# Learning High-Dimensional Evolving Data Streams With Limited Labels

Salah Ud Din, Jay Kumar, Junming Shao, Cobbinah Bernard Mawuli, and Waldiodio David Ndiaye

***Abstract*—In the context of streaming data, learning algorithms often need to confront several unique challenges, such as concept drift, label scarcity, and high dimensionality. Several concept drift-aware data stream learning algorithms have been proposed to tackle these issues over the past decades. However, most existing algorithms utilize a supervised learning framework and require all true class labels to update their models. Unfortunately, in the streaming environment, requiring all labels is unfeasible and not realistic in many real-world applications. Therefore, learning data streams with minimal labels is a more practical scenario. Considering the problem of the curse of dimensionality and label scarcity, in this article, we present a new semisupervised learning technique for streaming data. To cure the curse of dimensionality, we employ a denoising autoencoder to transform the high-dimensional feature space into a reduced, compact, and more informative feature representation. Furthermore, we use a cluster-and-label technique to reduce the dependency on true class labels. We employ a synchronization-based dynamic clustering technique to summarize the streaming data into a set of dynamic microclusters that are further used for classification. In addition, we employ a disagreement-based learning method to cope with concept drift. Extensive experiments performed on many real-world datasets demonstrate the superior performance of the proposed method compared to several state-of-the-art methods.**

***Index Terms*—Concept drift, denoising autoencoder (DAE), evolving data streams, semisupervised learning (SSL), synchronization.**

Manuscript received July 12, 2020; revised October 29, 2020; accepted March 25, 2021. This work was supported in part by the Fundamental Research Funds for the Central Universities under Grant ZYGX2019Z014; in part by the National Natural Science Foundation of China under Grant 61976044 and Grant 52079026; in part by the Fok Ying-Tong Education Foundation for Young Teachers in the Higher Education Institutions of China under Grant 161062; and in part by the National Key Research and Development Program under Grant 2016YFB0502300. This article was recommended by Associate Editor N. Zhang. *(Corresponding author: Junming Shao.)*

Salah Ud Din is with the School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China, also with the Yangtze Delta Region Institute (Huzhou), University of Electronic Science and Technology of China, Huzhou 313001, China, and also with the Department of Computer Science, COMSATS University Islamabad, Islamabad 45550, Pakistan (e-mail: salahuddin@std.uestc.edu.cn).

Jay Kumar, Junming Shao, Cobbinah Bernard Mawuli, and Waldiodio David Ndiaye are with the School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China, and also with the Yangtze Delta Region Institute (Huzhou), University of Electronic Science and Technology of China, Huzhou 313001, China (e-mail: jay@std.uestc.edu.cn; junmshao@uestc.edu.cn; conninahben@std.uestc.edu.cn; waldavidndiaye00@yahoo.fr).

This article has supplementary material provided by the authors and color versions of one or more figures available at https://doi.org/10.1109/TCYB.2021.3070420.

Digital Object Identifier 10.1109/TCYB.2021.3070420

## I. Introduction

With the ever-rising integration of technologies and the popularity of various real-world applications, more data are being generated in many diverse fields, such as sensor networks, supermarket transactions, and social networks. Such data generation is referred to as streaming data and contains valuable information that needs to be processed for real-time data analytics. To mine hidden patterns or knowledge from these streaming data, many data mining and machine-learning-based techniques [1]–[4] have been proposed and prove their usefulness in various domains.

*Challenge 1 (Concept Drift Adaptation):* The main challenge for data stream learning algorithms is a phenomenon called the concept drift [5]. The concept drift refers to a situation where a change in the distribution of data may occur over time. When the concept drift occurs, the classifier's performance trained on the historical data degrades significantly and gradually becomes obsolete, that is, it no longer accurately represents the new data distribution. It is, therefore, necessary to refine/update the classifier with the most recent data. Various state-of-the-art methods have been proposed to handle the concept drift problem [6]–[8]. However, most existing approaches utilize a supervised learning paradigm and require a complete set of actual class labels to update their models.

*Challenge 2 (High Data Dimensionality):* Curse of dimensionality is a major issue and severely affects the effectiveness of a learning algorithm [9]. The accuracy and performance of learning algorithms degrade as the number of data attributes increases. In addition, space and processing time of learning algorithms is also elevated with the increase in dimensionality.

*Challenge 3 (Label Scarcity):* It is impossible to manually label all data in practical applications because the volume of streaming data is potentially unbound and arrives at high speed [2]. In this case, supervised learning algorithms are not feasible in these applications where obtaining the labels is expensive, time consuming, or impractical. Therefore, to eliminate dependence on complete labeled information, semisupervised learning (SSL) algorithms can be used. SSL algorithms are designed to use abundant unlabeled data with label information to boost the classification performance [10]. Several SSL algorithms are available, such as low-density separation, graph-based models, co-training, self-training, and generative models [11]. However, these methods cannot be applied directly to streaming data due to several challenges discussed above.

Several approaches [12]–[17] have recently been proposed that focus on reducing label dependency and provide SSL for the concept drifting data stream. However, there are potential problems in these existing works. For example, the techniques proposed in [12]–[14], [18], and [19] utilize clustering to assign labels to the unlabeled data. The main drawback of these methods is that the number of clusters must be predefined. Determining the optimal value for the number of clusters is a difficult task. Thus, they cannot capture the hidden structure of the underlying data distribution and cannot tackle the noisy or evolving property of the streaming data, causing the degradation of their performance.

The approaches presented in [17], [20], and [21] use online clustering algorithms, such as CluStream and DenStream, to summarize streaming data by maintaining a fixed set of online microclusters and provide a semisupervised data stream classification. There are two major problems with these techniques. First, they assume the immediate availability of label data, which is unrealistic in many applications. Second, maintained microclusters cannot capture the hidden local cluster structure for incoming data with non-Gaussian distributions. Another potential problem with the clustering algorithm is that they tend to fail with high-dimensional data. Similarly, Sethi *et al.* [22] proposed a grid-density-based clustering [23] approach for classifying streaming data. However, since the grid size needed to be defined in advance in this method, determining the optimum value of the grid size is a difficult task. In addition, they are more suitable for low-dimensional data, as their complexity depends on the data dimension.

Furthermore, the techniques proposed in [15] and [24] are graph-based models and construct a graph from the most recent data to spread the label information to unlabeled data points. However, online graph construction (quality of graph) and maintenance is a major concern and seriously affects the classification performance [25]. Similarly, some other techniques proposed in [16], [20], and [26] are based on traditional co-training and self-training paradigms. These techniques use pseudo/predicted labels to refine or update their models. However, such a strategy can result in a high cumulative error if the unlabeled data points are wrongly classified.

In addition to SSL, the active learning paradigm is also used to reduce the need for labels in training data. In active learning, the few most uncertain data points are queried from the oracle or the expert to form training sets based on specific selection criteria. Several online active learning techniques for streaming data have been proposed that can work in a single-pass sample selection. A detailed discussion on issues and methods for online active learning can be found in [27].

To address these issues, in this article, we propose a new learning technique for high-dimensional data streams with limited supervision (labeled data). For this purpose, we combine several techniques to provide a reliable semisupervised data stream learning algorithm. The main contributions of this article are summarized as follows.

1) *Synchronization-Based Data Representation:* We employ a cluster-and-label strategy to overcome the problem of limited labeled data. As discussed above, the existing clustering-based algorithms cannot efficiently capture the hidden local cluster structure and produce low-quality clusters. In this work, we employ and extend the synchronization-based dynamic clustering algorithm to summarize the incoming streaming data by maintaining a set of dynamically synchronized microclusters. Synchronization provides intuitive and better data abstraction due to its ability to dynamically capture the underlying local data distributions. It can accurately capture regions with low/high density of data and thus support the low-density separation assumption of SSL. Also, it produces high-quality clusters and is robust to noisy streaming data.
2) *Unsupervised Deep Embedding:* We employ a denoising autoencoder (DAE) to extract useful and robust features to overcome the curse of dimensionality and to provide better data representation in reduced feature space. DAE is an unsupervised learning technique and has proven to be a very useful data representative learning strategy in many existing works.
3) *High Performance:* Combining several techniques such as DAE for feature extraction, synchronization-based dynamic cluster-and-label strategy, and effective online data maintenance, our algorithm allows capturing evolving concepts efficiently. It thus yields a high classification performance even with limited true class labels.

The remainder of this article is arranged as follows. A brief overview of the existing methods is presented in Section II. Section III explains the proposed algorithm in detail. Section IV provides empirical study, results, and discussion. Section V concludes this article with the final remarks.

