

Learning to Classify With Incremental New Class

Da-Wei Zhou^{ID}, Graduate Student Member, IEEE, Yang Yang^{ID}, and De-Chuan Zhan^{ID}

Abstract—New class detection and effective model expansion are of great importance in incremental data mining. In open incremental data environments, data often come with novel classes, e.g., the emergence of new classes in image classification or new topics in opinion monitoring, and is denoted as class-incremental learning (C-IL) in literature. There are two main challenges in C-IL: how to conduct novelty detection and how to update the model with few novel class instances. Most previous methods pay much attention to the former challenge while ignoring the problem of efficiently updating models. To solve this problem, we propose a novel framework to handle the incremental new class, named learning to classify with incremental new class (LC-INC), which can process these two challenges automatically in one unified framework. In detail, LC-INC utilizes a novel structure network to consider the prototype information between class centers of known classes and newly incoming instances, which can dynamically combine the prediction information with structure information to detect novel class instances efficiently. On the other hand, the proposed structure network can also act as a meta-network, which can learn to expand the model much faster and more efficiently with inadequate novel class instances. Experiments on synthetic and real-world datasets successfully validate the effectiveness of our proposed method.

Index Terms—Data stream classification, model adaptation, novel class detection, novelty detection.

I. INTRODUCTION

TRADITIONAL machine learning algorithms often work under static environments, requiring entire training data before conducting the learning process. While in many real-world applications, the environment is dynamic and data are often in stream format, this makes it very difficult to store all the examples over time. Thus, many incremental methods are proposed, e.g., incremental text clustering [1], online video classification [2], and online application activity analysis [3]. While these methods ignore an important issue of

Manuscript received October 6, 2020; revised March 29, 2021 and June 5, 2021; accepted August 11, 2021. Date of publication September 9, 2021; date of current version June 2, 2022. This work was supported in part by the NSFC under Grant 61773198, Grant 61632004, Grant 61921006, and Grant 62006118; in part by the NSFC-NRF Joint Research Project under Grant 61861146001; in part by the Collaborative Innovation Center of Novel Software Technology and Industrialization; in part by CCF–Baidu Open Fund under Grant CCF-BAIDU OF2020011; in part by Baidu TIC Open Fund; in part by the Natural Science Foundation of Jiangsu Province of China under Grant BK20200460; and in part by Nanjing University Innovation Program for Ph.D. Candidate under Grant CXYJ21-53. (Corresponding author: Yang Yang.)

Da-Wei Zhou and De-Chuan Zhan are with the State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China (e-mail: zhoudw@lamda.nju.edu.cn; zhandc@nju.edu.cn).

Yang Yang is with Nanjing University of Science and Technology, Nanjing 210094, China (e-mail: yyang@njust.edu.cn).

This article has supplementary material provided by the authors and color versions of one or more figures available at <https://doi.org/10.1109/TNNLS.2021.3104882>.

Digital Object Identifier 10.1109/TNNLS.2021.3104882

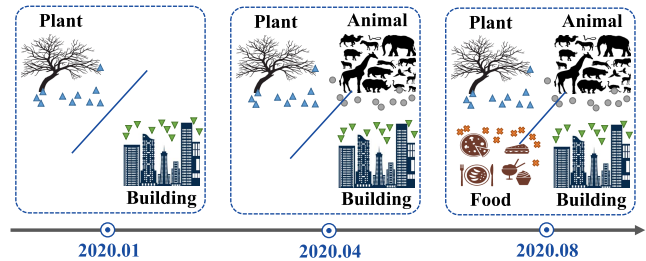


Fig. 1. Illustration of streaming classification with incremental new classes. Unknown class instances, i.e., animal, food, will arise over time, and there is no additional information in advance to know the time and order in which new classes will appear. The original model's performance will suffer decay if novel classes are not detected and the decision boundary stays unchanged.

streaming data, namely the emergence of unknown category instances [4]–[8], e.g., in news topic classification, new topics will arise [9]; in sneaker identification, new brand sneakers will emerge [10]; and in protein sequence classification, new types of proteins would arise as nature evolves [11]. It is noted that novel classes may appear at any time in the stream, and traditional data stream classification techniques cannot detect the novel class until they are trained with labeled novel class instances. Without an expert to detect novel class and expand classifiers manually, all novel class instances will remain misclassified.

Fig. 1 shows an image classification system working with an incremental data stream. The initial model is developed in early 2020, which is trained with two classes i.e., plant and building. Sometime later, new classes like animals and food emerge in the data stream. Considering no prevention, measures like novelty detection and model updating are adopted. The decision boundary will stay unchanged with time, ignoring the emergence of novel patterns. As a result, images of these novel classes will be misclassified by the system, and classification performance will suffer obvious decay. An ideal system should tackle such real-world scenarios, which requires it to automatically detect novel classes and incrementally self-update with detected instances.

There are two main challenges considering the class-incremental learning: 1) novelty detection, i.e., how to effectively detect unknown categories without prior knowledge and 2) model expansion with few novel class instances, i.e., how to update models with only limited new class examples. Several works are proposed to solve the former challenge, e.g., [12] developed isolation tree for nonparametrically separating novel instances apart from known instances. Yang *et al.* [13] proposed a KNN-based approach for topic novelty detection. However, these works only consider the detection process, lacking the ability to update the models with novel instances. Recent works such as [14] and [15] tried to update models with

the detected novel instances in a unified framework. Gruhl and Sick [16] and Gruhl *et al.* [17] proposed mCANDIES, which tackles novelty detection in a modular way with probabilistic models. However, these methods are tree-based or GMM-based and lack the ability in large-scale datasets such as image or video inputs. It takes a huge amount of time for distance calculation or tree structure construction and inefficient in the model update process.

To solve these problems, we propose learning to classify with incremental new class (LC-INC), which can detect the emergence of new classes and update the model with few novel class instances simultaneously. Specifically, LC-INC contains a novel structure comparison network, which aims to learn the ability of learning distance function over input space and the task space. Furthermore, the structure comparison network considers both uncertainty and prediction information to perform new class detection. Moreover, the encoded similarity measure can also be effectively generalized to the learning process of new classes. The main contributions can be summarized as follows.

- 1) We propose a novel strategy to handle the incremental new class, which can process novel class detection and effective model update in one unified framework.
- 2) We propose a novel network, which considers the prototype information between known class centers and incoming instances for novel class detection. The network can also expand efficiently with inadequate instances.
- 3) We empirically evaluate our method on synthetic and real-world scenarios. Experiments against existing state-of-the-art approaches successfully validate the effectiveness of our proposed method.

This article is organized as follows. Section II reviews the main related work. Section III formulates the investigated issue. Section IV describes LC-INC and details each of its elements. Section V presents the datasets used in the experiments, adopted algorithm settings, and empirical evaluations. Section VI discusses feasible solutions with broader stream classification. After that, we conclude the article.

II. RELATED WORK

The task of data stream classification with novel classes mainly involves three related subproblems: 1) detecting emerging novelty classes; 2) classifying known instances; and 3) updating models with detected novel patterns.

Previous novelty detection methods mainly tried to identify novel class or outlier data, focusing on the first subproblem. The novelty detection task [18], [19] can be seen as a two-class classification problem (unknown versus known). Liu *et al.* [12] proposed isolation forest for unsupervised novelty detection, [20] utilized parzen-window for intrusion detection. Yang *et al.* [13] studied a KNN-based approach for topic novelty detection. However, these methods need the whole dataset to process, while data often come with stream format in open-world scenarios, making them hard to be deployed. Recently, several online novelty detection methods have been proposed. Rettig *et al.* [21] proposed online anomaly detection over big data streams and [22] studied unsupervised

real-time anomaly detection for streaming data. These works extend novelty detection to dynamic streams and can work in an online manner. However, the detection and the update module differ in these methods, resulting in error accumulation and difficulty of parameter tuning.

Zero-shot learning (ZSL) [23], [24] is a popular topic in computer vision. The goal of ZSL is to recognize novel categories through the relationships with known classes. It requires auxiliary information like embedding or attributes to predict the real label. ZSL has been found useful in various applications, e.g., face recognition [25], resonance imaging [24], and object recognition [26]. However, when training ZSL models, the data should be precollected, and the auxiliary information like extra embedding is not practical in a real-world data stream. These problems hinder ZSL from being utilized with data streams.

Another line of work focuses on classifying known classes and rejecting novel ones, which is called open-set recognition (OSR) [27], [28]. OSR algorithms aim for the first two subproblems. OpenMax [29] utilized Weibull-based calibration to augment the softmax layer and detect novel classes. Proser [28] proposed to reserve the probability for novel classes during close-set training and transformed closed-set training into open-set training. CPL [30] optimized the embedding with margin-based classification loss and prototype loss for better feature extraction. Like ZSL, the OSR model needs to be trained with the whole dataset and cannot update incrementally. Additionally, when solely updating the embedding module with new classes, the network performance often degrades due to the imbalanced learning scenario [31].

As a result, several works are proposed to tackle these three subproblems in a unified framework, and integrate the newly discovered knowledge into the existing model. ECSMiner [9] introduced time constraints for delayed classification, where the labels of novel instances will reveal after a time constraint. OLINDDA [32] calculated the boundary of each class cluster, and treated instances out of the decision boundary as the novel class. Al-Khateeb *et al.* [33] employed the clustering method to find prototypes of each class and detect novel classes. MNIAS [34] modeled the known classes as microclusters. It calculated the Euclidean distance of a test instance to the closest microcluster centroid, and compared it to the cluster radius to detect novel classes. Learning with augmented class (LAC) [35] proposed tackling the problem with unlabeled data. LAC picked a classification boundary among all low-density separators that minimizes the empirical risk and structure risk and augment risk simultaneously. However, these methods need additional information, e.g., true labels, subsets of the original data, or extra unlabeled dataset. Heuristically saving subsets consumes huge memory and results in complex selection costs. Gruhl *et al.* [17] proposed mCANDIES, which combined the advantages of different approaches for anomaly identification by exploiting locality in different input space regions. Wang *et al.* [36] studied novelty detection under multilabel settings and proposed deep streaming label learning (DSLL) to effectively classify instances with newly emerged labels. Zhang *et al.* [37] proposed a setting where unknown instances emerge in the training dataset, which

focused on feature quality deficiency. Mu *et al.* [14] studied an isolation-based anomaly detection method SENCForest to construct the classifier and detector. However, the adopted tree structure is not suitable for high-dimensional inputs like natural images. SENNE [15] adopted nearest neighbor ensemble, which is the most relevant work to our approach. However, the parameters in the ensemble are hard to tune in deployment.

There are other topics in data stream mining [38]. Gomes *et al.* [39] discussed mining recurring concepts in a dynamic feature space. He *et al.* [40] handled the data streams with a varying feature space. Rutkowski *et al.* [41] proposed a new method for constructing decision trees for stream data. In this article, we focus on solving the three subproblems and discuss the broader streaming data mining in Section VI.

III. PRELIMINARIES

Before introducing the details of our proposed method, we give definitions of the important concepts in this article.

Definition 1 (Classification With Emerging Novel Classes): Given the initial training set $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^L$, where $\mathbf{x}_i \in \mathbb{R}^d$ is a training instance, and $y_i \in Y = \{1, 2, \dots, k\}$ is the associated class label; and the data stream $S = \{(\mathbf{x}'_i, y'_i)\}_{i=1}^\infty$, where $\mathbf{x}'_i \in \mathbb{R}^d$ and $y'_i \in Y' = \{1, 2, \dots, k, k+1, \dots, K\}$ with $K \gg k$. First, the algorithm should train an initial model \mathcal{M} with D ; then \mathcal{M} is used as a detector for novel class and a classifier for known class. The model \mathcal{M} is updated timely, and the goal is to maintain high performance for novel and known classes with the data stream S evolving.

