1 varying with φ. (c) Accuracy varying with z. (d) F1 varying with z. (e) Accuracy varying with p. (f) F1 varying with p.

2) *k:* It is the number of nearest neighbors in SSLDN-LL. Since we set $p = 4$ for these datasets, the minimum number of labels for a known class is 2. Therefore, we set $k = 1$.

C. SSLDN Versus Competing Algorithms

To verify the effectiveness of SSLDN, we conduct experiments on two types of stream data simulations: 1) fixed α: low α and high α and 2) random α.

1) Results on Fixed α: To examine the effect of high α and low α on the performances of all competing approaches, we select specific three classes in each dataset to simulate the SSENC scenarios in the cases of both high α and low α. The selected classes and the corresponding α values are shown in Table I.

Tables II and III present the accuracy and F1 of these competing algorithms on these datasets in the cases of high and low α respectively. In summary, the low α performance is worse than that on high α, mainly because it is more difficult to predict the data stream on low α than on high α. In the high α case, SSLDN achieves the best performance on all these eight datasets on accuracy. On $F1$, SSLDN performs best on six of eight datasets. Meanwhile, the F1 values of our algorithm on HAR and MNIST-10K are competing for the best results with only 0.0036 and 0.001 lower, respectively. In the low α case, SSLDN outperforms other algorithms on six of eight datasets, with an average improvement of 0.133 and 0.0487 on accuracy and F1, respectively. Therefore, SSLDN performs better than other competing algorithms in both fixed high α and low α.

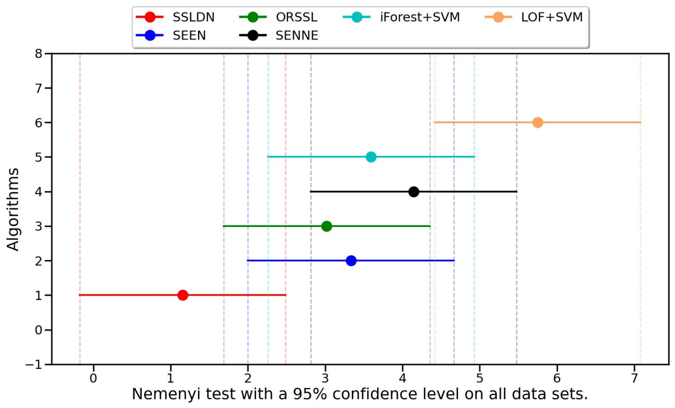


Fig. 6. Statistical test of these competing algorithms.

2) Results on Random α: Table IV shows the results of the random α scenario, where known classes and new stream classes are randomly selected from the whole class set. SSLDN performs better than all other competing methods on seven of eight datasets at least, in both cases of accuracy and F1. Besides, compared with the best performance of other methods, SSLDN improves accuracy and F1 by an average of 0.1111 and 0.0361, respectively. Thus, SSLDN achieves the best performance in the case of a random value of α.

3) Statistical Test: To compare the performances of the above algorithms from a statistical viewpoint, we use the Friedman test with a 95% confidence level. Null hypothesis H0 is proposed: there is no difference in the performance among these algorithms. If the null hypothesis is rejected, we further use the Nemenyi post hoc test to find these differences. Table V shows the average ranks of these competing algorithms in the cases of accuracy and F1 varying with different values of α. The p-value of Friedman test is $4.3382e^{-07}$. Thus, H0 is rejected, and these algorithms have a significant difference in performance. According to the Nemenyi test, the value of critical difference (CD) is 2.6657. We build CD diagrams, as shown in Fig. 6.

4) Result Analysis: From the results of Tables II–V, we can conclude that SSLDN is superior to all the other competing algorithms on these datasets. Two main factors guarantee the superiority of our algorithm: 1) For the novel class detection SSLDN-Forest, each SSLDN-Tree split instance uses the median of random feature values that ensure that the tree structure will not be affected even if there are different scales of feature values in the stream data and 2) for the classifier SSLDN-LL, we always maintain only one unlabeled known class instance and l labeled instances in the whole process, which ensures the accuracy of the result.

1) *SSLDN Versus SEEN:* SEEN is also based on random trees, but when SEEN constructs SEENTree, only half of the full features are used to determine the split point. Therefore, SEEN is likely to cause the loss of essential attributes on high-dimensional datasets (such as images).