## II. Related Work

Recently, SSL in data streams has gained significant attention in data stream mining research, and several techniques over recent years have been developed. In this section, we present an overview of existing methods for SSL in data streams from the literature. In addition, we also provide a brief summary of DAE-based algorithms.

### A. Semisupervised Data Stream Algorithms

In order to reduce the label dependency, different static SSL algorithms (such as clustering and graph based) are explored and modified to work with streaming data and handle concept drift. Here, we provide an overview of these approaches.

*Clustering-Based Algorithms:* The intuition behind using clustering for SSL is that similar data points within a cluster are likely to have the same class labels. Thus, labels of other data points in the cluster can be inferred with only a few labeled data points. Relying on cluster assumption, Masud *et al.* [12], and SPASC [13] present an ensemble of classifiers for SSL in data streams. Both methods use the expectation–maximization (EM) algorithm to create a new cluster-based classifier or update an existing one in the ensemble from a partially labeled data block. The work proposed in [14] creates its classification model by using an incremental decision tree. After building the decision tree, this method utilizes clustering to infer the labels of unlabeled data points in each tree's leaf node.

*Graph-Based Algorithms:* Graph-based algorithms are based on manifold assumption and assume that the instances on the same structure share a common label. For streaming data, the algorithms proposed in [15] and [24] maintain a graph on a sliding window and apply different strategies to update the graph with the recent data dynamically. For example, the proposed algorithm in [24] uses an online max-flow algorithm to calculate a min-cut on the current max-flow to classify the unlabeled instances. Similarly, Wagner *et al.* [15] proposed a temporal label propagation method for streaming data. In this method, a star-mesh transform procedure is applied to update the graph with new incoming data and utilize the harmonic solution technique to assign the labels to unlabeled data points.

*Clustering Combined With Graph-Based Methods:* In these approaches [18], [19], the data points for each new data block are divided into clusters. Each cluster is marked as labeled (if it contains label points) and unlabeled. Then, a graph is built of the centers of the clusters represented as vertices in the graph. Then, a label propagation algorithm is used to assign labels to unlabeled clusters. New clusters are created and added to the graph for each new data block, while some of the existing obsolete clusters are removed. Thus, all incoming clusters are summarized in a graph. The algorithm presented in [18] also trains a classifier (like decision tree) along with cluster groups and graph-based label propagation if the data block is labeled.

*Self-Training and Co-Training-Based Algorithms:* The work in [26] used the traditional SSL co-training technique for data stream classification. It extends the co-forest algorithm [28] and creates incremental random decision trees. First, random trees are trained on labeled training data. Afterward, each tree is incrementally updated with highly reliable data points selected by other ensembles of trees. Similarly, approaches proposed in [16] and [20] are based on a self-training paradigm. In these methods, first, a classifier is trained on a few labeled training data. Afterward, a subset of highly reliable (confidence) predicted class labels of unlabeled instances is selected, and the classifier is updated incrementally with these predicted labels. In addition to these, the work proposed in [29] and [30] is based on an active learning paradigm. Reference [29] processes the streaming data in fixed-size blocks. After classification, labels of the subset of data points in the block are queried from the oracle to form the training set. Afterward, the classifier is then rebuilt. In [30], an active learning algorithm is employed to choose few instances, and labels for these instances are queried from the experts. With these selected instance labels, a semisupervised method is used to monitor the performance of the classifier for potential change.

### *B. Denoising Autoencoder-Based Algorithms*

DAE has already been used in various domains (e.g., image processing and speech processing) to extract robust features (feature learning), which are particularly useful for the classification task. For example, the classification performance of SVMs can be improved when stacked DAEs [31] are used to extract the higher-level representation of input features. An unsupervised domain adaptation technique based on DAE is proposed in [32], where the main objective is to create a matching representation space for the attributes of the source and target sets. A classification technique to detect cancer nuclei in tissue images is proposed in [33]. Several DAE-based techniques [34], [35] are also proposed to improve the detection of anomalies and outliers in various applications. The work presented in [36] has shown that combining clustering and autoencoders significantly improves performance compared to traditional clustering algorithms.

## III. Proposed Algorithm

### *A. Notations and Symbols*

A data stream is an ordered sequence of data points ($\mathrm{DS} = x_1, x_2 \ldots x_n, \ldots$) which arrives continuously. We divide the data stream into a fixed-size length data block ($B_1, B_2, \ldots, B_n$), that is, all incoming data points are accumulated in chunks before providing them to the algorithm. Here, $B_i$ is the $i$th data block, and $B_n$ is the latest data block in the data stream. Each data block contains a fixed set of data points defined by block size $S$, that is, $B_i = \{x_1, x_2 \ldots x_S\}$.

### *B. Main Parts of the Proposed Algorithm*

The proposed algorithm has three main parts, namely: 1) DAE; 2) synchronization-based microclusters; and 3) model update. A detailed description of these parts is provided in the following sections.

*1) Denoising Autoencoders:* The primary purpose of the autoencoder is to perform a feature space transformation where the learned features have specific desired properties, such as desired analytical properties, higher sparsity, and, most important, lower dimensionality [37]. The learned model can then be used to transform new examples into a hidden feature space.

Autoencoders are typically neural networks that have two functions, namely: 1) the encoder and 2) the decoder [38]. The encoder denoted by $f$ transforms the original input features $x \in \mathcal{R}^d$ into a latent feature space $z \in \mathcal{R}^p$. The encoder function is given as follows:

$$f(x) = \theta(Wx + b) \tag{1}$$

where $W$ is a weight matrix, and $b$ is the bias. $\theta(s)$ is the sigmoid function and defined as $\theta(s) = (1 + \exp(s))^{-1}$. To include deeper features, many hidden layers can be added to the model. Let us consider $d$ as the size of the input vector $x$ and $p < d$ is the size of the deepest hidden layer. The decoder function denoted by $g$ transforms the latent feature space $z$ into $\acute{x}$ (reconstructed input feature). Here, the main purpose of the autoencoders is to minimize the reconstruction error. The decoder function is given as follows:

$$g(z) = \theta\left(W^T z + b\right). \tag{2}$$

The cost function for reconstruction loss can be computed using mean-square error (MSE)

$$\mathcal{L}_{\mathrm{mse}}(x, \acute{x}) = \left\| x - \acute{x} \right\|_2^2. \tag{3}$$

Similarly, other cost functions, such as cross-entropy, can also be used. The cost function is optimized using a gradient descent algorithm. The main idea behind DAE [39] is to learn a model that can generate robust features by reconstructing the input from a partially corrupted sample. To produce a corrupted version of the input $\tilde{x}$, a noise vector $\eta$ is sampled (using i.i.d.) from a noise distribution with mean zero and variance $\sigma^2$, that is, $\eta \sim \mathcal{N}(0, \sigma^2)$. The corrupted version of input $\tilde{x}$ is defined as $\tilde{x} = x + \eta$. DAE, therefore, intuitively performs two operations. First, it transforms or encodes the corrupted input $\tilde{x}$ into a hidden representation $z$. Second, it decodes the $z$ into $\acute{x}$ (reconstructed input). The goal is to minimize the reconstruction error from the original (uncorrupted) input $x$.

We train a DAE from the first few $D_{\text{init}}$ unlabeled data block by learning the weights $W$ and $b$. Both the encoder and decoder are implemented using deep, dense layers. The learned weights are then kept unchanged and used to transform (extract) the values of the features of all incoming instances during the entire stream. It is also important to note that DAE weights are computed only once during the initial training phase, and we do not update them during the rest of the streaming data. Updating DAE weights during the stream progress is a time-consuming process because such an update would require running the backpropagation process whenever new data are available. Here, we represent the transformed features of input sample $x \in \mathcal{R}^d$ as $z \in \mathcal{R}^p$.

*2) Synchronization-Based Dynamic Microclusters:* In the context of data streaming, it is impossible to store all historical data. Therefore, in this article, we introduce a synchronization-based clustering approach (such as SYNC [40]) for representing and summarizing online data. In synchronization-based clustering approaches, each data point is considered a phase oscillator that exhibits dynamic behavior over time. With time, the phase of a data point gradually aligns with its interacting similar neighboring data points. The local cluster structure controls the movements of these data points in a nonlinear fashion. After these interactions, all the data points of a cluster are synchronized and eventually converge in the same phase. The main feature of synchronization-based clustering algorithms is that the local data structure dynamically controls cluster formation. To simulate a dynamic clustering process, a synchronization-based clustering algorithm generally requires three definitions. The first is an interaction range parameter $\epsilon$ between data points. The second is the interaction model for clustering, and the third is the stopping criterion to terminate the clustering process. Details of these definitions are given as follows. In our method, we have adopted and extended SYNC [40], which is a synchronization-based clustering algorithm.