Besides, there are two settings as [14] and [9]: 1) the hardest scenario is that true class labels are not available throughout the entire data stream S , except for the initial dataset D and 2) a relaxation can be made when true class labels are available at some intervals in the stream. Considering the real-world applications, the two settings correspond to the scenario with or without artificial regulation. The model would be better trained if someone occasionally labels instances in the stream.

Definition 2 (Novel Class): If a class is not available in the initial training dataset D , and it only emerges in the data stream S ; then it is called a novel class. The phenomenon of novel classes emerging is also known as concept evolution [9].

In the real-world scenario, multiple novel classes may emerge in the data stream simultaneously, i.e., several novel classes emerge in a short period. Similar to the existing methods [9], [14], [35], we first treat all the novelty as a single meta-class; and then update the model by treating them as a united one.

Definition 3 (Novelty Score): During the data stream S , for each instance \mathbf{x} , the model \mathcal{M} produces a score which determines \mathbf{x} as belonging to either a known class or an emerging novel class. Instances with higher novelty scores are more likely to be novel classes.

Definition 4 (Fixed-Size Buffer): If the model detects a new instance as a novelty, it stores the instance temporally in the buffer. The buffer is empty at the beginning of the data stream, and then gradually filled up with detected novel instances. We observe sufficient anomalies over a certain period of time in such a way that the buffer is filled by instance from a single novelty. When the buffer reaches its maximum capacity s ,

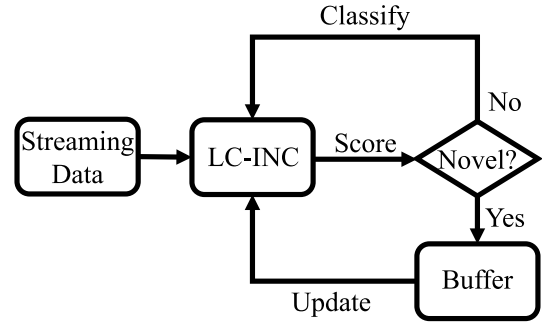


Fig. 2. Illustration of our framework. Streaming data is fed into LC-INC as input; the model scores each instance and predicts if it is novel. Known instances will be passed to LC-INC to be classified, i.e., the “classify” branch. Novel instances will be saved in a fixed-size buffer, and the model will update timely with the buffered instances, i.e., the “update” branch.

the saved instances will then be used for model update. After that, we empty the buffer and go on with the streaming data.

IV. PROPOSED METHOD

Facing a data stream with emerging novel classes, a suitable model should be capable of accurately classifying known classes, detecting novel classes, and quickly self-updating. Our main intuition is to represent each known class with a selected exemplar. When facing a test instance in the stream, we compare it to all the class prototypes and find the most similar one. If an instance is not similar to any class, we recognize it as novel and save it in the buffer. Additionally, if the buffer is full of saved instances, we then regard it as the emergence of a novel class and update the model. In this section, we will first introduce the framework of LC-INC. After that, we separately discuss the subproblems, i.e., novel class detection and model extension. We finally summarize the section with implementation guidelines.

A. Framework Overview of LC-INC

LC-INC aims to detect novel candidates and classify known instances in the data stream. Fig. 2 shows the framework. Streaming data emerges and LC-INC scores each instance with the probability of being a novel class. Instances predicted as known classes would be passed to the classifier for prediction, while those predicted as novel classes are saved in the buffer and used to update the model. Since new classes are well detected and learned, LC-INC can learn to classify with emerging new classes in the data stream.

In detail, we design a novel deep network to model the distribution of every known class. Two main components constitute the network: the *embedding* module $\mathcal{E}(\cdot)$ and the *structure comparison* module $\mathcal{C}(\cdot)$. The *embedding* module is utilized to extract features of the input instance. The *structure comparison* module is designed to compare the similarity between a pair of features. *The main idea of LC-INC is to compare the test instance to every known class, and decide the class affiliation via similarities.* To achieve this goal, we first introduce how to represent the known classes.

1) *Representing Known Class With Class Center:* In the data stream, class centers can be maintained as prototypes to represent every class, leading to the class center set

$C = \{\mathbf{c}_1, \mathbf{c}_2, \dots, \mathbf{c}_K\}$. Then for class i , the centroid of all instances belong to this class can be calculated as: $\mathbf{c}_i = 1/(|\Omega_i|) \sum_{\mathbf{x} \in \Omega_i} \mathbf{x}$, where Ω_i is the set of instances belonging to the i th class, and $|\cdot|$ calculates the set size. Considering the property of data stream, class centers can be updated incrementally

$$\hat{\mathbf{c}}_i = \mathbf{c}_i + \frac{\mathbf{x} - \mathbf{c}_i}{|\Omega_i| + 1}. \quad (1)$$

It is noted that for a novel class, the class center will be initialized with an all-zero vector. The class centroid models the average pattern of the class, which can also be viewed as the most common feature of the class. As a result, we can represent the known classes with class centers.

2) *Embedding Module*: Now we represent each known class with the class center, we further utilize deep network as embedding module to project the class centers into feature space: $\mathcal{E}(\cdot) : \mathbb{R}^d \rightarrow \mathbb{R}^{\hat{d}}$, $\hat{d} \ll d$. Facilitated by the powerful representing ability of deep networks, the embedded features can preserve the semantic relationships of the input. In the embedding space, features of similar/same classes have small distances, while dissimilar/different classes have large distances. The other components of LC-INC rely on high-quality features provided by $\mathcal{E}(\cdot)$.

3) *Structure Comparison Module*: Looking back to our motivation that we predict the class belonging via comparing to all known classes, and we need a structure comparison module $\mathcal{C}(\cdot)$ to serve as the role of ‘‘comparator.’’ Facing a test instance \mathbf{x}_j , the comparison module measures the similarity of $(\mathcal{E}(\mathbf{x}_j), \mathcal{E}(\mathbf{c}_i))$ for every known class in the embedding space, i.e., $i = 1, 2, \dots, K$. The class prototypes represent the common features of the specific class. As a result, a test instance shall have a strong correlation/similarity with the corresponding class center. For other nontarget class centers, it shall have small correlations. Consequently, we output the most similar class center as the predicted class.

Unlike the traditional fixed metric, we *build the structure comparison module with a three-layer fully connected network*, i.e., the output of the three-layer network is the predicted similarity of the input feature pair. We encode similarity measure as such a network in a data-driven way. A sigmoid function is attached to the last layer of $\mathcal{C}(\cdot)$, making the output similarity between (0, 1). We discuss the advantages of such *learnable* similarity metric in Section IV-C.

4) *Bilinear Pooling Module*: For now, we have introduced the main components of LC-INC, and a simple implementation is to feed the original feature pairs $(\mathcal{E}(\mathbf{x}_j), \mathcal{E}(\mathbf{c}_i))$ to the structure comparison module, achieving the similarity vector. However, there is still an essential part in the model, i.e., Bilinear pooling module $\mathcal{B}(\cdot, \cdot)$. Bilinear pooling [42] is an effective tool for feature fusion, which provides more powerful aggregated features than original ones. Unlike simply concatenating the extracted feature maps, bilinear pooling combines the features using the matrix outer product and average pools them at each location. As a result, it provides stronger features than linear aggregation and can be optimized end-to-end. We provide the details about bilinear pooling in the appendix.

5) *Dataflow of LC-INC*: Our model classifies each instance via the similarity to every class center. As a result, each time a new instance \mathbf{x}_j emerges, we embed it and known class centers with the embedding module: $\mathcal{E}(\mathbf{x}_j)$ and $\mathcal{E}(\mathbf{c}_i)$, $i = 1, 2, \dots, K$. The similarity among these feature maps indicates the semantic relationship of the input space. Since we need to compare it to the class centers, we fuse the embedded features with bilinear pooling, i.e., $\mathcal{B}(\mathcal{E}(\mathbf{c}_i), \mathcal{E}(\mathbf{x}_j))$. It is noted that we have K class centers, and the fusion process will repeat K times, yielding K fused feature pairs between the input instance and every class center. After that, the fused features are fed into the structure comparison module and get the similarity measure

$$r_{i,j} = \mathcal{C}(\mathcal{B}(\mathcal{E}(\mathbf{c}_i), \mathcal{E}(\mathbf{x}_j))), \quad i = 1, 2, \dots, K \quad (2)$$

where $r_{i,j}$ stands for the calculated similarity between instance \mathbf{x}_j and class center \mathbf{c}_i . We can get the similarity vector $[r_{1,j}, r_{2,j}, \dots, r_{K,j}]$, standing for the similarity of the test instance to every known class. We normalize it to produce the probability format and enable loss calculation

$$\hat{\mathbf{y}}_j = \text{softmax}([r_{1,j}, r_{2,j}, \dots, r_{K,j}]). \quad (3)$$

In the initial training phase, we optimize the model and get suitable $\mathcal{C}(\cdot)$, $\mathcal{E}(\cdot)$ by optimizing the cross-entropy between ground truth label \mathbf{y}_j and the normalized similarity vector $\hat{\mathbf{y}}_j$

$$L_j = - \sum_{i \in \text{known}} y_{i,j} \log \hat{y}_{i,j} \quad (4)$$

where \mathbf{y}_j is the one-hot vector and $y_{i,j}$ is the corresponding i th element of \mathbf{y}_j . During the training process, we optimize the model with (4). For a known class instance, the model maximizes the similarity score of \mathbf{x}_j being associated with a similar prototype, and minimizes the similarity score of \mathbf{x}_j being associated with a dissimilar one. As a result, it decreases the distance between an incoming instance with the centroid from its genuine class. In this way, the embedding module and structure comparison module will be optimized. The network structure is shown in Fig. 3. We then introduce the novel class detection algorithm.

B. Novel Class Detection

Classical novel class detection methods rely on predict uncertainty or structure information. However, no historical data can be fetched in the data stream, and the structure information of these novel classes is hard to fit, which fails these nonrobust methods. On the other hand, with novel class instances accumulating and model updating, prediction information can be more reliable. As a result, we should *adaptively combine the prediction information with structure information* to conduct novel class detection.