TABLE II
ACCURACY AND F1 IN THE HIGH α SCENARIO

Dataset	Metrics	SSLDN	SEEN	ORSSL	SENNE	iForest+SVM	LOF+SVM
satimage	accuracy	**0.8371 ± 0.01**	0.8151 ± 0.00	0.7389 ± 0.02	0.8369 ± 0.00	0.8151 ± 0.00	0.6686 ± 0.01
	F1	**0.7746 ± 0.00**	0.7063 ± 0.00	0.7654 ± 0.01	0.6116 ± 0.00	0.7983 ± 0.01	0.4668 ± 0.01
USPS	accuracy	**0.9431 ± 0.00**	0.8529 ± 0.12	0.7920 ± 0.02	0.8114 ± 0.00	0.8934 ± 0.00	0.6271 ± 0.02
	F1	**0.9160 ± 0.00**	0.7359 ± 0.20	0.8175 ± 0.02	0.7716 ± 0.01	0.8417 ± 0.01	0.5493 ± 0.01
HAR	accuracy	**0.9100 ± 0.01**	0.6674 ± 0.00	0.8473 ± 0.02	0.7413 ± 0.00	0.8307 ± 0.01	0.6667 ± 0.02
	F1	0.8739 ± 0.01	0.6681 ± 0.00	**0.8775 ± 0.01**	0.5514 ± 0.00	0.8227 ± 0.01	0.7183 ± 0.00
MNIST-10K	accuracy	**0.8371 ± 0.03**	0.8354 ± 0.03	0.7258 ± 0.02	0.7609 ± 0.00	0.7460 ± 0.04	0.5346 ± 0.02
	F1	0.8230 ± 0.03	**0.8240 ± 0.03**	0.7913 ± 0.01	0.5740 ± 0.00	0.7150 ± 0.05	0.4621 ± 0.02
pendigits	accuracy	**0.8386 ± 0.03**	0.7922 ± 0.01	0.7033 ± 0.04	0.7183 ± 0.01	0.7464 ± 0.01	0.5188 ± 0.01
	F1	**0.8305 ± 0.03**	0.8043 ± 0.01	0.7789 ± 0.02	0.5617 ± 0.01	0.7560 ± 0.01	0.4940 ± 0.00
MNIST	accuracy	**0.7808 ± 0.01**	0.6115 ± 0.00	0.6757 ± 0.02	0.6634 ± 0.00	0.6901 ± 0.03	0.5005 ± 0.01
	F1	**0.7697 ± 0.02**	0.5862 ± 0.00	0.7210 ± 0.02	0.5186 ± 0.00	0.6712 ± 0.02	0.4690 ± 0.00
Fashion-MNIST	accuracy	**0.9042 ± 0.03**	0.8853 ± 0.02	0.7912 ± 0.02	0.7044 ± 0.00	0.7702 ± 0.01	0.5767 ± 0.01
	F1	**0.9005 ± 0.03**	0.8812 ± 0.02	0.8350 ± 0.01	0.6351 ± 0.00	0.7805 ± 0.01	0.5641 ± 0.01
NYTimes	accuracy	**0.9619 ± 0.02**	0.8486 ± 0.03	0.9237 ± 0.02	0.8184 ± 0.01	0.6690 ± 0.06	0.3451 ± 0.02
	F1	**0.9616 ± 0.02**	0.8638 ± 0.02	0.9306 ± 0.02	0.8355 ± 0.00	0.7246 ± 0.02	0.4480 ± 0.01
Avg	accuracy	**0.8766**	0.7886	0.7747	0.7569	0.7701	0.5548
	F1	**0.8568**	0.7587	0.8147	0.6324	0.7638	0.5215