*Definition 1 ($\epsilon$-Range Neighborhood):* Given a dataset $\mathcal{D}$ of size $N$, $\epsilon \in \mathcal{R}$ and $a \in \mathcal{D}$, the $\epsilon$-range neighborhood of a data point $a$, denoted by $N_\epsilon(a)$, is defined as

$$N_\epsilon(a) = \{b \in \mathcal{D} | \text{dist}(a, b) \le \epsilon\}. \quad (4)$$

Here, dist*(a, b)* is the Euclidean distance between $a$ and $b$.

*Definition 2 (Interaction Model):* Let $a \in \mathcal{R}^d$ be a data point in dataset $\mathcal{D}$, and $a^i$ be the $i$th dimension of the data point $a$, respectively. With an $\epsilon$-range neighborhood interaction, the dynamics of each dimension $a^i$ of the data point $a$ is defined as

$$a^i(t+1) = a^i(t) + \frac{1}{|N_\epsilon(a)|} \sum_{\substack{b \in N_\epsilon(a) \\ \text{cond}(.)}} \sin(b^i(t) - a^i(t)). \quad (5)$$

Here, cond(.) is defined as follows. Two points $a$ and $b$ can interact according to a condition, that is, if one or both are unlabeled or if both have the same labels. This means that two points cannot interact if they have different class labels. sin(.) is the coupling function and is used in most synchronization-based algorithms. $a(t+1)$ is the updated phase value of the data point $a$ to $t = (0, \ldots, T)$ during the dynamic clustering process.

To characterize the level of synchronization between oscillators during the synchronization process, a cluster order parameter cr is defined to measure the coherence of the local oscillator population.

*Definition 3 (Cluster Order Parameter):* The cluster order parameter cr is used to terminate the dynamic clustering by investigating the degree of local synchronization, which is defined as

$$\text{cr}(t) = \frac{1}{N} \sum_{j=1}^{N} \frac{1}{|N_\epsilon(a_j)|} \sum_{b \in N_\epsilon(a_j)} e^{-||(b(t) - a_j(t)||}. \quad (6)$$

Here, $N$ is the total number of data points in $\mathcal{D}$. The process of dynamic clustering ends when cr converges (indicating a coherence of the local phase). In this article, we use the following condition to check the convergence:

$$|\text{cr}(t) - \text{cr}(t-1)| < 1e^{-6}. \quad (7)$$

**Algorithm 1:** Synchronization-Based Microclusters

```
Function SynMicroClusters($\mathcal{D}$)
    $MCs \leftarrow \phi$
    $t \leftarrow 0$
    while True do
        foreach $a_i(t) \in \mathcal{D}$ do
            Search its $\epsilon$-range neighborhood ($N_\epsilon(a_i)$) using Eq. 4
            foreach neighbor $b(t) \in N_\epsilon(a_i)$ do
                compute $a_i(t+1)$ using Eq. 5
            end
        end
        Compute the cluster order parameter $cr(t)$ using Eq. 6
        if $cr(t)$ converges then
            $\mathcal{G} \leftarrow$ SYNCHRONIZEDP($\mathcal{D}$)
            // Finding clusters
            Break
        end
        $t \leftarrow t+1$
    end
    foreach $\mathcal{G}_i \in \mathcal{G}$ do
        $mc \leftarrow \left(LS, SS, \mathcal{N}, \mathcal{W}, \mathcal{T}, CL\right)$
        // mc(micro-cluster) see Eq. 8
        $MCs \leftarrow MCs \cup mc$
    end
    return $MCs$ // set of micro-clusters
End Function
```

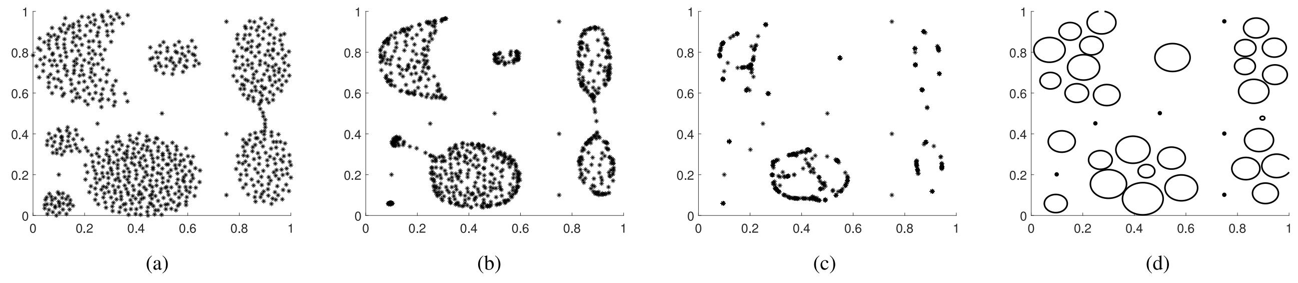


Fig. 1. Synchronization-based dynamic clustering process at different time steps. (a) Original dataset. (b) $T_3$. (c) $T_6$. (d) Final clusters.

After the process terminates, all the data points in the cluster have the same phase.

Algorithm 1 outlines the steps of synchronization-based clustering. In the beginning, each data point is considered a phase oscillator and has its own phase, that is, a feature vector. With time, each data point interacts with its range of neighborhood data points, according to (5). To ensure a stable interaction of each data point, the value of $\epsilon$ is determined from a sample of data points in the stream, as suggested in [6]. In particular, the value of $\epsilon$ is defined as an average of the $k$-nearest neighbor distance. In general, the value of $k$ is small (2–8), and this small range of interactions makes it possible to maintain the cluster structure for each block of data. At each time step, the phases of all data points in a local cluster structure gradually move to a common location. Thus, synchronization provides an intuitive way to represent and summarize the local data points. Finally, once the condition of the cluster order parameter (cr) is satisfied, the clustering process ends, and all the synchronized data points are recognized as a cluster. Fig. 1 illustrates the synchronization-based clustering process where similar data points form clusters after interacting with each other. The figure also shows that potential outliers retain their original values because they do not interact with other data points and can be easily identified.

After finding the clusters, a statistic summary [called cluster features (CFs)] for each cluster is calculated and stored in the form of a 6-tuple called microcluster (MC). The detail of each CF is given as follows:

$$\text{mc} = \begin{cases} \text{LS} = \sum_{i=1}^{n} x_i \\ \text{SS} = \sum_{i=1}^{n} (x_i)^2 \\ \mathcal{N} = n \\ \mathcal{W} = 1 \\ \mathcal{T} = \text{cT} \\ \text{CL} = \ell/\phi. \end{cases} \tag{8}$$

Here, LS and SS contain the linear and squared sum of all the cluster instances. $\mathcal{N}$ contains the total number ($n$) of instances in the cluster. $\mathcal{W}$ is the weight of the MC and is initially set to 1. $\mathcal{T}$ contains the current time (cT) at which the MC is created or updated. CL stores the class label $\ell$, provided that the cluster has a label point; otherwise, it is set to $\phi$.

*3) Model Update:* Learning algorithms should be modified/updated to adapt to evolving concepts in the data stream over time when the latest data is available. Thus, whenever the most recent data block is partially labeled, the existing model is updated to capture the current concept. Algorithm 2 describes the essential steps in making the model update-to-date. Here, the update procedure mainly includes the following steps.

*Step 1 (Error-Driven Representativeness Learning):* In this update procedure, labeled data points in the current data block are used to learn the representativeness of each MC that is involved in prediction. A supervised criterion is applied for updating the weight of each closest MC by checking the agreement between the real and the predicted label. Precisely, if an MC produces a correct prediction, its weight is increased and its time is updated while the weight of the wrongly predicted MC decreases (see lines 2–10 of Algorithm 2).

*Step 2 (Gradual Concept Drift Modeling):* In addition, the time effect is also a significant aspect to be taken into account in the context of mining streaming data. Over time, the learning model may contain obsolete or outdated concepts that need to be removed. Therefore, it is necessary to eliminate obsolete or old microclusters to make the model up to date with the recent trends (concepts) in the data stream. To gradually remove the old microclusters in the model, we use the exponential decay function. For each MC in the model, a weight is calculated using the exponential decay function, which is given by the following equation:

$$\mathcal{W}_{mc} \leftarrow \mathcal{W}_{mc} \times 2^{-\lambda \times eT}. \tag{9}$$

Here, $eT$ is the elapsed time of the MC from the last update, and $\lambda$ is a decay rate. The initial value of $\mathcal{W}$ is set to 1. With time, the microclusters' weight is gradually decreased (line 11 of Algorithm 2). However, the weight of microclusters is increased by one if they produce a correct prediction.