During the streaming data deployment, instances often come with the timestamp (e.g., we treat the index t of \mathbf{x}_t as the timestamp), with which we can divide the whole data stream into several chunks. For example, instances from 0 to chunksize belong to the zeroth chunk, chunksize + 1 to chunksize * 2 belong to the first chunk, and so on. If the time information is not available, then chunks can be defined by the

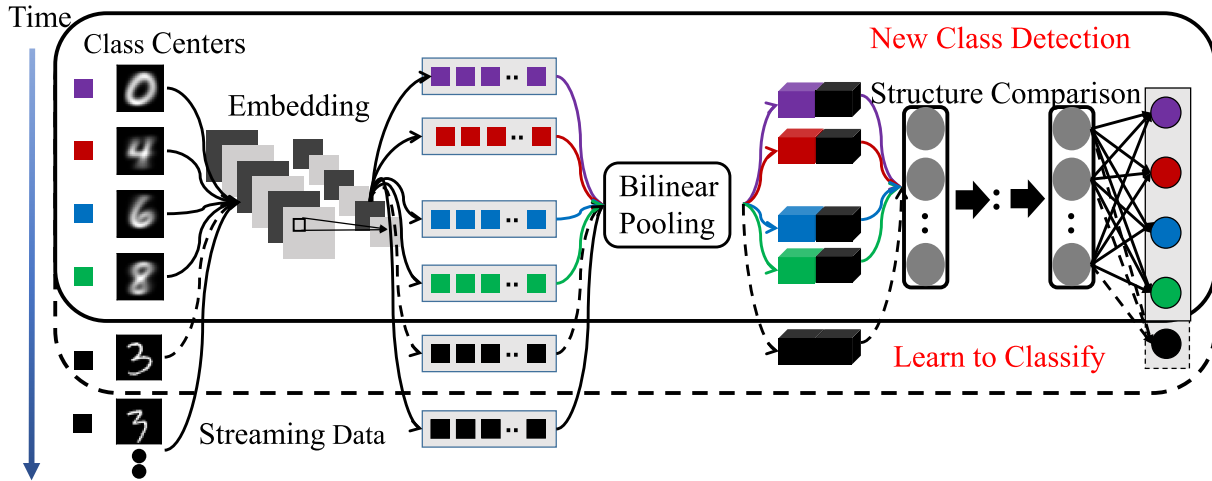


Fig. 3. Illustration of the proposed LC-INC. Class centers are fed as inputs for embedding along with an incoming instance to generate feature maps. Then we adopt bilinear pooling to fuse incoming instance feature maps with each class center. Finally, we calculate the similarity between each pair with the structure comparison module, obtaining K similarity scores, each stands for the probability that the incoming instance belongs to the corresponding class. LC-INC assigns a label of the incoming instance to the most similar class.

latest chunksize instances observed in the stream. We maintain a fixed-size buffer B to save the detected novel instances and use them for the model update. When the buffered instances reach buffer size, i.e., $|B| = s$, we update the model and then empty the buffer. We then introduce how to detect new classes.

1) *Entropy Measure*: Each time an instance arrives in the data stream, we calculate the similarity vector with (3). A known class instance is similar to the corresponding class centroid while dissimilar to other class centroids, yielding substantial similarity to ground truth and tiny similarity to the other classes. Nevertheless, for a novel class instance, we have no prior about what this class is like. As a result, this novel instance is not similar to any one of the known class centers, and the similarity scores to every class prototype tend to be close [43]. Evidently, the model tends to output a *certain* prediction for a known instance while producing an *uncertain* prediction for a novel instance. We can differentiate novel classes from known ones by prediction uncertainty, and use *information entropy* to measure the uncertainty of prediction

$$\text{Ent}(\mathbf{x}_t) = - \sum_{i \in \text{known}} \hat{y}_{i,t} \log \hat{y}_{i,t} \quad (5)$$

where \mathbf{x}_t stands for the t th instance in the data stream. Instances with higher prediction entropy are more likely to be novel than instances with lower entropy.

2) *Probability Measure*: On the other hand, we update our model with the buffered items when the buffer is full. With detected sufficient novel instances, we can build the prototype of the novel class and *accumulate the prior what the novel class is like*. As we constructed the prototype for the new class, we can also compare the incoming instance with the prototype of it. As a result, if an instance is similar to the novel class prototype, it can be a novelty with a high probability. We also use the normalized similarity to novel class as the

probability measure for novel class detection

$$\text{Prob}(\mathbf{x}_t) = \hat{y}_{k+1,t}. \quad (6)$$

3) *Dynamic Tradeoff*: As discussed before, $\text{Ent}(\mathbf{x}_t)$ measures the uncertainty of an instance facing different class prototypes, and $\text{Prob}(\mathbf{x}_t)$ measures how relative an instance is similar to the novel class center. To utilize both uncertainty information and structure information to detect novelty, we should *simultaneously consider the entropy and probability measure*.

We argue that during the evolution of streaming data, entropy and probability *play different roles at different stages*. In the beginning, the model is trained for several known classes, and unaware of the class center of the novel class. After model updating, we gradually update the novel class prototype and have a more precise estimation of the novel class center. As a result, the probability measure may be less effective at the early stages, and it is urgent to pick instances with higher entropy. As the data stream evolves, the model has accumulated prior information and builds a proper prototype for the novel instances. It is now meaningless to pick those instances with higher entropy, and we should attach a higher weight to the probability measure. Based on this motivation, we emphasize more on the entropy measure during the early stages and gradually increase the weight on probability measure. Formally, we employ a tradeoff parameter λ to dynamically balance these two criteria as the data stream evolving

$$\text{Score}(\mathbf{x}) = \min(\lambda c, 1) \text{Prob}(\mathbf{x}) + \max(1 - \lambda c, 0) \text{Ent}(\mathbf{x}) \quad (7)$$

where c is the chunk number starting from 0 at every novel class period and the max/min operator keeps the weights between (0, 1). Finally, instances with the larger score than a defined threshold will be predicted as novel instances, and further used to update the model. Correspondingly, instances with lower scores are fed to the model to find the most similar

Algorithm 1 Novel Class Detection**Input:**

- Streaming data: $S = \{\mathbf{x}_t\}_{t=1}^{\infty}$

Output:

- Predicted label of each \mathbf{x}_t

```

1: Initialize buffer with empty set  $B \leftarrow \emptyset$ ;
2: repeat
3:   Get a test instance  $\mathbf{x}_t$ ;
4:   Calculate the instance  $\text{Score}(\mathbf{x}_t) \leftarrow \text{Eq. 7}$ ;
5:   if  $\text{Score}(\mathbf{x}_t) > \text{threshold}$  then
6:      $y_t = k + 1$ . Predict as a novel class item;
7:      $B = B \cup \mathbf{x}_t$ ;
8:   else
9:      $y_t = \underset{k}{\text{argmax}}(\hat{y}_{k,t})$ . Predict as a known class item;
10:  end if
11:  if  $|B| = s$  then
12:    Update model  $\mathcal{M}$  with  $B$  use Algorithm 2;
13:     $B \leftarrow \emptyset$ ;
14:  end if
15: until Streaming data is empty

```

class. We will talk about the threshold setting in Section IV-D. In summary, through the dynamic tradeoff between structure information and prediction information, the model can adaptively combine entropy and probability measures during different periods. The algorithm for novel class detection is shown in Algorithm 1.

4) *Discussion About Multiple Novel Classes*: In real-world applications, multiple novel classes may emerge simultaneously, increasing the confusion of novelty detection. If multiple novel classes emerge in a short period, the buffer will contain instances from more than one class. Under such circumstances, we provide two feasible ways to handle multiple novel classes. First, if the learning system cannot get extra supervision, i.e., manually labeling the novel class, we can simply regard multiple novel classes as a union of meta-novel class. Second, if the system can get extra labels for the buffered instances, we can separately update the novel class center for each novel class. These two conditions correspond to the settings in Definition 1, and we handle both of them in the experiments.

C. Learning to Extend With New Classes

After detecting novel classes, another important topic of data stream classification with novel classes is extending the model and learning to classify new classes. In LC-INC, we update the model when the buffer is full. Limited by the buffer size, *we do not have plenty of novel class instances to perform the model update*. As a result, the model should quickly *adapt to new classes with inadequate instances* available. Facing such restrictions, traditional algorithms, e.g., decision trees or KNN-based methods rely on the structure information to model new classes and may fail to depict the novel class distribution with insufficient instances. On the other hand, deep models may fall overfitting and lack the generalization ability for unseen samples. By contrast,

Algorithm 2 Learn to Extend**Input:**

- Dataset: $D = \{(\mathbf{x}_j, y_j)\}_{j=1}^m$
- Class center set: $C = \{\mathbf{c}_1, \mathbf{c}_2, \dots, \mathbf{c}_K\}$

Output:

- Updated model: \mathcal{M}

```

1: for each incoming instance  $(\mathbf{x}_j, y_j)$  do
2:   Update class center  $\leftarrow \text{Eq.1}$ ;
3:   Calculate the embedding of each class center  $\mathcal{E}(\mathbf{c}_i), i = 1, 2, \dots, K$ ;
4:   Calculate the embedding of the incoming instance  $\mathcal{E}(\mathbf{x}_j)$ ;
5:   Calculate the bilinear pooling of  $\mathcal{E}(\mathbf{x}_j)$  with each class center  $\mathcal{E}(\mathbf{c}_i)$ :  $\mathcal{B}(\mathcal{E}(\mathbf{c}_i), \mathcal{E}(\mathbf{x}_j)) \quad i = 1, 2, \dots, K$ ;
6:   Calculate the structure similarity of each bilinear pooling pair:  $\mathcal{C}(\mathcal{B}(\mathcal{E}(\mathbf{c}_i), \mathcal{E}(\mathbf{x}_j))) \quad i = 1, 2, \dots, K$ ;
7:   Calculate the loss  $L_j \leftarrow \text{Eq. 4}$ ;
8:   Obtain the derivative  $\frac{\partial L_j}{\partial \Theta}$ , update the model;
9: end for

```

our model is transferable and feasible to reuse in terms of embedding and structure comparison.

1) *Model Adaptation With Inadequate Instances*: Essentially, the structure comparison network $\mathcal{C}(\cdot)$ acts as a comparator, which inputs feature pairs and outputs the task-related similarity vector. It is designed for learning the ability of learning distance function over input/feature space and the task/similarity space. Since it aims at producing the similarity measure for input pairs, it is born transferable and feature-independent. To be specific, no matter what kind of class pairs are fed as the input of $\mathcal{C}(\cdot)$, it only produces the learned similarity metric, *reflecting the pairwise relationship*. As a result, we can train a generalizable structure comparison network during the initial training process with few known classes. Afterward, it can generalize to the similarity calculation between unknown novel class pairs. Additionally, once we calculate the class center with buffered novel classes and add it as the input, the structure comparison module can calculate the similarity between the novel class center and the incoming input. We simply learn to classify new classes by leveraging the generalization ability of the structure comparison module.

2) *Advantages of Trainable Similarity Measure*: One may query why we adopt the structure comparison module instead of traditional fixed metrics. Considering other ways to produce structure comparison modules, like shallow Mahalanobis metric for feature representation or cosine/Euclidean distance for classification, the metrics or features are *fixed* in the learning process. By contrast, our framework can be seen as both learning a deep embedding $\mathcal{E}(\cdot)$ and a deep nonlinear metric $\mathcal{C}(\cdot)$. These two parts are mutually tuned end-to-end to facilitate each other throughout the learning process. A flexible function approximator calculates the similarity, and we learn a good metric in a data-driven way. There is no need to manually choose the proper metric between Euclidean, cosine, or Mahalanobis, which heavily depends on the performance of the embedding network. Under such circumstances, if the learned representations of the embedding

module are not discriminative, the united model’s performance will be limited. By contrast, with the structure comparison module, a nonlinear similarity metric jointly with the embedding module is optimized for better recognition.

To conclude, we provide the algorithm of model updating in Algorithm 2. For every detected novel instance, we first generate the prototype to represent this class through (1). With more data chunks passes by, we gradually have a precise estimation of new classes and fine-tune the network. The generalizable and transferable similarity measure helps the model even with insufficient novel instances.

D. Guideline for Implementation

We have discussed the data flow and algorithms for detecting and learning new classes. In this section, we discuss how to set the parameters and the intuition of choosing class prototypes.