TABLE III
ACCURACY AND F1 IN THE LOW α SCENARIO

Dataset	Metrics	SSLDN	SEEN	ORSSL	SENNE	iForest+SVM	LOF+SVM
satimage	accuracy	**0.8252 ± 0.00**	0.7388 ± 0.00	0.4183 ± 0.05	0.4434 ± 0.02	0.7663 ± 0.02	0.5686 ± 0.01
	F1	**0.6868 ± 0.01**	0.5319 ± 0.01	0.6721 ± 0.01	0.3534 ± 0.00	0.6800 ± 0.01	0.3739 ± 0.01
USPS	accuracy	**0.8702 ± 0.01**	0.6784 ± 0.03	0.7004 ± 0.05	0.7247 ± 0.00	0.6720 ± 0.02	0.5244 ± 0.02
	F1	**0.8453 ± 0.01**	0.6429 ± 0.03	0.7589 ± 0.04	0.6699 ± 0.00	0.6392 ± 0.02	0.4212 ± 0.01
HAR	accuracy	**0.7022 ± 0.02**	0.5620 ± 0.00	0.4838 ± 0.02	0.6976 ± 0.00	0.6336 ± 0.01	0.5389 ± 0.01
	F1	0.6495 ± 0.03	0.4503 ± 0.00	**0.6620 ± 0.01**	0.5205 ± 0.00	0.6139 ± 0.01	0.4630 ± 0.02
MNIST-10K	accuracy	0.6813 ± 0.01	0.6205 ± 0.01	0.5296 ± 0.03	**0.7089 ± 0.01**	0.5267 ± 0.04	0.5478 ± 0.00
	F1	0.6320 ± 0.03	0.5295 ± 0.02	**0.6949 ± 0.01**	0.5168 ± 0.01	0.5877 ± 0.02	0.4802 ± 0.00
pendigits	accuracy	0.8624 ± 0.02	**0.8636 ± 0.03**	0.7035 ± 0.05	0.5256 ± 0.01	0.7612 ± 0.00	0.4494 ± 0.00
	F1	**0.8792 ± 0.02**	0.8699 ± 0.02	0.7920 ± 0.02	0.4218 ± 0.01	0.7932 ± 0.00	0.4861 ± 0.00
MNIST	accuracy	**0.7808 ± 0.01**	0.6115 ± 0.00	0.6757 ± 0.02	0.6634 ± 0.00	0.6901 ± 0.03	0.5005 ± 0.01
	F1	**0.7697 ± 0.02**	0.5862 ± 0.00	0.7210 ± 0.02	0.5186 ± 0.00	0.6712 ± 0.02	0.4690 ± 0.01
Fashion-MNIST	accuracy	**0.7894 ± 0.01**	0.7213 ± 0.03	0.7080 ± 0.02	0.6777 ± 0.00	0.5471 ± 0.07	0.5075 ± 0.00
	F1	**0.7717 ± 0.01**	0.6397 ± 0.02	0.7283 ± 0.01	0.5325 ± 0.00	0.5667 ± 0.05	0.4687 ± 0.00
NYTimes	accuracy	**0.8821 ± 0.02**	0.7397 ± 0.03	0.8579 ± 0.02	0.7736 ± 0.01	0.6970 ± 0.04	0.3842 ± 0.02
	F1	**0.8865 ± 0.01**	0.7545 ± 0.03	0.8747 ± 0.02	0.7716 ± 0.00	0.7551 ± 0.01	0.4539 ± 0.01
Avg	accuracy	**0.8083**	0.6753	0.6407	0.6634	0.6677	0.5040
	F1	**0.8065**	0.6709	0.7578	0.5446	0.6842	0.4455