*Step 3 (Online Data Maintenance):* This step consists of microclusters removal, insertion, and merging.

1) *Microclusters Removal:* After error-driven representativeness and the effect of time, some MCs have negative or low weights. These microclusters must be eliminated to adapt to concept drift and make the model consistent with the current concept. Therefore, in line 12 of Algorithm 2, microclusters with weights approximately equal to zero or microclusters with negative weight are removed from the current model.
2) *Insertion of New Synchronization-Based Microclusters:* In any data stream learning algorithm, the most recent data are considered the most informative. Therefore, synchronization-based clustering is applied to create dynamic microclusters to summarize the current data block while maintaining the local data structure

**Algorithm 2:** Updating Model

```
1  Function UPDATE($Z, cT$)
       // Error driven representative learning
2      foreach labeled point $x \in Z$ do
3          [$p\_label, mc$] ← $model$.Predict($x$)
           // 1-NN (nearest labeled mc in the model)
4          if p_label matches true model then
5              Increment mc weight: $\mathcal{W}_{mc} \leftarrow \mathcal{W}_{mc} + 1$
6              Update time: $\mathcal{T}_{mc} \leftarrow cT$
7          else
8              Decrement mc weight: $\mathcal{W}_{mc} \leftarrow \mathcal{W}_{mc} - 1$
9          end
10     end
       // Weight of MCs over time. see Eq. 9
11     model.DecreaseWeights($cT$)
       // Remove outdated and low accurate MCs in model
12     model.Remove($\mathcal{W}_{mc} <\approx 0$)
       // Creating new MCs for recent block
13     $MCs_{new}$ ← SYNMICROCLUSTERS($Z$) // see Algorithm 1
14     if model.Size() = maxMC then
15         repeat
16             Find nearest labeled ($mc_i$) and unlabeled ($mc_j$) micro-cluster in the model (based on Euclidean distance)
               // see Eq. 10
17             if No unlabeled $mc_j$ exist then
18                 Find nearest largest class micro-clusters ($mc_i$) and ($mc_j$) in the model
                   // see Eq. 10
19             end
20             model.Merge($mc_i$,$mc_j$) // see Eq. 11
21         until space requirement if fullfiled
22     end
23     model.Append($MCs_{new}$)
24 End Function
```

(line 13). These new microclusters are more informative and represent current concepts.

3) *Merging Existing Microclusters:* To save memory space, the number of microclusters maintained must be limited so that the approach can also be implemented in smart devices with low virtual memory and low computing power. To keep the memory constant over time, a maximum boundary (i.e., maxMC) is set, which indicates the total number of microclusters that the model can maintain. After creating dynamic microclusters from the recent data block, the maximum model boundary is checked. If the model reaches its maximum boundary, then we need to create space for newly created microclusters. For this, existing nearest microclusters are merged in the model (lines 14–21). The process of merging existing microclusters is performed in two steps. First, if there are unlabeled microclusters in the model, their nearest labeled microclusters are searched and merged. Second, if there no unlabeled microclusters exist, then the following procedure is followed. A class that has the highest number of microclusters in the model is searched, and the closest microclusters of that class are merged. This process is repeated until space

**Algorithm 3:** Synchronization-Based Semisupervised Data Stream Classification

```
Input: DS: data stream i.e., $DS = \{B_1, B_2, \ldots, B_n, \ldots\}$; $D_{init}$: Initial training data; maxMC: maximum micro-clusters in the model
Output: Classification results: P_label
// Training Denoising Autoencoder DAE (See Section III-B1)
1  DAE← DAE.Train($D_{init}$)
   // $B_1$: first block of data
2  $Z$ ←DAE.Transform($B_1$)
   // Creating a set of micro-clusters (initial learning model)
3  model← SYNMICROCLUSTERS(Z) // see Algorithm 1
4  while Stream DS has instances do
5      Read($B_i$) // latest data block
6      $Z$ ←DAE.Transform($B_i$)
7      foreach data point $x \in Z$ do
           // 1-NN (nearest labeled mc in the model)
8          p_label ← model.Predict($x$)
9          Display output p_label
10         Save output: P_label.add(p_label)
11     end
       // When $B_i$ is partially labeled
12     $cT$ ← currentTime
13     model.Update($Z, cT$) // See Algorithm 2
14 end
```

for new clusters is created. The merging criteria for microclusters are given in the following equation:

$$\text{mcs} = \begin{cases} \operatorname{argmin}_{\substack{mc_i \in mc^{lb} \\ mc_j \in mc^{ul}}} \text{dist}(mc_i, mc_j) & : \text{If } mc^{ul} \neq \phi \\ \operatorname{argmin}_{\substack{mc_{i,j} \in mc^{lb^{\ell}} \\ i \neq j}} \text{dist}(mc_i, mc_j) & : \text{otherwise.} \end{cases} \tag{10}$$

Here, $mcs$ represents microcluster $mc_i$ and $mc_j$. $mc^{lb^{\ell}}$ is a set of labeled microclusters belonging to a specific class $\ell$. dist($mc_i$, $mc_j$) is the Euclidean distance between $mc_i$ and $mc_j$, where $mc^{ul}$ and $mc^{lb}$ are the set of unlabeled and labeled microclusters in the model, respectively. Once the closest microclusters are searched, they are merged as follows:

$$\text{mc}_m = \begin{cases} \text{LS}_m \leftarrow \text{LS}_{mc_i} + \text{LS}_{mc_j} \\ \text{SS}_m \leftarrow \text{SS}_{mc_i} + \text{SS}_{mc_j} \\ \mathcal{N}_m \leftarrow \mathcal{N}_{mc_i} + \mathcal{N}_{mc_j} \\ \mathcal{W}_m \leftarrow \max(\mathcal{W}_{mc_i}, \mathcal{W}_{mc_j}) \\ \mathcal{T}_m \leftarrow \max(\mathcal{T}_{mc_i}, \mathcal{T}_{mc_j}) \\ \text{CL}_m \leftarrow \text{CL}_{mc_i}. \end{cases} \tag{11}$$

Finally, newly created synchronization-based microclusters are added in the model (line 13 of Algorithm 2).

Finally, Algorithm 3 combines all parts and provides the pseudocode of our algorithms. The overall time complexity of the proposed algorithm is $\mathcal{O}(\text{maxMC} + S + \text{maxMC} + T.S^2 + \text{maxMC}^2) \approx \mathcal{O}(\text{maxMC} + T.S^2 + \text{maxMC}^2)$, where $S$ (block size) is the number of instances and $T$ is the number of iterations after the clustering process terminates. maxMC is the total number of maintained microclusters in the model. The space complexity of the model is given as follows. A single *mc* takes $\mathcal{O}(2 \times V_{1\times z} + 4)$ of space. Thus, the overall space complexity of the model is $\mathcal{O}(\text{maxMC} \times mc)$.

## IV. Experiments

In this section, we provide the details of datasets used in our empirical assessment, description, and list of competing algorithms for semisupervised data stream classification, and finally, the analysis of the obtained empirical results.

### A. Datasets

We have selected seven benchmark real-world datasets to assess and analyze the performance of all algorithms. All data sets are normalized to [0, 1]. The list and description of these datasets are given as follows.

*Spam*[1]*:* This dataset is provided by Spam Assassin and is a collection of e-mails, sorted by arrival time.

*Gas Sensor Array Drift (GSD)*[2]*:* This dataset is a collection of records of six different pure gaseous materials.

*Human Activity Recognition (HAR)*[2]*:* This dataset contains the records of six daily living activities (like walking, sitting, and laying) performed by 30 subjects.

*KDDcup99*[2]*:* This dataset is a collection of raw TCP dump data generated in MIT Lincoln Labs over a local network for nine weeks. Each TCP connection record is classified as a regular or attacked connection.

*Forest Cover Type (FCT)*[3]*:* This dataset contains different geographic measurements describing a type of forest cover on a cell measuring $30 \times 30$ m$^2$.

*IoT Botnet Attack (IoT)*[2]*:* This dataset consists of nine IoT devices traffic data that are attacked by two botnets.

*MNIST*[4]*:* It is a standard real-world handwritten digit's dataset. Finally, a summary of each dataset is given in Table I. Apart from these datasets, experiments on additional four datasets (two reals and two synthetics) are also conducted. Specifically, synthetic datasets are generated with different characteristics (with different types and drift rates) to analyze the effectiveness of the proposed algorithms. The detailed results on these datasets are presented in the supplementary material.