1) *Threshold and Buffer Size*: In Algorithm 1, two parameters will influence the detection performance, i.e., the score threshold and the buffer size. Especially, the threshold should separate the known from the unknown, and we provide a heuristic way to define the threshold. Traditional algorithms for novelty detection rely on an extra validation set to define the threshold [43]. However, in the data stream, it is infeasible to obtain an extra validation set, and we propose to define the threshold without the validation set. Since the stream is evolving, after the learning process of each period, we tune the threshold to ensure 95% of the training data to be recognized as known. The buffer size influences the frequency of the model update. Adopting a larger buffer size, the model will update with more instances and correspondingly require more memory. Correspondingly, a smaller buffer needs less memory, but it updates more frequently with inadequate instances. In our experiments, we find that the performance is almost robust for buffer size around 200, and we suggest it as default value.

2) *Effects of Class Centers*: In (1), we propose to represent each known class with the Euclidean vectors that describe mean values of class instances in the input-space. We can also extract class prototypes in the feature space, i.e., $\hat{\mathbf{c}}_i = \mathbf{c}_i + (\mathcal{E}(\mathbf{x}) - \mathcal{E}(\mathbf{c}_i)) / (|\Omega_i| + 1)$. However, since the stream is evolutive and the embedding module is updating incrementally, there is a gap between the heterogeneous features in different time stages. To avoid the accumulation of such feature drift, we use the centroid of input space instead of feature space to serve as the class prototype.

V. EXPERIMENTS

In this section, we first introduce the datasets and then show how to simulate the data stream. Finally, we give the empirical results of LC-INC and compared methods.

A. Setups

1) *Datasets and Configurations*: We evaluate our proposed method with a synthetic 2-D stream, four benchmark datasets and one real-world incremental dataset:

Synthetic [14]: A 2-D dataset contains 10 000 instances and has three overlapping Gaussian distribution;

KDDCUP99 [44]: Consists of four gigabytes of compressed binary TCP dump data from seven weeks of network traffic. We extracted 16 largest classes for simulating a long data stream;

MNIST [45]: Has a training set of 60 000 examples. We use them from ten classes to simulate data stream;

Fashion-MNIST [46]: Is a dataset of Zalando’s article images—consisting of 60 000 examples, we use these examples from ten classes to simulate data stream;

HAR [47]: Is an activity recognition database, which consists of 6 classes of data with 561 dimensions;

YSneaker [10]: Consists of 12 000 sneaker images crawled over six months from the internet. We extract six largest classes, i.e., Nike, Adidas, Jordan, Puma, Reebok and Converse, and use pretrained resnet18 [48] to extract features toward 512 dimensions. Since images were collected within 6 months, they can be easily simulated into a data stream with the timestamp.

2) *Data Stream Simulation*: We follow the common setting in [14] to simulate the data stream. We randomly choose two classes out of all available classes as known ones in the initial training stage, and each of them contains 2500 instances. We randomly select the instances and data sequence to form the initial training set, following uniform class distribution. The emerging new classes in each period are different from the known classes, which are randomly sampled from the rest classes. After learning the initial two classes, the model will be deployed in the data stream. In the beginning, instances from one novel class and these two known classes will emerge in the stream with uniform distribution. The model should detect novel instances and self-update within the first period to learn the novel pattern. After that, instances of the three known classes and another novel class will emerge with uniform distribution. In each period, the total number of known and novel class instances is set to 4000. Every model is aware of the current data period, and will not mix different novel classes up as one class. Following the rule of data stream processing, instances appear sequentially in the stream, and the model should predict for every instance before processing the next. To illustrate the distribution of experimental streams clearly, we present an example in Fig. 4(a), corresponding to one simulated MNIST stream. Each dataset is used to simulate a data stream over five trials, and both mean and standard variance is reported.

Multiple novel classes may arise simultaneously, and an ideal system should detect components of different clusters and learn to recognize them. We also simulate multiple novel classes emerging at the same time and conduct experiments. One simulated stream of multiple novel classes can be found in Fig. 4(b).

3) *Compared Methods*: We compare LC-INC with other common-used novelty detection algorithms and methods which combine novel class detection with model update. In detail, they are listed as:

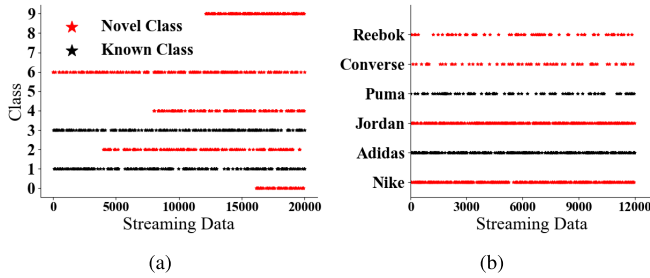


Fig. 4. Illustration of simulated streams. X-axis stands for stream timestamp, and Y-axis stands for the class label. Instances from known classes are in black color, and novel classes are in red color. (a) One simulated MNIST stream. In this trial, only one novel class emerges in each period. (b) One simulated YSneaker stream. In this trial, multiple novel classes emerge simultaneously. Legends are shown in (a).

- 1) *iForest + KNN* [12]: An unsupervised anomaly detector with isolation forest. Since it can only detect novelty, we combine it with KNN as the classifier.
- 2) *ODIN* [49]: A deep out-of-distribution detection algorithm, which uses temperature scaling and input perturbation to detect out-of-distribution items.
- 3) *LACU-SVM* [35]: By adopting the one-versus-rest approach, it picks a classification boundary among all low-density separators that minimizes the empirical risk and structure risk as well as the augment risk simultaneously.
- 4) *mCANDIES* [17]: A modular novelty detection framework encapsulating different detectors from the current SOTA.
- 5) *ECSMiner* [9]: An ensemble framework uses K nearest neighbor as the classifier to make predictions, a new measure is defined to detect novelty.
- 6) *CLAM* [33]: By clustering instances into several groups, it finds representative instances for each class, and utilizes these prototypes in a condensed nearest neighbor classifier.
- 7) *SENCForest* [14]: An unsupervised anomaly detection-focused approach, coupled with an integrated method using completely random trees.
- 8) *SENNE* [15]: Uses nearest-neighbor ensembles for SENC, which requires storing a buffer of instance subsets.
- 9) *CPL* [30]: Uses the same structure network as LC-INC for embedding, and adopts distance-based cross-entropy loss to improve the intraclass compactness of the feature.
- 10) *DNN-Based* [50]: A deep network uses the same structure as LC-INC. By defining a threshold, we use the predicted entropy as the indicator to detect novel classes. When it needs an update, we expand the output layer and then fine-tune it with incoming instances.

We implement ECSMiner and CLAM based on the original article, and use the source codes release by the corresponding authors for other methods. We grid search the optimal parameters of these compared methods with bootstrapping. Details are provided in the appendix.

4) *Implementation Details*: Our model is implemented with PyTorch [51] 1.6.0 on NVIDIA RTX 2080-Ti GPU. In the initial training process, we optimize the model with SGD for

ten epochs. The learning rate starts from 0.01 for 5 epochs and then 0.001 for later 5 epochs. The batch size is set to 128, and we set weight decay to 0.001. The buffer size s is set to 200, and the chunksize is set to 500 for all datasets. For nonimage datasets, i.e., Synthetic, HAR, and KDDCUP99, we simply use a three-layer fully connected network as the embedding module \mathcal{E} . The embedding dim is set to 64. For image datasets, i.e., MNIST and Fashion-MNIST, we use a three layer-convolutional network as the embedding module, and the embedding dim is set to 128. We implement the Bilinear pooling module \mathcal{B} according to [42], and set the output dimension to 256 for all datasets. We adopt a three-layer fully connected network as the structure comparison module, which takes the pooling result of 256 dimensions as input, and outputs the similarity prediction. A sigmoid function is attached to the structure comparison module to ensure the similarity between 0 and 1.

5) *Evaluation Measures*: To evaluate the prediction performance of the model during the data stream, we adopt two evaluation measures, i.e., Accuracy and F-Measure. These metrics are widely adopted by Mu *et al.* [14], Cai *et al.* [15], Wang *et al.* [52] to estimate the performance. Even though these measures take no consideration of temporal behavior or class imbalance, we report these metrics for a fair comparison to related work. Let m be the total length of the stream, A_{novel} be the number of novel instances identified correctly, and A_{known} be the number of known class instances classified correctly, we have Accuracy = $(A_{\text{novel}} + A_{\text{known}})/m$. We also measure the performance with F-Measure, which is calculated at several intervals, and use the average as the final evaluation. F-Measure = $(2 * P * R)/(P + R)$, which is defined jointly by precision ($P = (\text{TP})/(\text{TP} + \text{FP})$) and recall ($R = (\text{TP})/(\text{TP} + \text{FN})$). TP means true positive, FP means false positive, and FN means false negative. We calculate F-Measure at the end of each period.

B. Synthetic Streaming Data Classification

To visualize the discrimination ability of LC-INC, we first simulate a synthetic short stream to show the novel class detection and classification results intuitively. We generate a 2-D synthetic dataset in Fig. 5(a), consisting of three overlapping Gaussian distribution instances. We treat two of them as known classes and the other as the novel class to simulate the data stream.

Results about Accuracy and F-measure are reported in Table I. LC-INC can get better performance compared to other methods with a substantial margin. We also visualize the prediction results in Fig. 5. Comparing to the runner-up method SENNE and anomaly detection method iForest, LC-INC predicts more similar to the ground truth. The visualization also validates the feasibility of representing each class with the class centroid.

C. Real-World Streaming Data Classification

The stream prediction results, which calculate the Accuracy and F-Measure of the data stream, the results of LC-INC and other compare methods on four benchmark datasets are