TABLE IV
ACCURACY AND F1 IN THE RANDOM α SCENARIO

Dataset	Metrics	SSLDN	SEEN	ORSSL	SENNE	iForest+SVM	LOF+SVM
satimage	accuracy	**0.8533 ± 0.01**	0.7804 ± 0.07	0.6292 ± 0.07	0.7292 ± 0.13	0.8092 ± 0.03	0.5900 ± 0.03
	F1	**0.8101 ± 0.02**	0.6783 ± 0.14	0.7441 ± 0.04	0.5172 ± 0.06	0.7450 ± 0.03	0.4586 ± 0.05
USPS	accuracy	**0.8469 ± 0.03**	0.7656 ± 0.00	0.7029 ± 0.07	0.6851 ± 0.12	0.7078 ± 0.13	0.5382 ± 0.02
	F1	**0.8038 ± 0.04**	0.7545 ± 0.02	0.7874 ± 0.04	0.6614 ± 0.02	0.7068 ± 0.11	0.4655 ± 0.02
HAR	accuracy	**0.8463 ± 0.07**	0.3511 ± 0.00	0.8075 ± 0.06	0.5708 ± 0.04	0.6580 ± 0.16	0.3890 ± 0.02
	F1	**0.8522 ± 0.07**	0.4523 ± 0.02	0.8362 ± 0.04	0.4977 ± 0.05	0.7071 ± 0.15	0.4671 ± 0.02
MNIST-10K	accuracy	0.7971 ± 0.04	**0.7993 ± 0.05**	0.6100 ± 0.07	0.7353 ± 0.02	0.6816 ± 0.12	0.5304 ± 0.01
	F1	**0.7650 ± 0.05**	0.7605 ± 0.07	0.7274 ± 0.02	0.5600 ± 0.03	0.6626 ± 0.11	0.4582 ± 0.02
pendigits	accuracy	**0.8578 ± 0.06**	0.8556 ± 0.09	0.7911 ± 0.04	0.7822 ± 0.04	0.7103 ± 0.15	0.5570 ± 0.04
	F1	**0.8527 ± 0.05**	**0.8522 ± 0.09**	0.7785 ± 0.12	0.6676 ± 0.13	0.7174 ± 0.11	0.4575 ± 0.03
MNIST	accuracy	**0.7257 ± 0.06**	0.6078 ± 0.00	0.6779 ± 0.03	0.6803 ± 0.05	0.6397 ± 0.07	0.5096 ± 0.01
	F1	**0.7272 ± 0.05**	0.5546 ± 0.03	0.7209 ± 0.10	0.6009 ± 0.07	0.6291 ± 0.06	0.4654 ± 0.02
Fashion-MNIST	accuracy	**0.8782 ± 0.04**	0.8615 ± 0.05	0.7512 ± 0.14	0.7196 ± 0.02	0.6416 ± 0.09	0.4995 ± 0.01
	F1	**0.8660 ± 0.04**	0.8387 ± 0.05	0.8337 ± 0.03	0.5844 ± 0.01	0.6256 ± 0.08	0.4591 ± 0.02
NYTimes	accuracy	**0.9183 ± 0.05**	0.7907 ± 0.04	0.8652 ± 0.02	0.7470 ± 0.03	0.6276 ± 0.02	0.3486 ± 0.01
	F1	**0.9247 ± 0.04**	0.7991 ± 0.03	0.8796 ± 0.01	0.7536 ± 0.04	0.7124 ± 0.02	0.4431 ± 0.01
Avg	accuracy	**0.8405**	0.7265	0.7294	0.7062	0.6845	0.4953
	F1	**0.8252**	0.7113	0.7885	0.6054	0.6883	0.4593

2) *SSLDN Versus ORSSL:* ORSSL dynamically maintains a set of microclusters, eliminating outdated low-precision microclusters from the model to adapt to the evolving concepts of data streams. However, the microclusters maintained by ORSSL could not capture the hidden local cluster structure for incoming data with non-Gaussian distributions, and they tend to have bad performance on high-dimensional datasets.

3) *SSLDN Versus SENNE:* SENNE addresses the similar problems proposed in this article, but it has the following limitations: 1) the method has a fixed judgment threshold, so it cannot adapt to the continuously generated

TABLE V
AVERAGE RANKS OF THE COMPETING ALGORITHMS VARYING WITH DIFFERENT VALUES OF α

α types	Metrics	SSLDN	SEEN	ORSSL	SENNE	iForest+SVM	LOF+SVM
random	accuracy	**1.1250**	3.0000	3.3750	3.7500	3.8750	5.8750
	F1	**1.0625**	3.3125	2.5000	4.6250	3.6250	5.8750
high	accuracy	**1.0000**	3.1875	3.6250	3.7500	3.4375	6.0000
	F1	**1.2500**	3.2500	2.5000	4.8750	3.3750	5.7500
low	accuracy	**1.2500**	3.2500	4.0000	3.2500	3.8750	5.3750
	F1	**1.2500**	4.0000	2.1250	4.6250	3.3750	5.6250
Avg		**1.1563**	3.3333	3.0208	4.1458	3.5938	5.7500

data stream and 2) SENNE uses supervised data when constructing the initial model, which is often not in line with the actual situation.

4) *SSLDN Versus iForest:* iForest is sensitive to the stream instances because both attributes and split points are selected at random. When the feature values in the data stream differ significantly, the reliability of iForest is reduced.
5) *SSLDN Versus LOF:* LOF is a density-based outlier detection method. However, LOF has a significant detection effect on data with large density differences and has limited adaptation to different datasets.

In summary, due to the effective novel class detection and known classes' classification components, SSLDN performs better than other competing algorithms in solving SSENC problems varying different degrees of separation.