### B. Comparison Algorithms

To provide an in-depth performance analysis, we compare our algorithm with five state-of-the-art representative data stream SSL algorithms. In addition, four supervised data stream classification methods are also selected for a topline comparison. A brief description of these algorithms is given as follows.

*Semisupervised Methods:* Here, ReSSL [17], SPASC [13], TLP [15], CAL [20], and SFT [20] algorithms are selected for comparison. ReSSL, SPASC, and CAL are cluster-based models, while TLP is a graph-based model. Similarly, SFT is a self-training-based model.

[1]http://mlkd.csd.auth.gr/concept_drift.html

[2]https://archive.ics.uci.edu/ml/datasets.php

[3]https://moa.cms.waikato.ac.nz/datasets/

[4]http://yann.lecun.com/exdb/mnist/

TABLE I
Characteristics of Datasets Used for Classification Performance Evaluation. #F: Number of Features, #C: Number of Classes, and #Ins: Number of Instances

| Dataset | #F | #C | #Ins | Max Class % | Min Class % |
|---|---|---|---|---|---|
| **Spam** | 500 | 2 | 9,324 | 74.40 | 25.60 |
| **GSD** | 128 | 6 | 13,910 | 21.63 | 11.80 |
| **HAR** | 561 | 6 | 10,299 | 18.88 | 13.65 |
| **KDDcup** | 34 | 23 | 494,021 | 56.84 | 0.00 |
| **FCT** | 54 | 7 | 581,012 | 48.76 | 0.47 |
| **IoTBotnet** | 115 | 11 | 663,795 | 18.47 | 4.38 |
| **MNIST** | 784 | 10 | 70,000 | 11.25 | 9.14 |

*Supervised Methods:* Here, four state-of-the-art supervised methods SAMkNN [8], EMC [41], ARF [42], and AUE [43] are selected for comparison. EMC is a cluster-based single classifier, SAMkNN is an instance-based $k$-NN classification model, and ARF and AUE are ensemble models.

*Algorithm Settings:* We thank the authors of ReSSL, EMC, and SPASC for providing their paper source codes. The TLP implementation has been completed according to the details given in this article. SAMkNN, ARF, AUE, CAL, and SFT are available in a software framework called massive online analysis (MOA) [44]. For all competing algorithm default parameters settings, given in the corresponding papers are utilized. While parameters settings for the proposed model are given as follows: 1) maxMC $= 1000$; 2) $S$ (Block size) $= 1000$; and 3) $\lambda = 1e^{-4}$ (decay rate) is set in a way that after ten new consecutive blocks, if an MC is not used for prediction or not update then it is removed from the model; and 4) $k = 3$, to compute the value of $\epsilon$ which is defined as an average of $k$-nearest neighbor distance. Furthermore, parameters for DAE are set as follows: 1) $\sigma^2$ (additive Gaussian noise) $= 1.1$ and 2) hidden layers $= 3$ ($h_1 = (2/3)d, h_2 = (1/3)h_1, z = h_3 = (1/3)h_2$) for datasets with $d$ (input features) $\geq 500$, and hidden layers $= 2$ ($h_1 = (2/3)d, z = h_2 = (1/3)h_1$), for datasets with $d < 500$, where $h_i$ is the number of neurons in the hidden layer $i$ and $z << d$ is the final reduced features.

### C. Analysis of Results

In this section, we present a detailed analysis of all experiments performed on all datasets. A complete discussion and comparative analysis of the proposed algorithm against several state-of-the-art algorithms are presented. Finally, in the end, we report the parameter sensitivity analysis of the proposed algorithm with respect to different parameters.

*1) Performance Comparison:* We use two measures (Accuracy and F1-score) to assess the predictive performance of all competing algorithms. In our experiments, we adopted the chunk-based evaluation technique [43] to compute the performance measures over the stream. This technique is similar to the interleaved test-then-train evaluation technique [42] where each data point is first used to test the classifier and then used to update the classifier. In the chunk-based evaluation technique, incoming data points are accumulated in a

TABLE II
AVERAGE CLASSIFICATION ACCURACY ACHIEVED BY COMPETING ALGORITHMS ON DIFFERENT DATASETS. THE BOLD FONT INDICATES BEST RESULT

| Datasets | Label % | Our | ReSSL | SPASC H | SPASC B | TLP | CAL | ST |
|---|---|---|---|---|---|---|---|---|
| **Spam** | 5 | **93.28** | 80.97 | 83.74 | 87.62 | 67.58 | 84.30 | 85.06 |
| | 10 | **93.93** | 83.77 | 83.73 | 87.99 | 65.13 | 86.21 | 86.82 |
| | 20 | **94.61** | 86.08 | 83.88 | 84.90 | 70.47 | 86.20 | 86.45 |
| **GSD** | 5 | **88.14** | 74.36 | 51.87 | 58.81 | 57.62 | 57.17 | 56.33 |
| | 10 | **90.14** | 82.17 | 53.13 | 55.52 | 65.49 | 58.88 | 58.84 |
| | 20 | **93.14** | 88.64 | 58.11 | 58.11 | 72.60 | 58.20 | 60.00 |
| **HAR** | 5 | **84.48** | 79.90 | 75.16 | 74.64 | 57.41 | 34.88 | 35.00 |
| | 10 | **86.57** | 84.50 | 68.60 | 68.66 | 59.27 | 36.39 | 36.70 |
| | 20 | **87.33** | 87.05 | 70.83 | 70.83 | 62.84 | 36.68 | 37.13 |
| **KDDcup** | 5 | 98.24 | 97.99 | 94.06 | 93.40 | 76.88 | **98.75** | 95.50 |
| | 10 | **98.55** | 98.33 | 94.58 | 93.06 | 78.26 | 96.04 | 96.34 |
| | 20 | **98.79** | 98.61 | 90.54 | 89.57 | 78.88 | 96.74 | 97.04 |
| **FCT** | 5 | 79.58 | **81.84** | 58.62 | 57.34 | 72.03 | 68.23 | 62.22 |
| | 10 | 81.63 | **83.44** | 62.58 | 58.91 | 79.93 | 69.66 | 70.15 |
| | 20 | 83.29 | 85.05 | 64.15 | 61.88 | **85.40** | 71.32 | 71.95 |
| **IoTBotnet** | 5 | **97.70** | 66.28 | 84.57 | 85.42 | 89.05 | 68.69 | 67.03 |
| | 10 | **98.63** | 91.54 | 90.46 | 87.98 | 89.96 | 76.21 | 76.14 |
| | 20 | **99.17** | 98.11 | 78.48 | 78.40 | 91.24 | 81.63 | 85.46 |
| **MNIST** | 5 | **79.15** | 72.70 | 57.29 | 65.30 | 39.86 | 66.52 | 71.61 |
| | 10 | **84.17** | 79.88 | 62.16 | 70.50 | 40.06 | 74.32 | 70.80 |
| | 20 | **87.65** | 85.06 | 60.32 | 65.85 | 40.27 | 72.53 | 73.53 |
| ***Avg Accuracy*** | | **90.39** | 85.06 | 72.71 | 74.03 | 68.58 | 70.45 | 70.48 |
| ***Avg Rank*** | | **1.24** | 2.62 | 5.29 | 4.90 | 4.95 | 4.57 | 4.33 |

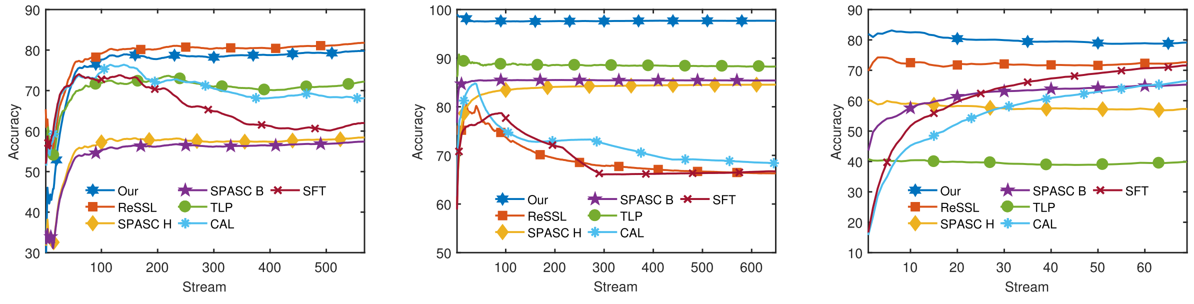


Fig. 2. Classification accuracy achieved by competing algorithms with 5% labeled data over the entire stream. (a) FCT dataset. (b) IoTBotnet dataset. (c) MNIST dataset.

data chunk of size $S$ without processing them. After forming a data chunk, it is first used to test the model and then update it. Initially, every classifier is trained on the first data chunk; afterward, accumulated classification accuracy and F1-score are computed over the entire stream data. We repeat all the experiments ten times, and the average accuracy and F1-score are calculated. We have selected both semisupervised and supervised data stream algorithms described in Section IV-B to deeply analyze the predictive performance. For SSL algorithms, after classifying every incoming data block, $p\%$ (5%, 10%, and 20% in our experiments) data points are randomly labeled and given to the algorithms to refine or update their models. For supervised algorithms, actual class labels of the entire data block are provided to update the classifiers.