TABLE I

RESULT (MEAN \pm STANDARD DEVIATION) OF DIFFERENT COMPARED METHODS ON SIMULATED STREAMS. THE BEST PERFORMANCE IS BOLDED

Methods	Synthetic		MNIST		Fashion-MNIST		HAR		KDD CUP 99	
	F-Measure	Accuracy	F-Measure	Accuracy	F-Measure	Accuracy	F-Measure	Accuracy	F-Measure	Accuracy
iForest+KNN	0.688 \pm 0.08	0.688 \pm 0.08	0.697 \pm 0.04	0.687 \pm 0.05	0.623 \pm 0.03	0.679 \pm 0.06	0.690 \pm 0.02	0.709 \pm 0.02	0.673 \pm 0.08	0.710 \pm 0.12
ODIN	0.726 \pm 0.03	0.745 \pm 0.02	0.752 \pm 0.05	0.787 \pm 0.05	0.744 \pm 0.04	0.729 \pm 0.03	0.776 \pm 0.02	0.783 \pm 0.01	0.790 \pm 0.04	0.806 \pm 0.06
mCANDIES	0.811 \pm 0.04	0.812 \pm 0.05	0.682 \pm 0.05	0.690 \pm 0.04	0.621 \pm 0.06	0.643 \pm 0.05	0.741 \pm 0.03	0.750 \pm 0.03	0.688 \pm 0.05	0.692 \pm 0.06
LACU-SVM	0.707 \pm 0.01	0.671 \pm 0.02	0.699 \pm 0.06	0.705 \pm 0.08	0.633 \pm 0.06	0.677 \pm 0.04	0.712 \pm 0.05	0.771 \pm 0.03	0.721 \pm 0.06	0.696 \pm 0.07
ECSMiner	0.706 \pm 0.05	0.732 \pm 0.07	0.714 \pm 0.06	0.737 \pm 0.09	0.700 \pm 0.02	0.746 \pm 0.03	0.665 \pm 0.04	0.688 \pm 0.06	0.752 \pm 0.05	0.723 \pm 0.02
CLAM	0.750 \pm 0.04	0.728 \pm 0.05	0.738 \pm 0.11	0.761 \pm 0.05	0.751 \pm 0.06	0.755 \pm 0.08	0.670 \pm 0.03	0.717 \pm 0.03	0.745 \pm 0.04	0.730 \pm 0.04
SENCForest	0.767 \pm 0.05	0.758 \pm 0.06	0.747 \pm 0.04	0.763 \pm 0.03	0.779 \pm 0.02	0.781 \pm 0.06	0.762 \pm 0.06	0.782 \pm 0.04	0.711 \pm 0.03	0.732 \pm 0.02
SENNE	0.802 \pm 0.03	0.810 \pm 0.02	0.807 \pm 0.04	0.789 \pm 0.04	0.794 \pm 0.06	0.783 \pm 0.05	0.811 \pm 0.02	0.808\pm 0.03	0.825 \pm 0.02	0.830 \pm 0.01
CPL	0.761 \pm 0.05	0.759 \pm 0.07	0.747 \pm 0.06	0.755 \pm 0.06	0.766 \pm 0.08	0.750 \pm 0.07	0.755 \pm 0.04	0.792 \pm 0.06	0.752 \pm 0.09	0.756 \pm 0.10
DNN-Based	0.734 \pm 0.07	0.739 \pm 0.06	0.745 \pm 0.07	0.767 \pm 0.09	0.746 \pm 0.10	0.737 \pm 0.06	0.754 \pm 0.03	0.791 \pm 0.05	0.734 \pm 0.07	0.740 \pm 0.04
LC-INC	0.858\pm 0.04	0.855\pm 0.03	0.828\pm 0.04	0.815\pm 0.04	0.807\pm 0.05	0.802\pm 0.02	0.818\pm 0.02	0.809\pm 0.03	0.855\pm 0.05	0.848\pm 0.03

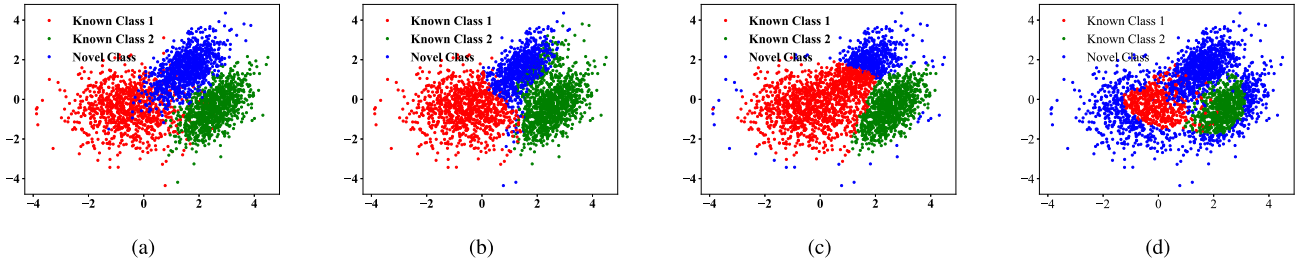


Fig. 5. Prediction of different methods on the synthetic 2-D dataset. Red and green dots are items of known classes and blue dots are items of novel class. (a) Ground truth. (b) Prediction of LC-INC. (c) Prediction of SENNE. (d) Prediction of iForest.

reported in Table I. Fig. 6 top line shows the accuracy curves with the timestamp of LC-INC and other compared methods.

Table I reveals that for all datasets, LC-INC consistently achieves significantly superior performance compared to other methods. LACU-SVM requires additional unlabeled instances in training and every model update, and ECSMiner needs all the true labels to train a new classifier for the ensemble. They still perform worse than LC-INC in both measures. The performance of iForest is dissatisfactory on the dataset with high dimensions because it is tree-based, which only uses a few dimensions for tree building. Similarly, mCANDIES has trouble modeling high-dimensional data. CLAM performs weakly for the imperfect detection and classifier. DNN-based framework only uses the prediction entropy and pays no attention to the structure information, which results in its inferior performance. The comparison between the DNN-based method and LC-INC can also be viewed as an ablation study where only entropy metric is used for novelty detection, and neglecting probability. The gap between them shows the superiority of jointly considering probability and entropy as the novelty score. SENNE uses an extra buffer to store prototypical samples, but our LC-INC still outperforms it in all tasks.

We also employ Friedman test [53] to statistically analyze the comparison over different datasets. The Friedman statistics F_F over accuracy and F-measure are 9.94 and 6.56, respectively. The critical value across 48 benchmark cases (6 datasets \times 8 compared methods in Fig. 6) at 0.05 significance level is 2.2852. As a result, the hypothesis of “equal” performance among compared methods is *clearly rejected* as the F_F statistics is greater than the critical value.

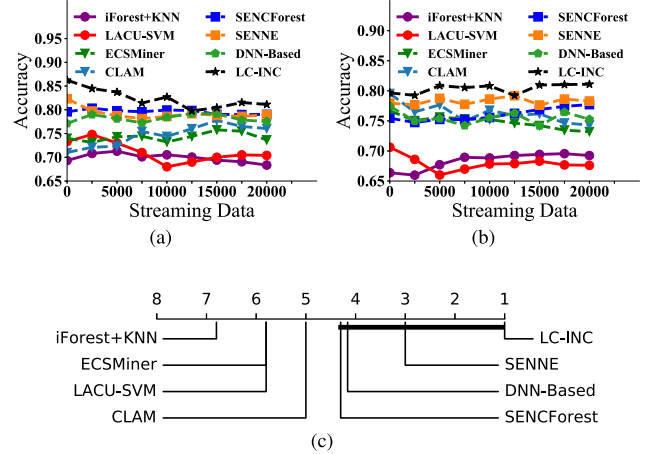


Fig. 6. Top: Accuracy curves of LC-INC and other compared methods. X-axis stands for stream timestamp and Y-axis stands for accuracy. (a) Accuracy of MNIST. (b) Accuracy of Fashion-MNIST. Bottom: Comparison of LC-INC (control algorithm) against other methods with Bonferroni–Dunn test. Approaches not connected with LC-INC in the CD diagram are considered to have significantly different performance from the control algorithm (CD = 3.804 at 0.05 significance level).

Consequently, we employ Bonferroni–Dunn test [53] as the posthoc test to figure out the relative performance. We treat LC-INC as the control approach and calibrate the critical difference (CD) between the average rank over all datasets of LC-INC and other compared methods. If the difference of average rank is greater than one CD (CD = 3.804 at 0.05 significance level), the performance between LC-INC and the corresponding method is significantly different. We report the CD diagram [53] of F-measure in Fig. 6(c) with LC-INC

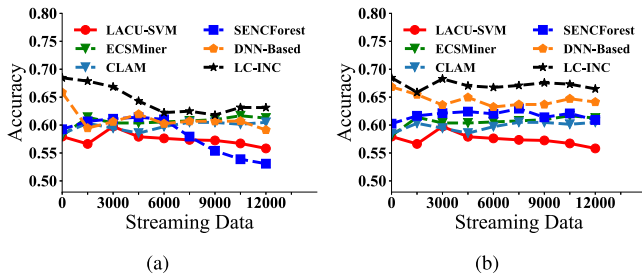


Fig. 7. Accuracy curve of YSneaker dataset under different conditions. True labels provide a more precise estimation of new class prototypes and facilitate the learning process of LC-INC. (a) Update **without** true label. (b) Update **with** true label.

being the control approach. We add a thick line to interconnect the control approach and another compared method if their difference in average rank is within one CD. We can infer from Fig. 6(c) that no compared method has significantly outperformed LC-INC in terms of F-measure. Additionally, LC-INC achieves the lowest average rank, which indicates its strong performance. The results indicate that LC-INC achieves superior performance than iForest, ECSMiner, LACU-SVM, and CLAM, and comparable performance to SENNE, DNN-Based, and SENCForest.

D. Stream Sneaker Identification

In this section, we conduct experiments in a real-world image classification data stream, i.e., YSneaker. Furthermore, two types of setting defined in Definition 1, that is, whether labels are available when deploying in the data stream are both studied. The more challenging setting is we cannot get the real labels and can only handle instances themselves for updating in the stream, while the easier one is that we can get instance labels at some time intervals. We set the model can get labels *when the buffer is full* for the easier condition. There are six brands of shoes in the YSneaker dataset, and we treat two of them as known classes at the initial stage and simulate others to appear in the later stage sequentially.

Fig. 7 shows the prediction results under different settings, which reveals that LC-INC can get better performance under both circumstances. Besides, we can compare the accuracy curves between Fig. 7(a) and (b). We can see that the accuracy decays along the time horizon in Fig. 7(a). Such decay mainly comes from two aspects. First, since the model learns to classify more classes, it has to predict among more classes with stream evolving. Second, since the model updates without true labels, the “wrongly detected instances” would influence the estimation of new class prototypes. Such influence accumulates, and the performance decays with stream evolving. However, in Fig. 7(b), accurate labels of the buffered instances can be fetched, and we can get a more precise estimation of the novel class center. This indicates the model can benefit from extra supervision information and reduce the performance decay of error accumulation.

Furthermore, we show some novel class detection examples about YSneaker in Fig. 8(a). We can see for novel instances from Converse and Puma, our LC-INC can consistently predict novel labels, while SENCForest failed in most detection tasks.

Additionally, we provide the t-SNE [54] visualization of input space and the learned embedding space in Fig. 8(b) and (c). In Fig. 8(b), we can see that it is hard to differentiate known from the unknown with the input features. However, with our learned embedding module, the known classes are clearly separated, and we can easily pick out the novel class instances, as shown in Fig. 8(c). T-SNE results also verify the feasibility of representing known classes with the class center.

E. Detecting and Learning With Multiple Novel Classes

A real-world scenario may simultaneously face the emergence of multiple novel classes. We design MNIST-multi and YSneaker-multi datasets to simulate this situation and present an example stream in Fig. 4(b). In the implementation, we simulate two novel classes to emerge meanwhile in one period for MNIST-multi, forming a data stream of 3 periods with 12000 instances. For YSneaker-multi, we set two classes as known classes, and make the rest four classes emerge simultaneously in the first period to synthesis a more complex situation.

As we stated in Section IV-B, there are two ways to handle multiple novel classes. We can either treat the novel classes in one period as a *meta-novel class*, or utilize the manual labels to differentiate them. In the former strategy, we label the meta-novel class as one united class at the later stage [14], and the other details are the same as the single novel class scenario. In the latter setting, the ground truth labels of buffered instances can be fetched before model updating. We conduct experiments under both settings, and report the performance in Fig. 9.

Fig. 9(a) reports the accuracy and F1 for MNIST-multi when treating multiple novel classes as a meta-novel class, and our proposed LC-INC consistently outperforms compared methods by a substantial margin. Furthermore, we can compare the performance with the experiments with the single novel-class scenario. We notice that most methods, including LC-INC suffer a performance decay. Such phenomena are reasonable because the meta-novel class forces the model to treat multiple different classes into one, and the estimation of class centers is thus influenced. Despite the decay in every method, LC-INC can still perform competitively. Fig. 9(b) reports the accuracy and F1 for YSneaker-multi when manual labels are available at model updating. The performance is consistent with the single novel-class scenario, and LC-INC performs best. The results also validate the transferable structure comparison module helps model updating with insufficient instances.