D. Long Data Stream Simulation on NYTimes

In addition to experiments on seven benchmark datasets, we validate the efficacy of SSLDN in real-world data streaming applications. We apply the NYTimes dataset as a long data stream simulation.

The simulation of long data streams on the NYTimes dataset consists of multiple rounds. In the first round, we select two known classes, one novel class from the dataset, and 500 randomly chosen instances for each class (a total of 1500 instances). We still chose two known classes and one novel class in the second round, but 600 randomly selected instances for each class (1800 instances). In the following rounds, each class increases by 100 instances. We finish the simulation until the number of instances for each class is 3300. All selected instances are sorted by the timestamp and flow into the model one by one.

We plot the accuracy and F1 curves regarding different timestamps of these competing approaches, as reported in Fig. 7. For LOF + SVM, it behaves the worst because LOF requires the detected instances that must have a significant density difference. Meanwhile, SEEN, SENNE, and iForest + SVM achieve a comparably satisfying performance. With the increase of data stream, all these competing algorithms can maintain a relatively stable performance. When the data streams reached 4000, SSLDN and ORRSL gradually increased and have remained stable since then. In total, SSLDN can perform better than other competing algorithms, even in a long data stream.

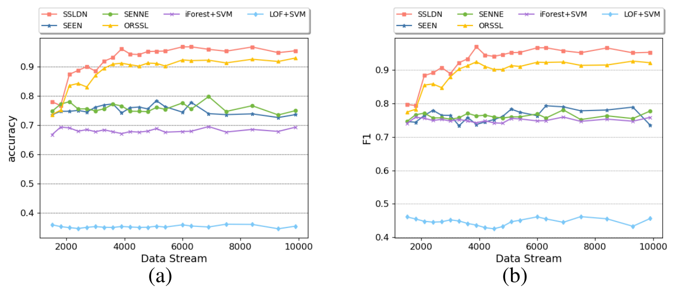


Fig. 7. (a) Accuracy and (b) F1 curves of these competing algorithms on a long data stream simulation.

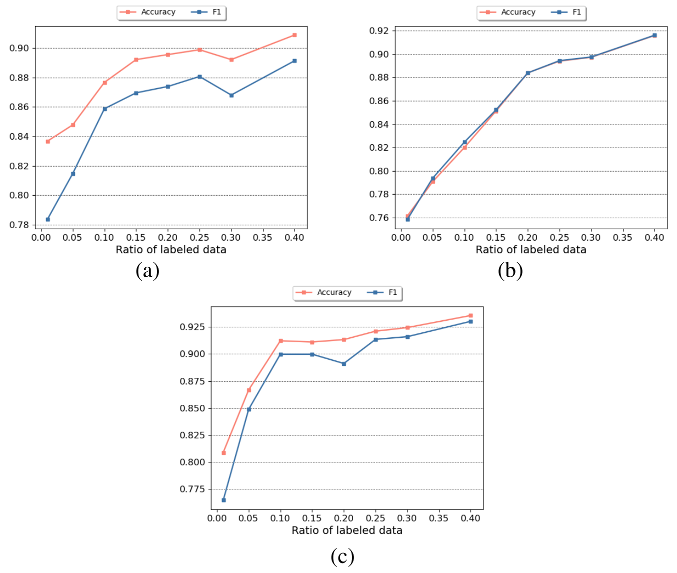


Fig. 8. Influence of the number of labeled instances on three datasets. (a) USPS. (b) HAR. (c) Pendigits.

E. Impact of the Number of Labeled Instances

The number of labeled instances can significantly affect the prediction accuracy of unlabeled stream instances. For SSLDN, we study the impact of the number of labeled instances on three datasets (USPS, HAR, and pendigits), and the results are reported in Fig. 8.

With the increase in the number of labeled instances, both accuracy and F1 have some rapid improvements. On the HAR dataset, both accuracy and F1 keep rising with the ratio of labeled instances. On the USPS and pendigits datasets, there is a period of decline when the ratio is between 0.2 and 0.3. Therefore, more labeled instances do not necessarily mean an increase in performance. However, in real-world applications, the ratio of labeled instances is very rare, often less than 1%. Thus, we choose to use 1% labeled instances in the experiments to simulate semisupervised stream data. Nevertheless, even if only 1% of the data are labeled, SSLDN can perform well on these datasets.

F. Runtime Comparison

We apply the running time comparison in the case of random α that can better reflect real application scenarios.