*a) Comparison with semisupervised algorithms:* In this section, we first provide a comparative analysis of the proposed algorithm against five state-of-the-art data stream SSL algorithms. We report the average classification accuracy and the F1-score in Tables II and III, respectively, for all datasets with different percentages of label data. Here, the best performance achieved is presented in bold font.

From Table II, we can see that our algorithm shows better prediction performance than other competing algorithms. Our algorithm achieves an overall average classification accuracy of 90% on all datasets and is ranked top among different algorithms. The classification performance obtained with our algorithm is 5% and 16% superior to the second and third best algorithms, respectively. Similarly, we can see from Table III that our algorithm maintains high F1-scores compared to

TABLE III
AVERAGE F1-SCORE ACHIEVED BY COMPETING ALGORITHMS ON DIFFERENT DATASETS. THE BOLD FONT INDICATES BEST RESULT

| Data sets | Label % | Our | ReSSL | SPASC H | SPASC B | TLP | CAL | ST |
|---|---|---|---|---|---|---|---|---|
| **Spam** | 5 | **86.36** | 47.41 | 67.90 | 76.27 | 61.58 | 78.23 | 78.82 |
| | 10 | **87.95** | 59.92 | 69.10 | 76.11 | 58.91 | 81.24 | 82.15 |
| | 20 | **89.26** | 67.78 | 73.57 | 68.82 | 66.75 | 81.13 | 81.50 |
| **GSD** | 5 | **87.27** | 72.92 | 50.76 | 58.37 | 56.88 | 57.26 | 54.95 |
| | 10 | **89.13** | 81.16 | 50.72 | 53.49 | 64.87 | 57.69 | 57.23 |
| | 20 | **92.58** | 87.84 | 58.12 | 58.12 | 71.62 | 56.79 | 58.09 |
| **HAR** | 5 | **84.66** | 79.74 | 74.49 | 74.01 | 49.54 | 32.32 | 31.68 |
| | 10 | **86.88** | 84.37 | 67.83 | 67.87 | 51.92 | 33.33 | 33.51 |
| | 20 | **87.58** | 87.01 | 70.82 | 70.82 | 54.52 | 33.48 | 33.99 |
| **KDDcup** | 5 | 23.55 | 17.86 | 24.68 | 24.34 | 7.79 | **37.23** | 30.25 |
| | 10 | 27.78 | 21.22 | 21.15 | 27.79 | 7.96 | 34.75 | **35.36** |
| | 20 | 32.57 | 25.47 | 19.14 | 15.23 | 8.13 | 34.23 | **36.90** |
| **FCT** | 5 | 67.86 | **71.92** | 45.18 | 42.91 | 58.22 | 55.95 | 49.05 |
| | 10 | 69.79 | **74.88** | 47.76 | 40.86 | 64.25 | 57.44 | 57.25 |
| | 20 | 72.22 | **77.01** | 51.62 | 45.04 | 75.70 | 58.22 | 59.78 |
| **IoTBotnet** | 5 | **98.20** | 62.63 | 85.00 | 85.62 | 88.35 | 69.51 | 65.53 |
| | 10 | **98.90** | 90.84 | 90.84 | 89.17 | 90.70 | 76.66 | 77.10 |
| | 20 | **99.29** | 98.31 | 79.98 | 79.94 | 92.34 | 80.37 | 84.99 |
| **MNIST** | 5 | **78.79** | 72.16 | 55.60 | 64.27 | 39.18 | 66.14 | 71.66 |
| | 10 | **83.89** | 79.60 | 61.57 | 69.88 | 39.30 | 74.30 | 70.52 |
| | 20 | **87.48** | 84.90 | 60.38 | 65.40 | 39.12 | 72.38 | 73.54 |
| ***Avg F1-Score*** | | **77.71** | 68.81 | 58.39 | 59.73 | 54.65 | 58.51 | 58.28 |
| ***Avg Rank*** | | **1.62** | 3.14 | 5.00 | 4.76 | 5.00 | 4.29 | 4.10 |

others in most datasets. In summary, the reported results demonstrate the effectiveness of the proposed approach compared to other algorithms. The proposed algorithm, which is equipped with dynamic clustering and error-driven representativeness learning, makes effective use of both labeled and unlabeled data to improve the predictive performance in the concept drifting streaming data. It is also important to note that our algorithm specifically outperforms others when the labeled data is limited.

In addition to analyzing the average classification accuracy of the competing algorithms, we have also generated graphical plots (see Fig. 2) for some datasets to show the performance of all approaches over the entire stream.

We also use the Friedman rank-sum test to check the statistical significance between competing algorithms. For this, we define the null hypothesis as "no significant difference among comparing algorithms." If the null hypothesis is rejected, we further used the Nemenyi *post-hoc* test to find these differences. The Friedman rank-sum test is a nonparametric statistical test and we perform this test on Tables II and III with a 95% confidence level and obtain a $p$-value $= 2.63e^{-11}$ and $= 2.12e^{-7}$, respectively, for all datasets. The obtained $p$-value is less than the standard threshold (0.05), which confirms that our approach statistically differs from other competing algorithms. Based on the rejection of the null hypothesis, we apply the Nemenyi *post-hoc* test to build a critical difference diagram (see Fig. 3) to identify the difference between the algorithms. Although the critical difference score of ReSSL resembles our algorithm, the main drawback of ReSSL is that it requires

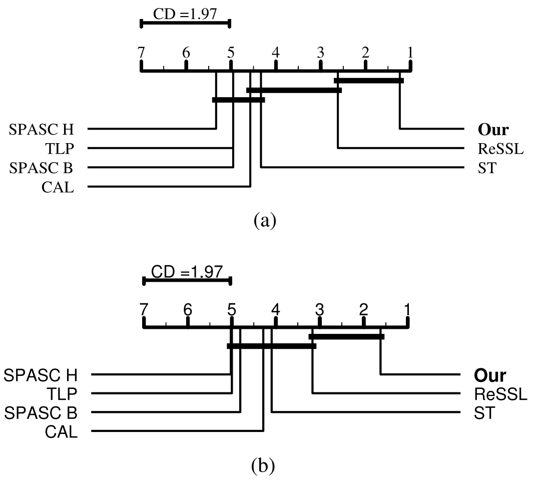
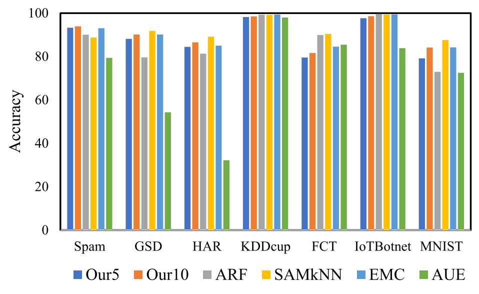


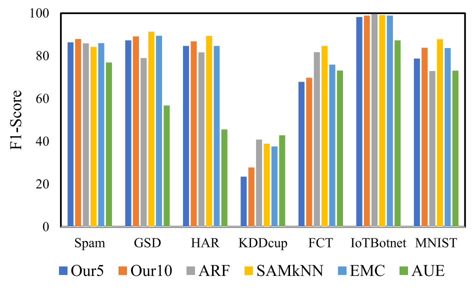
Fig. 3. Critical difference diagram using the Nemenyi test with a 95% confidence level for different semisupervised algorithms on all datasets. (a) Accuracy (Table II). (b) F1-Score (Table III).

immediate true labels, which is not a realistic assumption in many practical applications.

*b) Comparison with supervised algorithms:* In order to provide a topline, in this experiment, we also compare our algorithm against four state-of-the-art supervised algorithms, including ARF, EMC, AUE, and SAMkNN, which require the data stream to be fully labeled to update or refine their models. We test the performance of the proposed algorithm with 5%, and 10% (called *Our*5 and *Our*10) labeled data against these four algorithms. We report the overall average performance



(a)



(b)

Fig. 4. (a) Average accuracy and (b) F1-score comparison with supervised algorithms on all datasets.