Ideally, the model should automatically assign novel instances to its true label instead of the meta-novel class. A feasible way to improve the current model is to combine it with clustering algorithms [34] before model updating. Extra supervision must be introduced to match cluster index to ground-truth labels such as one-shot labeled instances [55]. This is interesting future work, and we would explore it in the future.

F. Unknown Detection

Apart from end-to-end learning results, we also conduct experiments on unknown detection as the ablation study,

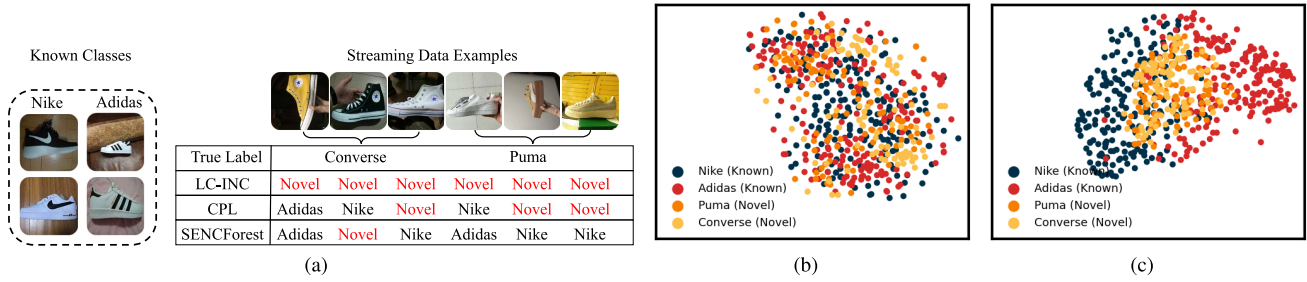


Fig. 8. Classification and novel class detection examples on the YSneaker dataset. In this trail, Nike and Adidas are known classes, Converse and Puma are novel classes emerging in the stream. We show the t-SNE visualization of the original feature space and the learned embedding space in (b) and (c). (a) Image examples of YSneaker. (b) Original feature space. (c) Embedding space of LC-INC.

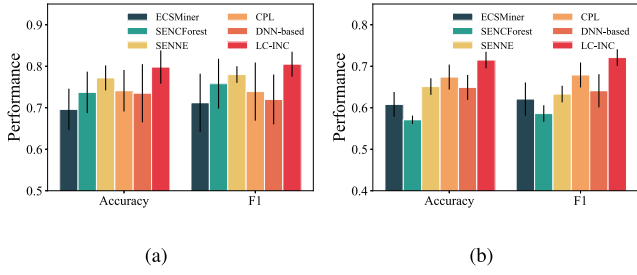


Fig. 9. Performance comparison when multiple novel classes emerge simultaneously. We can either treat them as a meta-novel class [in (a)] or require manual labeling [in (b)] to handle this situation. (a) MNIST-multi, without labeling. (b) YSneaker-multi, with labeling.

i.e., 2-class classification task on “known” versus “unknown.” We follow the setting in [28] and conduct experiments with MNIST. Several classes are sampled from a multiclass dataset as known and the others are viewed as unknown. We use the commonly used metric i.e., average area under the ROC curve (a.k.a AUC) to evaluate the detection performance. Since real-world scenarios are complex, where the ratio of seen and unseen differs in diverse tasks, we use openness [27], [56] to represent the complexity of the detection task¹

$$\text{Openness} = 1 - \sqrt{\frac{k}{K}}. \quad (8)$$

The openness becomes higher with more unknown classes, which indicates the dataset is more “open.” Since there are 10 classes in MNIST, we vary the number of known classes by {2, 4, 6, 8}, and the corresponding openness is between 10.55% and 55.27%. We compare to the traditional anomaly detection method iForest and the runner-up method in end-to-end experiments, i.e., SENNE. We also compare to DNN-based as the baseline of LC-INC, which predicts the novelty score via predicted information entropy. The results are shown in Fig. 10. We can infer that the performance of iForest and SENNE rise as the openness becomes larger, i.e., the detection task becomes harder with more known classes, and these methods hardly handle more classes. For comparison, DNN-based and LC-INC work robustly with the change of openness. Additionally, LC-INC achieves the best AUC performance among all compared methods. The ablation results indicate that LC-INC can work robustly with unknown detection tasks.

¹There are two similar definitions of openness, and we follow [56].

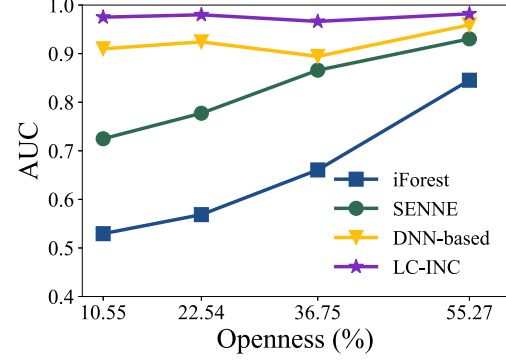


Fig. 10. Ablation study of unknown detection task on MNIST. X-axis stands for openness and Y-axis stands for the AUC measure.

G. Sensitivity About Parameters

Lastly, we study the influence of two parameters in LC-INC, including the tradeoff parameter λ and the buffer size s . The initial conditions are the same as the former experiment, and we test performance accuracy with different λ and s on the MNIST dataset. We fix the other parameters when testing one parameter. In the implementation, the range of tradeoff parameter $\lambda \geq 0$. $\lambda = 0$ indicates the model only pays attention to the entropy metric while ignoring the effect of probability, and a larger λ indicates the model will quickly increase the attention to probability metric. We tune λ in {0.05, 0.1, ..., 0.5}. For buffer size s , the model with little buffer size will update more frequently with the buffered instances. Furthermore, the estimation of the new class prototype will become more precise with more detected novel instances at a time, and consume more memory for instance saving. Considering the data size of each period is around 4000, we tune the buffer size in {100, 150, ..., 500} to make sure the model will update several times in every single period.

The results of these parameters are shown in Fig. 11. We can infer that LC-INC is robust over these two parameters. Results also guide the setup of parameters. In other datasets, we also set λ to 0.4 and s to 200, since they lead to the best performance for the current dataset. Additionally, we find the selection of best λ is consistent with our hypothesis that entropy plays an essential part in the early stages, and probability gradually gains the discriminability as model updating.

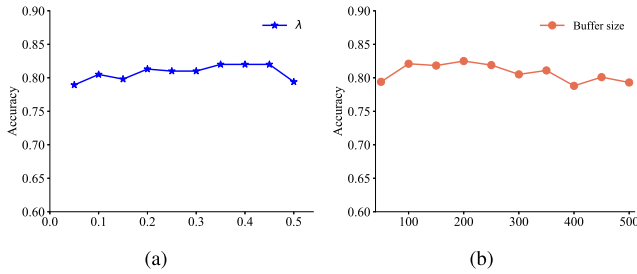


Fig. 11. Influence of λ and buffer size on MNIST dataset. X-axis stands for parameter value and Y-axis stands for Accuracy. (a) Influence of λ . (b) Influence of buffer size.

VI. DISCUSSION

Novelty detection, a.k.a. anomaly detection [18], [9], [57]–[62] refers to the process of detecting data instances that significantly deviate from the majority of data instances. We combine the novelty detection process with classification and model updating in our setting, enabling the model to detect and learn new classes in the data stream. However, related problems may cooccur in real-world data streams, e.g., incremental learning [2], concept drift detection [63], and noise data [64]. Specifically, incremental learning aims to adapt to new class data without forgetting existing knowledge, which does not refer to the detection process of novelties. Concept drift [65] refers to an online supervised learning scenario when the relation between the input data and the target variable changes over time, and concept drift detection aims to detect such change. In our setting, $p(y)$ changes with stream evolving, which is a kind of concept drift. However, concept drift detection methods pay most attention to the detection of “real concept drift” [65], i.e., the change of $p(y|x)$. We provide how to handle these problems in the following section.

To be specific, if concept drift $p(y|x)$ changes in our data stream, and our model needs to maintain classification performance to avoid destruction. We need to additionally maintain a change detection module like [66] with the current model. When $p(y|x)$ changes, the model would mistakenly treat known ones as a novel class, and the buffer would be filled with instances from known classes. Additionally, we need to move known instances to another buffer (drift memory bank) when labels are revealed to further deploy drift detection. We can now utilize concept drift detection algorithms on the market [34] to detect the drift. If concept drift is detected, the estimation of former known classes has drifted and should be corrected. We now utilize instances in the drift memory bank to recalculate known class prototypes of the current concept. If catastrophic forgetting [67] emerges, the classification performance of former known classes will be damaged by incorporated new class knowledge. Correspondingly, we need to sample exemplars from each class to form an exemplar set, and these exemplars will be utilized to compensate for the knowledge forgetting in incremental learning. In detail, the saved exemplars can be combined with new class instances as the training set, i.e., replay old class instances when learning new classes [68] to overcome forgetting. Additionally, knowledge distillation [69] can also be adopted to map the old

model and updated model and restrict model forgetting [70]. With the saved exemplars, knowledge distillation forces the updated model to have the same discriminability as old ones and preserve former knowledge. Lastly, if noise data exists in the data stream, the model will correspondingly detect noisy instances as novelty in the buffer. A consensus reached in [14] and [15] says that novel classes place further from normal regions than noisy instances, and we follow this assumption. We need feedback to remind the model of noisy data, e.g., manual labels when the buffer is full. If the model faces noisy data, the ratio of known instances detected in the buffer will rise. We can either raise the threshold to separate noisy instances from novelty, or adopt clustering algorithms [34] in the buffer. After that, instance clusters beside known class centers will be treated as noisy data and not used for model updating.

VII. CONCLUSION

In many real-world incremental data scenarios, data arise with novel classes. To solve novel class detection and model extension in this problem, we propose a novel framework LC-INC, which can efficiently detect the new class and rapidly extend the model with limited exposure to the novel instances. LC-INC can detect novel class instances by adaptively combine the prediction information with structure information. Furthermore, it can learn to expand through a discriminative distance function over input and the task space. Experiments on various datasets validate the effectiveness of our proposed method. How to consider the concept drift and feature evolution in the incremental dynamic scenario are interesting future works.

APPENDIX

The appendix provides more empirical evaluation about LC-INC, including synthesis data with manifold distribution, details of bilinear pooling, parameter search for compared methods, and experiments with imbalanced data streams.