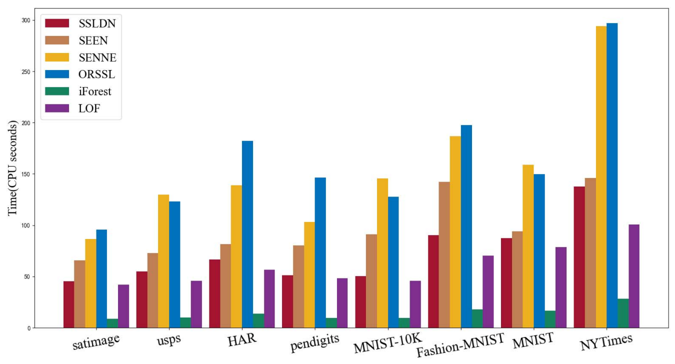


Fig. 9. Runtime comparisons in the random α scenario.

Fig. 9 reports the runtime comparisons of these competing algorithms in the random α scenario.

Fig. 9 shows that SSLDN has the shortest running time on all these datasets. There are two reasons to ensure the efficiency of our new algorithm: 1) only a small number of randomly selected samples are used to construct the SSLDN-Tree in the SSLDN-Forest and 2) for SSLDN-LL, the number of labeled instances is fixed and not all labeled instances are used for classification prediction, which significantly reduces the running time. In contrast, SEEN spends more running time than SSLDN because, when constructing SEENTree, each node must use the K-means algorithm to get two cluster centers. ORSSL dynamically maintains a set of microclusters, eliminating outdated low-precision microclusters from the model to adapt to the evolving concepts of data streams. However, frequent updates and the importance of cluster analysis increase the overall runtime of ORSSL. SENNE needs to construct a hypersphere for each training set when constructing the initial model. In order to obtain the smallest center point covering the test data, SENNE calculates the distance between the test data and a specific class of sampled data, and the distance calculation takes much time. Since iForest and LOF have no updating process, the running time of these two algorithms is only the detection time. Therefore, these two algorithms are faster than SSLDN. To sum up, SSLDN outperforms other competing algorithms on running time, which contains the updating process.

VI. Conclusion

The difficult novel class detection in semisupervised streaming data is a practical and challenging problem. This article proposes the SSLDN framework to tackle the difficult novel class detection in the SSENC problem. The proposed method consists of three main components: 1) a detector based on the random tree that is used to detect novel class; 2) a classifier with restricting the number of labeled instances that can classify unlabeled stream instances accurately; and 3) an efficiency update model. SSLDN makes full use of few labeled instances, automatically updates the threshold, improves detection and classification accuracy, and shortens running time. Empirical studies on several stream simulation datasets validate the effectiveness of SSLDN in handling emerging novel class under a semisupervised streaming data environment. However, SSLDN can handle only one new emerging class during the data stream each time. Thus, we will attempt to deal with multiple new emerging classes in streaming data in our future work. Meanwhile, the connection between concept drift and concept evolution is an exciting issue, and we will study it too.

References

[1] Z.-H. Zhou, "LearnWare: On the future of machine learning," *Frontiers Comput. Sci.*, vol. 10, no. 4, pp. 589–590, Aug. 2016.

[2] M. Sayed-Mouchaweh and E. Lughofer, *Learning in Non-Stationary Environments: Methods and Applications*. London, U.K.: Springer, 2012.

[3] G. Krempl et al., "Open challenges for data stream mining research," *ACM SIGKDD Explor. Newslett.*, vol. 16, no. 1, pp. 1–10, 2014.

[4] X. Mu, F. Zhu, J. Du, E.-P. Lim, and Z.-H. Zhou, "Streaming classification with emerging new class by class matrix sketching," in *Proc. 31st AAAI Conf. Artif. Intell.*, 2017, pp. 2373–2379.

[5] Z. Wang, Z. Kong, S. Changra, H. Tao, and L. Khan, "Robust high dimensional stream classification with novel class detection," in *Proc. IEEE 35th Int. Conf. Data Eng. (ICDE)*, Apr. 2019, pp. 1418–1429.

[6] X. Qin, L. Cao, E. A. Rundensteiner, and S. Madden, "Scalable kernel density estimation-based local outlier detection over large data streams," in *Proc. EDBT*, 2019, pp. 421–432.