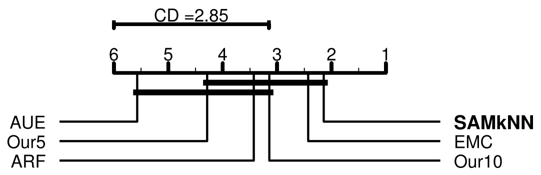
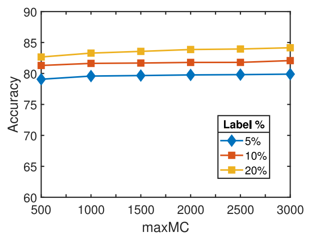


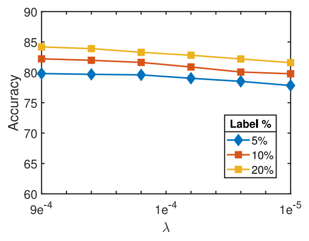
Fig. 5. Critical difference diagram using the Nemenyi test for supervised algorithms on all datasets presented in Fig. 4.

in terms of accuracy and F1-score obtained on all datasets in Fig. 4. From the figure, we can observe that the proposed algorithm with different percentages of labeled data achieves competitive predictive performance compared to supervised algorithms using 100% of labeled data. On the Spam dataset, the proposed algorithm outperforms these algorithms with 5% of the labeled data. While with 10% of the data labeled on other datasets, the performance matches the supervised algorithms. This analysis demonstrates that the proposed algorithm offers superior classification performance with limited labeled data compared to various semisupervised algorithms. With a higher percentage (10% or 20%) of labeled data, competitive performance can be obtained similar to supervised algorithms. Thus, our algorithm reduces the dependency on labeled data and provides an efficient and effective technique to mine the concept drifting data streams. We also perform the Friedman rank-sum test with the Nemenyi *post-hoc* test to build a critical difference diagram to identify the difference between the supervised algorithms. Fig. 5 shows that there is no significant difference between the competing algorithms.

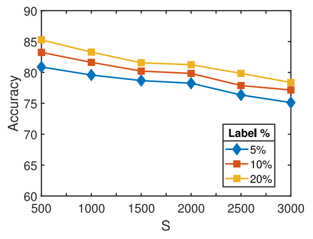
*2) Parameter Sensitivity Analysis:* In order to analyze the impact of different parameters on prediction performance, we perform several experiments on FCT (see Fig. 6) and MNIST (see Fig. 7) datasets.



(a)

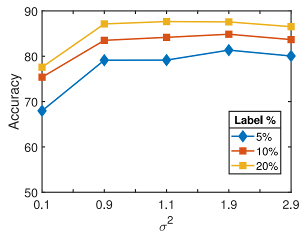


(b)

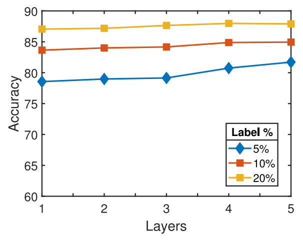


(c)

Fig. 6. Impact of different parameters on accuracy of the FCT dataset. (a) Maximum number of microclusters. (b) Decay rate. (c) Block size.



(a)



(b)

Fig. 7. Impact of DAE parameters on accuracy of the MINST dataset. (a) Noise rate. (b) Number of layers.

Fig. 6(a) shows the impact of parameter maxMC on classification accuracy. This parameter represents the maximum number of microclusters that can be stored in the model. We perform experiments on different values of maxMC ranging from 500 to 3000. From the figure, we can see that maintaining a high number of microclusters in the model increase the classification accuracy. The reason behind this is that with a higher number of microclusters, the model can accurately capture the underlying data distribution (low/high density of data). Nonetheless, maintaining a large number of microclusters increases resource consumption (memory and computational time).

Fig. 6(b) examines the impact of parameter $\lambda$ (decay rate) on prediction performance with respect to different values of $\lambda$ varying from $9e^{-4}$ to $1e^{-5}$. This parameter implements the forgetting mechanism (time effect) in the model. With a high value of $\lambda$, the model quickly removes the outdated microclusters, which are not updated over time. From the figure, we can see that the performance gradually decreases with the decreasing value of $\lambda$. Due to slow forgetting, the model may contain obsolete concepts that cause performance degradation.

Fig. 6(c) analyzes the impact of parameter $S$ (block size) on classification accuracy with different values ranging from 500 to 3000. From the figure, we can observe that a small block size produces the best result while with a larger block size, the performance gradually decreases. The potential reason for this is that the model has been outdated for a long time and is, therefore, unable to capture recent evolving concepts.

Similarly, we perform experiments with the MNIST dataset to study the effect of DAE parameters on classification performance. Fig. 7 shows the effect of $\sigma^2$ (noise rate) and the number of hidden layers on the precision. In the figure, we can see that higher values of additive Gaussian noise and a higher number of layers extract more robust hidden features, which yields good classification performance.

## V. Conclusion

Most existing approaches focus on developing effective and efficient supervised data streaming learning algorithms. However, labeling data in fast and continuous streaming is an expensive and often impractical task in many real-time applications. Therefore, to overcome the labeling task's difficulty, in this article, we present a new SSL technique for streaming data. The primary purpose of our proposed algorithms is to leverage both labeled and unlabeled data to enhance the classification performance of the learning algorithm. In the proposed algorithm, we combine several techniques to provide effective SSL on concept drifting data streams. First, to tackle the high data dimensionality, we use DAE to convert the original input feature space into a smaller and more compact (deeper) feature space. Second, we use synchronization-based dynamic clustering and maintain a set of microclusters to summarize and classify the incoming data. In addition, effective online data maintenance with an error-driven method provides a better way to deal with the concept drift than other competing methods. Finally, we conducted in-depth experiments with various real-world datasets. The reported results indicate that the proposed algorithm allows for a significant improvement in performance compared to existing state-of-the-art algorithms. Future research will aim to propose a drift detection method based on labeled and unlabeled data to react to concept drift quickly in the case of higher drift rates. Also, we would like to explore efficient ways to update the DAE weights during the stream progress incrementally.

## References

[1] P. Li *et al.*, "Learning from short text streams with topic drifts," *IEEE Trans. Cybern.*, vol. 48, no. 9, pp. 2697–2711, Sep. 2018.

[2] H. M. Gomes, J. Read, A. Bifet, J. P. Barddal, and J. Gama, "Machine learning for streaming data: State of the art, challenges, and opportunities," *ACM SIGKDD Explor. Newslett.*, vol. 21, no. 2, pp. 6–22, 2019.

[3] T. S. Sethi and M. Kantardzic, "On the reliable detection of concept drift from streaming unlabeled data," *Exp. Syst. Appl.*, vol. 82, pp. 77–99, Oct. 2017.

[4] P. Duda, L. Rutkowski, M. Jaworski, and D. Rutkowska, "On the parzen kernel-based probability density function learning procedures over time-varying streaming data with applications to pattern classification," *IEEE Trans. Cybern.*, vol. 50, no. 4, pp. 1683–1696, Apr. 2020.

[5] J. a. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Comput. Surveys*, vol. 46, no. 4, pp. 1–44, 2014.

[6] 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.

[7] L. Chi, B. Li, X. Zhu, S. Pan, and L. Chen, "Hashing for adaptive real-time graph stream classification with concept drifts," *IEEE Trans. Cybern.*, vol. 48, no. 5, pp. 1591–1604, May 2018.

[8] 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, 2018.

[9] Z. Yu *et al.*, "Adaptive semi-supervised classifier ensemble for high dimensional data classification," *IEEE Trans. Cybern.*, vol. 49, no. 2, pp. 366–379, Feb. 2019.

[10] C.-L. Liu, W.-H. Hsaio, C.-H. Lee, T.-H. Chang, and T.-H. Kuo, "Semi-supervised text classification with universum learning," *IEEE Trans. Cybern.*, vol. 46, no. 2, pp. 462–473, Feb. 2016.

[11] X. J. Zhu, "Semi-supervised learning literature survey," Dept. Comput. Sci., Univ. Wisconsin-Madison, Madison, WI, USA, Rep. TR 1530, 2005.

[12] M. M. Masud, J. Gao, L. Khan, J. Han, and B. Thuraisingham, "A practical approach to classify evolving data streams: Training with limited amount of labeled data," in *Proc. 8th IEEE Int. Conf. Data Mining*, 2008, pp. 929–934.