A. Synthesis Data With Manifold Distribution

To synthesis a more challenging scenario of 2-D data, we keep the other settings and modify the feature of three classes to be “banana-shaped,” as shown in Fig. 12(a). According to the ground truth, these classes have elongated distributions and increase novelty detection difficulty. We compare the detection results of LC-INC to the best comparison method SENCForest. The visualization results are now shown in Fig. 12(b) and (c). Although the detection of new classes suffers from confusing information from known class centers, the classification performance of known classes is almost steady. However, in SENCForest, the tree-building process mistakenly treats half of the known classes as novelty, and performs worse than LC-INC. We can infer that although our proposed LC-INC is not designed for nonconvex data, the embedding module can learn a suitable projection to the embedding space and facilitate detecting novel classes with nonconvex data. Additionally, since the embedding module is trainable and joinable, dimension reduction algorithms can be adopted as part of the embedding module. If we

TABLE II

WE USE GRID SEARCH WITH BOOTSTRAPPING, AND REPORT THE COMPETITIVE PARAMETER WHICH LEADS TO THE BEST PERFORMANCE

Methods	Parameter search range	Best parameter
iForest	$t \in \{50, 60, \dots, 200\}; \psi \in \{100, 110, \dots, 300\}$	$t = 100, \psi = 200$
ODIN	$\tau \in \{1, 10, \dots, 1e6\}; \epsilon \in \{0.001, 0.0011, \dots, 0.002\}$	$\tau = 1000, \epsilon = 0.0012$
LACU-SVM	$\lambda \in \{0.001, 0.01, \dots, 0.1\}; max_iter \in \{5, 10, \dots, 30\}; \eta \in \{0.1, 0.2, \dots, 2\}$	$\lambda = 0.1, max_iter = 15, \eta = 1.4$
ECSMiner	$M \in \{3, 4, \dots, 10\}; K \in \{1, 2, \dots, 10\}$	$M = 6, k = 5$
CLAM	$K \in \{1, 2, \dots, 10\}; q \in \{50, 60, \dots, 200\}$	$K = 5, q = 130$
SENCForest	$\psi \in \{100, 110, \dots, 300\}; z \in \{50, 60, \dots, 200\}; \rho \in \{1, 2, \dots, 10\}$	$\psi = 200, z = 100, \rho = 3$
SENNE	$\psi \in \{100, 110, \dots, 300\}; s \in \{50, 60, \dots, 200\}$	$\psi = 200, s = 130$
CPL	$K \in \{1, 2, \dots, 10\}; \lambda \in \{0.0001, 0.001, \dots, 0.1\}$	$K = 2, \lambda = 0.01$
DNN-based	$t \in \{0.1, 0.15, \dots, 1\}; \tau \in \{1, 10, \dots, 1e5\}$	$t = 0.65, \tau = 1e5$

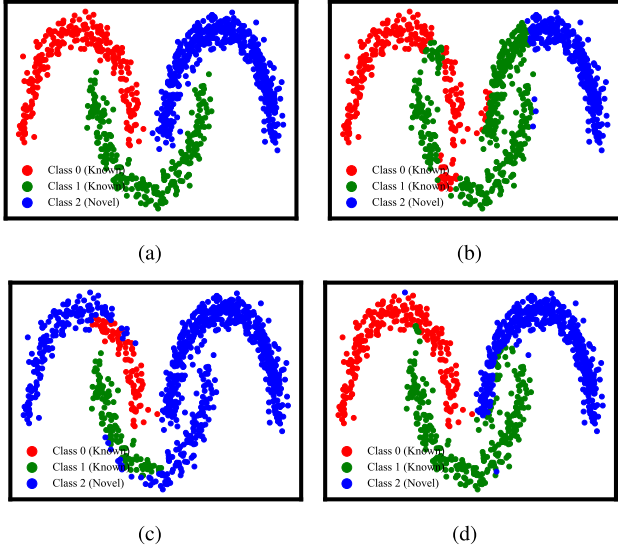


Fig. 12. Prediction of different compared methods on the synthetic 2-D banana-shaped dataset. (a) Ground Truth. (b) LC-INC. (c) SENCForest. (d) LC-INC[†].

are going to handle nonconvex data with LC-INC, we can combine an extra manifold dimension reduction module [54] and the embedding network as a unified $\mathcal{E}(\cdot)$. Correspondingly, we now extract class prototypes with the centroid of embedding space. The current embedding is well at capturing the instance-wise manifold information, denoted as LC-INC[†]. The results in Fig. 12(d) get the best result in the current task.

B. Details of Bilinear Pooling

Let $\{\mathbf{x}_s \in \mathbb{R}^p\}_{s=1}^m$ and $\{\mathbf{y}_t \in \mathbb{R}^q\}_{t=1}^n$ be two groups of features. In bilinear pooling, the feature fusion is achieved by

$$\begin{aligned} \mathbf{Z} &= \sum_{(s,t) \in \mathcal{S}} \mathbf{x}_s \mathbf{y}_t^\top \in \mathbb{R}^{p \times q} \\ \mathbf{z} &= \text{Vec}(\mathbf{Z}) = \sum_{(s,t) \in \mathcal{S}} \text{Vec}(\mathbf{x}_s \mathbf{y}_t^\top) \in \mathbb{R}^{pq} \end{aligned} \quad (9)$$

where \mathbf{z} is the fused representation, and \mathbf{Z} is its matrix form, and \mathcal{S} is the feature pair set of the two groups of features. The feature pair set \mathcal{S} has two common forms. In the image classification and action recognition tasks [42], [71], the two groups $\{\mathbf{x}_s \in \mathbb{R}^p\}_{s=1}^m$ and $\{\mathbf{y}_t \in \mathbb{R}^q\}_{t=1}^n$ have the same number of features, i.e., $m = n$, and \mathbf{x}_s and \mathbf{y}_s are extracted from the

same spatial or temporal location of the data. In the current case, bilinear pooling takes the form

$$\mathbf{z} = \sum_{s=1}^m \text{Vec}(\mathbf{x}_s \mathbf{y}_s^\top). \quad (10)$$

In the other form, \mathbf{x}_s contains all pairs of features from the two groups, which is commonly used in VQA tasks [72]. Under such setting, the fusion is represented as

$$\mathbf{z} = \sum_{s=1}^m \sum_{t=1}^n \text{Vec}(\mathbf{x}_s \mathbf{y}_t^\top). \quad (11)$$

Since the dimension of the final output $\mathbf{z} \in \mathbb{R}^{pq}$ can become overwhelming, some methods are proposed to produce a compact fusion feature. In this article, we use compact bilinear pooling [42] to reduce the pooling dimension.

C. Tuning Parameters for Compared Methods

We adopt grid search with bootstrapping to find the optimal parameter for compared methods. Following the dataset construction in [17], we draw three training sets (which contain instances from two known classes) and three testing data streams (which contain instances from both known and novel classes) to evaluate the parameter combination. We set the size of the sampled datasets as half of the original ones. Since the combination cost and running time for high-dimensional datasets is unacceptable, we perform a grid search over all possible combinations on the HAR dataset.² We report the range of parameter search and the adopted competitive hyperparameters of these compared methods in Table II.

D. Learning With Imbalanced Data Stream

We experiment with imbalanced data streams, where we simulate novel classes to emerge more frequently/infrequently in each period. We control the percentage of the novel class instances in each period, and denote it as ‘‘novel ratio,’’ i.e., stream with a high novel ratio contains more novel class instances. In contrast, a stream with a low novel ratio contains fewer novel class instances. We provide the comparison on the real-world dataset YSneaker by changing the novel ratio to 30% and 70%. The results of imbalanced data streams

²The running time of mCANDIES on high-dimensional input is high, and we use the default parameter instead.

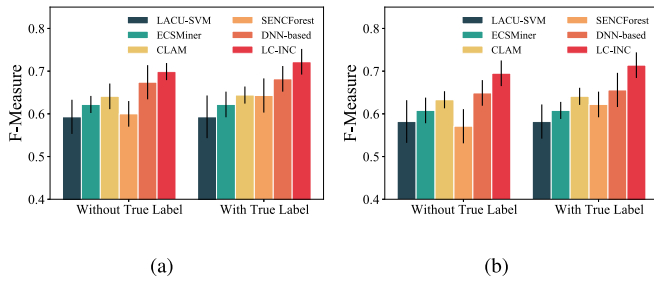


Fig. 13. F-measure of imbalanced YSneaker data stream. Novel ratio stands for the percentage of new class instances, and higher novel ratio indicates more novelty in the stream. (a) YSneaker, novel ratio = 30%. (b) YSneaker, novel ratio = 70%.

are shown in Fig. 13, where we show the F-measure (accuracy is not a good measure for imbalanced data) of compared methods. We further show the comparison of updating with/without true labels in the figure. We can infer from Fig. 13 that even for an imbalanced data stream, LC-INC can work robustly and outperform the other compared methods. Comparing Fig. 13(a) and (b), we found that most methods perform slightly better with fewer novel instances.