[7] A. Cappozzo, F. Greselin, and T. B. Murphy, "Anomaly and novelty detection for robust semi-supervised learning," *Statist. Comput.*, vol. 30, no. 5, pp. 1545–1571, Sep. 2020.

[8] R. Domingues, M. Filippone, P. Michiardi, and J. Zouaoui, "A comparative evaluation of outlier detection algorithms: Experiments and analyses," *Pattern Recognit.*, vol. 74, pp. 406–421, Feb. 2018.

[9] B. Tang and H. He, "A local density-based approach for outlier detection," *Neurocomputing*, vol. 241, pp. 171–180, Jun. 2017.

[10] A. Haque, L. Khan, and M. Baron, "Sand: Semi-supervised adaptive novel class detection and classification over data stream," in *Proc. 30th AAAI Conf. Artif. Intell.*, 2016, pp. 1652–1658.

[11] S. Ud Din, J. Shao, J. Kumar, W. Ali, J. Liu, and Y. Ye, "Online reliable semi-supervised learning on evolving data streams," *Inf. Sci.*, vol. 525, pp. 153–171, Jul. 2020.

[12] Y.-N. Zhu and Y.-F. Li, "Semi-supervised streaming learning with emerging new labels," in *Proc. 34th AAAI Conf. Artif. Intell.*, 2020, pp. 7015–7022.

[13] Z. Zhang, Y. Li, Z. Zhang, C. Jin, and M. Gao, "Adaptive matrix sketching and clustering for semisupervised incremental learning," *IEEE Signal Process. Lett.*, vol. 25, no. 7, pp. 1069–1073, Jul. 2018.

[14] X.-Q. Cai, P. Zhao, K.-M. Ting, X. Mu, and Y. Jiang, "Nearest neighbor ensembles: An effective method for difficult problems in streaming classification with emerging new classes," in *Proc. IEEE Int. Conf. Data Mining (ICDM)*, Nov. 2019, pp. 970–975.

[15] M. M. Masud et al., "Addressing concept-evolution in concept-drifting data streams," in *Proc. IEEE Int. Conf. Data Mining*, Dec. 2010, pp. 929–934.

[16] M. Masud, J. Gao, L. Khan, J. Han, and B. M. Thuraisingham, "Classification and novel class detection in concept-drifting data streams under time constraints," *IEEE Trans. Knowl. Data Eng.*, vol. 23, no. 6, pp. 859–874, Jun. 2011.

[17] X. Mu, K. M. Ting, and Z.-H. Zhou, "Classification under streaming emerging new classes: A solution using completely-random trees," *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 8, pp. 1605–1618, Aug. 2017.

[18] J. Shao, F. Huang, Q. Yang, and G. Luo, "Robust prototype-based learning on data streams," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 5, pp. 978–991, May 2018.

[19] D. Brzezinski and J. Stefanowski, "Combining block-based and online methods in learning ensembles from concept drifting data streams," *Inf. Sci.*, vol. 265, pp. 50–67, May 2014.

[20] V. Losing, B. Hammer, and H. Wersing, "Tackling heterogeneous concept drift with the self-adjusting memory (SAM)," *Knowl. Inf. Syst.*, vol. 54, no. 1, pp. 171–201, Jan. 2018.

[21] S. U. Din et al., "Data stream classification with novel class detection: A review comparison and challenges," *Knowl. Inf. Syst.*, vol. 63, no. 9, pp. 2231–2276, 2021.

[22] D.-W. Zhou, Y. Yang, and D.-C. Zhan, "Learning to classify with incremental new class," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 6, pp. 2429–2443, Jun. 2022.

[23] J. Zhang, T. Wang, W. W. Y. Ng, and W. Pedrycz, "KNNENS: A k-nearest neighbor ensemble-based method for incremental learning under data stream with emerging new classes," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Feb. 25, 2022, doi: 10.1109/TNNLS.2022.3149991.

[24] Y. Song, J. Lu, H. Lu, and G. Zhang, "Learning data streams with changing distributions and temporal dependency," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Nov. 24, 2021, doi: 10.1109/TNNLS.2021.3122531.

[25] M. M. Masud et al., "Detecting recurring and novel classes in concept-drifting data streams," in *Proc. IEEE 11th Int. Conf. Data Mining*, Dec. 2011, pp. 1176–1181.