[13] M. J. Hosseini, A. Gholipour, and H. Beigy, "An ensemble of cluster-based classifiers for semi-supervised classification of non-stationary data streams," *Knowl. Inf. Syst.*, vol. 46, no. 3, pp. 567–597, 2016.

[14] X. Wu, P. Li, and X. Hu, "Learning from concept drifting data streams with unlabeled data," *Neurocomputing*, vol. 92, pp. 145–155, Sep. 2012.

[15] T. Wagner, S. Guha, S. Kasiviswanathan, and N. Mishra, "Semi-supervised learning on data streams via temporal label propagation," in *Proc. 35th Int. Conf. Mach. Learn.*, vol. 80, 2018, pp. 5095–5104.

[16] S. Khezri, J. Tanha, A. Ahmadi, and A. Sharifi, "STDS: Self-training data streams for mining limited labeled data in non-stationary environment," *Appl. Intell.*, vol. 50, pp. 1448–1467, Jan. 2020.

[17] J. Shao, C. Huang, Q. Yang, and G. Luo, "Reliable semi-supervised learning," in *Proc. IEEE 16th Int. Conf. Data Mining*, 2016, pp. 1197–1202.

[18] P. Zhang, X. Zhu, J. Tan, and L. Guo, "Classifier and cluster ensembles for mining concept drifting data streams," in *Proc. 10th IEEE Int. Conf. Data Mining*, 2010, pp. 1175–1180.

[19] M. M. Masud *et al.*, "Facing the reality of data stream classification: Coping with scarcity of labeled data," *Knowl. Inf. Syst.*, vol. 33, no. 1, pp. 213–244, 2012.

[20] M.-H. Le Nguyen, H. M. Gomes, and A. Bifet, "Semi-supervised learning over streaming data using MOA," in *Proc. IEEE Int. Conf. Big Data*, 2019, pp. 553–562.

[21] S. U. 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.

[22] T. S. Sethi, M. M. Kantardzic, and H. Hu, "A grid density based framework for classifying streaming data in the presence of concept drift," *J. Intell. Inf. Syst.*, vol. 46, no. 1, pp. 179–211, 2016.

[23] L. Tu and Y. Chen, "Stream data clustering based on grid density and attraction," *ACM Trans. Knowl. Discov. Data*, vol. 3, no. 3, pp. 1–27, 2009.

[24] L. Zhu, S. Pang, A. Sarrafzadeh, T. Ban, and D. Inoue, "Incremental and decremental max-flow for online semi-supervised learning," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 8, pp. 2115–2127, Aug. 2016.

[25] Y.-F. Li, S.-B. Wang, and Z.-H. Zhou, "Graph quality judgement: A large margin expedition," in *Proc. 25th Int. Joint Conf. Artif. Intell.*, 2016, pp. 1725–1731.

[26] Y. Wang and T. Li, "Improving semi-supervised co-forest algorithm in evolving data streams," *Appl. Intell.*, vol. 48, no. 10, pp. 3248–3262, 2018.

[27] E. Lughofer, "On-line active learning: A new paradigm to improve practical useability of data stream modeling methods," *Inf. Sci.*, vols. 415-416, pp. 356–376, Nov. 2017.

[28] M. Li and Z.-H. Zhou, "Improve computer-aided diagnosis with machine learning techniques using undiagnosed samples," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 37, no. 6, pp. 1088–1098, Nov. 2007.

[29] P. Lindstrom, S. J. Delany, and B. M. Namee, "Handling concept drift in a text data stream constrained by high labelling cost," in *Proc. 23rd Int. Florida Artif. Intell. Res. Soc. Conf.*, 2010, pp. 32–37.

[30] E. Lughofer, E. Weigl, W. Heidl, C. Eitzinger, and T. Radauer, "Recognizing input space and target concept drifts in data streams with scarcely labeled and unlabelled instances," *Inf. Sci.*, vols. 355–356, pp. 127–151, Aug. 2016.

[31] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *J. Mach. Learn. Res.*, vol. 11, pp. 3371–3408, Dec. 2010.

[32] J. Deng, Z. Zhang, F. Eyben, and B. Schuller, “Autoencoder-based unsupervised domain adaptation for speech emotion recognition,” *IEEE Signal Process. Lett.*, vol. 21, no. 9, pp. 1068–1072, Sep. 2014.

[33] J. Xu *et al.*, “Stacked sparse autoencoder (SSAE) for Nuclei detection on breast cancer histopathology images,” *IEEE Trans. Med. Imag.*, vol. 35, no. 1, pp. 119–130, Jan. 2015.

[34] J. Chen, S. Sathe, C. Aggarwal, and D. Turaga, “Outlier detection with autoencoder ensembles,” in *Proc. SIAM Int. Conf. Data Min.*, 2017, pp. 90–98.

[35] J. Castellini, V. Poggioni, and G. Sorbi, “Fake Twitter followers detection by denoising autoencoder,” in *Proc. Int. Conf. Web Intell.*, 2017, pp. 195–202.

[36] J. Xie, R. Girshick, and A. Farhadi, “Unsupervised deep embedding for clustering analysis,” in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 478–487.

[37] Y. Bengio, A. Courville, and P. Vincent, “Representation learning: A review and new perspectives,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 8, pp. 1798–1828, Aug. 2013.

[38] D. Charte, F. Charte, S. García, M. J. del Jesus, and F. Herrera, “A practical tutorial on autoencoders for nonlinear feature fusion: Taxonomy, models, software and guidelines,” *Inf. Fusion*, vol. 44, pp. 78–96, Nov. 2018.

[39] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, “Extracting and composing robust features with denoising autoencoders,” in *Proc. 25th Int. Conf. Mach. Learn.*, 2008, pp. 1096–1103.

[40] C. Böhm, C. Plant, J. Shao, and Q. Yang, “Clustering by synchronization,” in *Proc. 16th ACM SIGKDD Int. Conf. Knowl. Disc. Data Mining*, 2010, pp. 583–592.

[41] S. U. Din and J. Shao, “Exploiting evolving micro-clusters for data stream classification with emerging class detection,” *Inf. Sci.*, vol. 507, pp. 404–420, Jan. 2020.

[42] H. M. Gomes *et al.*, “Adaptive random forests for evolving data stream classification,” *Mach. Learn.*, vol. 106, no. 9, pp. 1469–1495, 2017.

[43] D. Brzezinski and J. Stefanowski, “Reacting to different types of concept drift: The accuracy updated ensemble algorithm,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, no. 1, pp. 81–94, Jan. 2014.

[44] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer, “MOA: Massive online analysis,” *J. Mach. Learn. Res.*, vol. 11, pp. 1601–1604, Apr. 2010.

**Salah Ud Din** received the Ph.D. degree from the University of Electronic Science and Technology of China, Chengdu, China, in 2020.

He is currently serving as an Assistant Professor with the Department of Computer, COMSATS University Islamabad, Islamabad, Pakistan. His research interests focus on data stream mining, especially on data stream classification, novel class detection, and semi-supervised learning.

**Jay Kumar** received the master’s degree from Quaid-i-Azam University, Islamabad, Pakistan, in 2018. He is currently pursuing the Ph.D. degree with the Data Mining Lab, School of Computer Science and Engineering, University of Electronics Science and Technology of China, Chengdu, China.

His main research interests include text mining, data stream mining, and natural language processing.

**Junming Shao** received the Ph.D. degree (*summa cum laude* with Highest Hons.) from the University of Munich, Munich, Germany, in 2011.

He became the Alexander von Humboldt Fellow in 2012. He is currently a Professor of Computer Science with the University of Electronic Science and Technology of China, Chengdu, China. He has not only published papers on top-level data mining conferences, such as KDD, ICDM, SDM, and IEEE/TKDE, but also published data mining-related interdisciplinary work in leading journals, including *Brain*, *Neurobiology of Aging*, and *Water Research*. His research interests include data mining and machine learning, especially for clustering in high-dimensional data, graph mining, and brain structural network analysis.

**Cobbinah Bernard Mawuli** received the master’s degree from the University of Electronic Science and Technology of China, Chengdu, China, in 2019, where he is currently pursuing the Ph.D. degree with the School of Computer Science.

His main research interests include transfer learning, federated learning, and neuroimaging analysis.

**Waldiodio David Ndiaye** received the master’s degree in computer science from Thiès University Senegal, Thiès, Senegal, in 2018. He is currently pursuing the Ph.D. degree with the Data Mining Lab, School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu, China.

His research interest focuses on novel natural language processing techniques for fake news/rumor detection on social media and the web in general.