REFERENCES

- [1] J. Cao, Z. Wu, J. Wu, and H. Xiong, "SAIL: Summation-based incremental learning for information-theoretic text clustering," *IEEE Trans. Cybern.*, vol. 43, no. 2, pp. 570–584, Apr. 2013.
- [2] Y. Yang, D.-W. Zhou, D.-C. Zhan, H. Xiong, and Y. Jiang, "Adaptive deep models for incremental learning: Considering capacity scalability and sustainability," in *Proc. KDD*, 2019, pp. 74–82.
- [3] J. Liu, Y. Fu, J. Ming, Y. Ren, L. Sun, and H. Xiong, "Effective and real-time in-app activity analysis in encrypted internet traffic streams," in *Proc. KDD*, 2017, pp. 335–344.
- [4] Z.-H. Zhou, "Learnware: On the future of machine learning," *Frontiers Comput. Sci.*, vol. 10, no. 4, pp. 589–590, 2016.
- [5] X.-S. Wei, H.-J. Ye, X. Mu, J. Wu, C. Shen, and Z.-H. Zhou, "Multi-instance learning with emerging novel class," *IEEE Trans. Knowl. Data Eng.*, vol. 33, no. 5, pp. 2109–2120, May 2020.
- [6] H.-J. Ye, D.-C. Zhan, Y. Jiang, and Z.-H. Zhou, "Rectify heterogeneous models with semantic mapping," in *Proc. ICML*, 2018, pp. 5630–5639.
- [7] H.-J. Ye, D.-C. Zhan, Y. Jiang, and Z.-H. Zhou, "Heterogeneous few-shot model rectification with semantic mapping," *IEEE Trans. Pattern Anal. Mach. Intell.*, early access, May 14, 2020, doi: 10.1109/TPAMI.2020.2994749.
- [8] H.-J. Ye, S. Lu, and D.-C. Zhan, "Distilling cross-task knowledge via relationship matching," in *Proc. CVPR*, Jun. 2020, pp. 12396–12405.
- [9] 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.
- [10] Y. Yang, N. Zhu, Y. Wu, J. Cao, D. Zhan, and H. Xiong, "A semi-supervised attention model for identifying authentic sneakers," *Big Data Mining Anal.*, vol. 3, no. 1, pp. 29–40, Mar. 2020.
- [11] D. Zhang, Y. Liu, and L. Si, "Serendipitous learning: Learning beyond the predefined label space," in *Proc. KDD*, 2011, pp. 1343–1351.
- [12] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *Proc. ICDM*, 2008, pp. 413–422.
- [13] Y. Yang, J. Zhang, J. Carbonell, and C. Jin, "Topic-conditioned novelty detection," in *Proc. KDD*, 2002, pp. 688–693.
- [14] X. Mu, K. M. Ting, and Z. 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.
- [15] 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. ICDM*, 2019, pp. 970–975.
- [16] C. Gruhl and B. Sick, "Novelty detection with CANDIES: A holistic technique based on probabilistic models," *Int. J. Mach. Learn. Cybern.*, vol. 9, no. 6, pp. 927–945, Jun. 2018.
- [17] C. Gruhl, B. Sick, and S. Tomforde, "Novelty detection in continuously changing environments," *Future Gener. Comput. Syst.*, vol. 114, pp. 138–154, Jan. 2021.
- [18] G. Pang, C. Shen, L. Cao, and A. Hengel, "Deep learning for anomaly detection: A review," *ACM Comput. Surv.*, vol. 54, no. 2, pp. 1–38, 2020.
- [19] G. Pang, L. Cao, and C. Aggarwal, "Deep learning for anomaly detection: Challenges, methods, and opportunities," in *Proc. WSDM*, 2021, pp. 1127–1130.
- [20] D.-Y. Yeung and C. Chow, "Parzen-window network intrusion detectors," in *Proc. Object Recognit. Supported User Interact. Service Robots*, vol. 4, 2002, pp. 385–388.
- [21] L. Rettig, M. Khayati, P. Cudré-Mauroux, and M. Piórkowski, "Online anomaly detection over big data streams," in *Applied Data Science*, 2019, pp. 289–312.
- [22] S. Ahmad, A. Lavin, S. Purdy, and Z. Agha, "Unsupervised real-time anomaly detection for streaming data," *Neurocomputing*, vol. 262, pp. 134–147, Nov. 2017.
- [23] H. Larochelle, D. Erhan, and Y. Bengio, "Zero-data learning of new tasks," in *Proc. AAAI*, 2008, pp. 1–6.
- [24] M. Palatucci, D. Pomerleau, G. E. Hinton, and T. M. Mitchell, "Zero-shot learning with semantic output codes," in *Proc. NIPS*, 2009, pp. 1410–1418.
- [25] N. Kumar, A. C. Berg, P. N. Belhumeur, and S. K. Nayar, "Attribute and simile classifiers for face verification," in *Proc. ICCV*, 2009, pp. 365–372.
- [26] C. H. Lampert, H. Nickisch, and S. Harmeling, "Attribute-based classification for zero-shot visual object categorization," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 3, pp. 453–465, Mar. 2014.
- [27] C. Geng, S.-J. Huang, and S. Chen, "Recent advances in open set recognition: A survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, early access, Mar. 18, 2020, doi: 10.1109/TPAMI.2020.2981604.
- [28] D.-W. Zhou, H.-J. Ye, and D.-C. Zhan, "Learning placeholders for open-set recognition," in *Proc. CVPR*, 2021, pp. 4401–4410.
- [29] A. Bendale and T. E. Boult, "Towards open set deep networks," in *Proc. CVPR*, Jun. 2016, pp. 1563–1572.
- [30] H.-M. Yang, X.-Y. Zhang, F. Yin, and C.-L. Liu, "Robust classification with convolutional prototype learning," in *Proc. CVPR*, Jun. 2018, pp. 3474–3482.
- [31] D.-W. Zhou, Y. Yang, and D.-C. Zhan, "Detecting sequentially novel classes with stable generalization ability," in *Proc. PAKDD*, 2021, pp. 371–382.
- [32] E. J. Spinoso, A. P. de Leon F. de Carvalho, and J. A. Gama, "OLINDDA: A cluster-based approach for detecting novelty and concept drift in data streams," in *Proc. ACM Symp. Appl. Comput.*, 2007, pp. 448–452.
- [33] T. Al-Khateeb, M. M. Masud, L. Khan, C. Aggarwal, J. Han, and A. B. Thuraisingham, "Stream classification with recurring and novel class detection using class-based ensemble," in *Proc. ICDM*, 2012, pp. 31–40.
- [34] E. R. de Faria, A. C. Ponce de Leon Ferreira Carvalho, and J. Gama, "MINAS: Multiclass learning algorithm for novelty detection in data streams," *Data Mining Knowl. Discovery*, vol. 30, no. 3, pp. 640–680, May 2016.
- [35] Q. Da, Y. Yu, and Z.-H. Zhou, "Learning with augmented class by exploiting unlabeled data," in *Proc. AAAI*, 2014, pp. 1760–1766.
- [36] Z. Wang, L. Liu, and D. Tao, "Deep streaming label learning," in *Proc. ICML*, 2020, pp. 9963–9972.
- [37] Y.-J. Zhang, P. Zhao, and Z.-H. Zhou, "Exploratory machine learning with unknown unknowns," in *Proc. AAAI*, 2021, pp. 10999–11006.
- [38] B. Babcock, M. Datar, and R. Motwani, "Sampling from a moving window over streaming data," in *Proc. SODA*, 2002, pp. 633–634.
- [39] J. B. Gomes, M. M. Gaber, P. A. C. Sousa, and E. Menasalvas, "Mining recurring concepts in a dynamic feature space," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, no. 1, pp. 95–110, Jan. 2014.
- [40] Y. He, B. Wu, D. Wu, E. Beyazit, S. Chen, and X. Wu, "Toward mining capricious data streams: A generative approach," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 3, pp. 1228–1240, Mar. 2021.
- [41] L. Rutkowski, M. Jaworski, L. Pietruczuk, and P. Duda, "A new method for data stream mining based on the misclassification error," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 5, pp. 1048–1059, May 2015.
- [42] Y. Gao, O. Beijbom, N. Zhang, and T. Darrell, "Compact bilinear pooling," in *Proc. CVPR*, Jun. 2016, pp. 317–326.
- [43] K. Lee, K. Lee, K. Min, Y. Zhang, J. Shin, and H. Lee, "Hierarchical novelty detection for visual object recognition," in *Proc. CVPR*, Jun. 2018, pp. 1034–1042.

- [44] M. Tavallae, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," in *Proc. IEEE Symp. Comput. Intell. Secur. Defense Appl.*, Jul. 2009, pp. 1–6.
- [45] Y. LeCun, C. Cortes, and C. Burges, "MNIST handwritten digit database," *AT&T Labs*, vol. 2, p. 18, 2010. [Online]. Available: <http://yann.lecun.com/exdb/mnist>
- [46] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms," 2017, *arXiv:1708.07747*. [Online]. Available: <http://arxiv.org/abs/1708.07747>
- [47] T.-S. Chua, J. Tang, R. Hong, H. Li, Z. Luo, and Y. Zheng, "NUS-WIDE: A real-world web image database from National University of Singapore," in *Proc. CIVR*, Santorini Island, Greece, 2009, pp. 1–6.
- [48] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. CVPR*, Jun. 2016, pp. 770–778.
- [49] S. Liang, Y. Li, and R. Srikant, "Enhancing the reliability of out-of-distribution image detection in neural networks," in *Proc. ICLR*, 2018, pp. 1–15.
- [50] D. Hendrycks and K. Gimpel, "A baseline for detecting misclassified and out-of-distribution examples in neural networks," in *Proc. ICLR*, 2016, pp. 1–12.
- [51] A. Paszke *et al.*, "PyTorch: An imperative style, high-performance deep learning library," 2019, *arXiv:1912.01703*. [Online]. Available: <http://arxiv.org/abs/1912.01703>
- [52] Z. Wang, Z. Kong, S. Changra, H. Tao, and L. Khan, "Robust high dimensional stream classification with novel class detection," in *Proc. ICDE*, Apr. 2019, pp. 1418–1429.
- [53] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *J. Mach. Learn. Res.*, vol. 7, pp. 1–30, Jan. 2006.
- [54] L. V. D. Maaten and G. Hinton, "Visualizing data using t-SNE," *J. Mach. Learn. Res.*, vol. 9, no. 11, pp. 2579–2605, 2008.
- [55] J. N. Kundu, R. M. Venkatesh, N. Venkat, A. Revanur, and R. V. Babu, "Class-incremental domain adaptation," in *Proc. ECCV*, 2020, pp. 53–69.
- [56] P. Perera *et al.*, "Generative-discriminative feature representations for open-set recognition," in *Proc. CVPR*, Jun. 2020, pp. 11814–11823.
- [57] Y. Zhao, Z. Nasrullah, and Z. Li, "PyOD: A Python toolbox for scalable outlier detection," *J. Mach. Learn. Res.*, vol. 20, no. 96, pp. 1–7, Jan. 2019.
- [58] G. Pang, K. M. Ting, D. Albrecht, and H. Jin, "ZERO++: Harnessing the power of zero appearances to detect anomalies in large-scale data sets," *J. Artif. Intell. Res.*, vol. 57, pp. 593–620, Dec. 2016.
- [59] G. Pang, H. Xu, L. Cao, and W. Zhao, "Selective value coupling learning for detecting outliers in high-dimensional categorical data," in *Proc. CIKM*, 2017, pp. 807–816.
- [60] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, pp. 1–58, Jul. 2009.
- [61] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "On detecting clustered anomalies using SCiForest," in *Proc. ECML-PKDD*, 2010, pp. 274–290.
- [62] Y. Zhu, K. M. Ting, and Z.-H. Zhou, "Discover multiple novel labels in multi-instance multi-label learning," in *Proc. AAAI*, vol. 31, 2017, pp. 1–7.
- [63] A. Haque, L. Khan, M. Baron, B. Thuraisingham, and C. Aggarwal, "Efficient handling of concept drift and concept evolution over stream data," in *Proc. ICDE*, 2016, pp. 481–492.
- [64] F. Cao, M. Estert, W. Qian, and A. Zhou, "Density-based clustering over an evolving data stream with noise," in *Proc. SIAM*, 2006, pp. 328–339.
- [65] J. Gama, I. Žliobaite, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Comput. Surv.*, vol. 46, no. 4, pp. 1–37, 2014.
- [66] A. Haque, L. Khan, and M. Baron, "SAND: Semi-supervised adaptive novel class detection and classification over data stream," in *Proc. AAAI*, 2016, pp. 1–7.
- [67] K. James *et al.*, "Overcoming catastrophic forgetting in neural networks," *Proc. Nat. Acad. Sci. USA*, vol. 114, no. 13, pp. 3521–3526, Mar. 2017.
- [68] H. Shin, J. K. Lee, J. Kim, and J. Kim, "Continual learning with deep generative replay," in *Proc. NIPS*, 2017, pp. 2990–2999.
- [69] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015, *arXiv:1503.02531*. [Online]. Available: <http://arxiv.org/abs/1503.02531>
- [70] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, "iCaRL: Incremental classifier and representation learning," in *Proc. CVPR*, Jul. 2017, pp. 2001–2010.
- [71] Y. Zhang, S. Tang, K. Muandet, C. Jarvers, and H. Neumann, "Local temporal bilinear pooling for fine-grained action parsing," in *Proc. CVPR*, Jun. 2019, pp. 12005–12015.
- [72] J.-H. Kim, J. Jun, and B.-T. Zhang, "Bilinear attention networks," in *Proc. NIPS*, 2018, pp. 1–13.



Da-Wei Zhou (Graduate Student Member, IEEE) is currently pursuing the Ph.D. degree with the State Key Laboratory for Novel Software Technology, Department of Computer Science and Technology, Nanjing University, Nanjing, China.

He is currently working on incremental learning. His research interests lie primarily in machine learning and data mining.



Yang Yang received the Ph.D. degree in computer science from Nanjing University, Nanjing, China, in 2019.

In 2019, he became a Faculty Member with Nanjing University of Science and Technology, Nanjing. He is currently a Professor with the Department of Computer Science and Technology, Nanjing University. He has authored or coauthored more than ten articles in leading international journal/conferences. His research interests lie primarily in machine learning and data mining, including heterogeneous learning, model reuse, and incremental mining.

Dr. Yang serves as a program and general chairs (PC) in leading conferences such as IJCAI, AAAI, ICML, NIPS, and so on.



De-Chuan Zhan received the Ph.D. degree in computer science from Nanjing University, Nanjing, China, in 2010.

In 2010, he became a Faculty Member with the Department of Computer Science and Technology, Nanjing University. He is currently a Professor with the Department of Computer Science and Technology, Nanjing University. He has authored or coauthored more than 20 articles in leading international journal/conferences. His research interests are mainly in machine learning, data mining and mobile intelligence.

Dr. Zhan serves as an Editorial Board Member of IDA and IJAPR, and serves as a SPC/PC in leading conferences such as IJCAI, AAAI, ICML, NIPS, and so on.