[26] A. M. Mustafa et al., "Unsupervised deep embedding for novel class detection over data stream," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Apr. 2017, pp. 1830–1839.

[27] Z. S. Abdallah, M. M. Gaber, B. Srinivasan, and S. Krishnaswamy, "AnyNovel: Detection of novel concepts in evolving data streams," *Evolving Syst.*, vol. 7, no. 2, pp. 73–93, Jun. 2016.

[28] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "LOF: Identifying density-based local outliers," in *Proc. ACM SIGMOD Int. Conf. Manag. Data*, 2000, pp. 93–104.

[29] F. Liu, Y. Yu, P. Song, Y. Fan, and X. Tong, "Scalable KDE-based top-n local outlier detection over large-scale data streams," *Knowl.-Based Syst.*, vol. 204, Sep. 2020, Art. no. 106186.

[30] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *Proc. 8th IEEE Int. Conf. Data Mining*, Dec. 2008, pp. 413–422.

[31] R. M. Felder and R. Brent, "Active learning: An introduction," *ASQ Higher Educ. Brief*, vol. 2, no. 4, pp. 1–5, 2009.

[32] Y. Yang et al., "Learning adaptive embedding considering incremental class," *IEEE Trans. Knowl. Data Eng.*, early access, Sep. 2, 2021, doi: 10.1109/TKDE.2021.3109131.

[33] X. Zheng, P. Li, X. Hu, and K. Yu, "Semi-supervised classification on data streams with recurring concept drift and concept evolution," *Knowl.-Based Syst.*, vol. 215, Mar. 2021, Art. no. 106749.

[34] R. G. F. Soares and L. L. Minku, "OSNN: An online semi-supervised neural network for nonstationary data streams," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Dec. 21, 2021, doi: 10.1109/TNNLS.2021.3132584.

[35] F. Nie, G. Cai, and X. Li, "Multi-view clustering and semi-supervised classification with adaptive neighbours," in *Proc. AAAI Conf. Artif. Intell.*, 2017, pp. 2408–2414.

[36] F. Nie, X. Wang, and H. Huang, "Clustering and projected clustering with adaptive neighbors," in *Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2014, pp. 977–986.

[37] X. Zhu, Z. Ghahramani, and J. D. Lafferty, "Semi-supervised learning using Gaussian fields and harmonic functions," in *Proc. 20th Int. Conf. Mach. Learn. (ICML)*. Washington, DC, USA: AAAI Press, Aug. 2003, pp. 912–919.

Peng Zhou received the Ph.D. degree from the Hefei University of Technology, Hefei, China, in 2018.

He is currently a Lecturer with Anhui University, Hefei. His research interests include data mining and knowledge engineering.

Ni Wang received the B.Sc. degree from the Wannan Medical College, Wuhu, China, in 2015. She is currently pursuing the M.Sc. degree with the School of Computer Science and Technology, Anhui University, Hefei, China.

Her current research interests include stream data detection and robust clustering.

Shu Zhao received the Ph.D. degree in computer science from Anhui University, Hefei, China, in 2007.

She is currently a Professor with the Department of Computer Science and Technology, Anhui University. Her current research interests include quotient space theory, granular computing, data mining, and machine learning.

Yanping Zhang received the Ph.D. degree in computer science from Anhui University, Hefei, China, in 2003.

She is currently a Professor with the Department of Computer Science and Technology, Anhui University. Her current research interests include deep learning, quotient space theory, granular computing, and machine learning.

Xindong Wu (Fellow, IEEE) received the Ph.D. degree in artificial intelligence from The University of Edinburgh, Edinburgh, U.K., in 1993.

He is currently a Chang Jiang Scholar with the School of Computer Science and Information Engineering, Hefei University of Technology, Hefei, China. His current research interests include data mining, big data analytics, and knowledge-based systems.

Prof. Wu is fellow of the American Association for the Advancement of Science (AAAS). He served as the Program Committee Chair/Co-Chair for the 2003 IEEE International Conference on Data Mining (ICDM '03), the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-07), and the 19th ACM Conference on Information and Knowledge Management (CIKM2010). He was the Editor-in-Chief of the IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING from 2005 to 2008. He is also the Steering Committee Chair of the IEEE International Conference on Data Mining, the Editor-in-Chief of the *Knowledge and Information Systems* (Springer), and the Series Editor-in-Chief of the *Advanced Information and Knowledge Processing* (Springer Book Series).